UNIVERSITY OF SOUTHAMPTON

An Investigation of High Level Synthesis for Computational Hardware

by

Arash Ahmadi

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

December 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Arash Ahmadi

Using digital systems for numerical computation has a wide range of applications in engineering and science. In most of the cases, employing general purpose computers is a commonly accepted way to deploy an application, however, general purpose hardware does not provide the best solution when there are implementation restrictions, such as embedded systems, or the application requires very high performance calculations. On the other hand, in result of recent achievements in custom computing and hardware synthesis methods, it is possible to customise hardware to the applications. It means that the computing features of the underlying hardware can be configured to provide the best match with the application requirements. This thesis is concerned with the development and validation of a specific application high level synthesis methods to implement computational algorithms in digital hardware. Special emphasis is placed upon datapath structure and accuracy analysis and optimisation between computational error and hardware implementation costs.

The first part of the thesis concentrates on the high level synthesis of computational algorithms. We investigate how the arithmetic characteristics of the hardware affects implementation costs. In the following, the problem of choosing different arithmetic characteristics for each functional unit considering circuit cost is addressed. We use a symbolic method of computational error analysis in which the computational errors are characterised as sets of statistical symbols.

The proposed accuracy modelling is applied to a new architecture for datapath synthesis. Hierarchical target architecture is proposed which is built on a multiple-width multiple-way partitioned bus which provides the capability of parallel implementation of computational algorithms as well as flexibility in synthesis and optimisation. This architecture is based on an extension to the shared bus structure which is built by connecting a set of series and parallel bus segments building up a communication medium for all functional units. In combination with the multiple word-length design approach, each bus segment, and all the connected functional units to it, are allocated to a distinct bit-width.

Results demonstrate that by customising the structure and building blocks of computational intensive datapaths, savings can be made in the overall area, delay, and power consumption of a hardware implementation. This approach represents an improvement when compared with the previous work in this field and provides more understanding of implementing the computational algorithms into the customised hardware.

Contents

List of Figures

List of Tables

Nomenclature

AA	Affine Arithmetic
ASIC	Application-Specific Integrated Circuit
AE	Algorithm Executer
ALAP	As Late As Possible
ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application Specified Integrated Circuit
ASM	Algorithmic State Machine
BSW	Bus Switch
CAD	Computer Aided Design
CDFG	Control Data Flow Graph
CIH	Computationally Intensive Hardware
CMOS	Complementary MetalOxideSemiconductor
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DFG	Data Flow Graph
DFT	Discrete Fourier Transform
DIF	Dissemination In Frequency
DIT	Dissemination In Time
DSP	Digital Signal Processing
DST	Discrete Sine Transform
DWT	Discrete Wavelet Transform
EDA	Electronic Design Automation
FBDB	Functional Block Data Base
FDS	Force Directed Scheduling
FFC	Floating-point to Fixed-point Conversion
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
FSM	Finite State Machine
GA	Genetic Algorithm
HDL	Hardware Description Language

HLS	High Level Synthesis
IA	Interval Arithmetic
ICD	Intermediate Code
$IDCT$	Inverse Discrete Cosine Transform
IIR	Infinite Impulse Response
ILP	Integer Linear Programming
IP	Intellectual Property
LCM	Least Common Multiple
LP	Linear Programming
LSB	Least Significant Bit
LTI	Linear Time Invariant
MAC	Multiplication Accumulation
MC	Macro Cell
MOO	Multi Objective Optimisation
MOP	Multi Objective Programming
MSB	Most Significant Bit
$MWP - Bus$	Multiple Width Partitioned Bus
$MW^2P - Bus$	Multiple Width Multiple Way Partitioned Bus
PDF	Probability Density Function
RAM	Random Access Memory
RNG	Random Number Generator
RNS	Residue Number Systems
ROM	Read Only Memory
RTL	Register Transfer Level
SA	Simulated Annealing
SNA	Symbolic Noise Analysis
SoC	System On Chip
SP	Signal Processing
ULP	Unit in the Last Place
$VEGA$	Vector Evaluated Genetic Algorithm
$VHDL$	Very high speed integrated circuits Hardware Description Language
$VLSI$	Very Large Scale Integrated circuit
WHT	Walsh Hadamard Transform

Acknowledgements

I would like to express my gratitude to Professor Mark Zwolinski for his time, constructive advice, and willingness to share his insight and wisdom. An attentive reader who consistently offered a fresh perspective on the work, his efforts on my behalf has helped me to become a better scholar.

I also would like to thank all the faculty and staff in the Electronics Systems and Devices (ESD) Group for their continued support and a warm atmosphere in which to work. Specifically Professor Bashir M. Al-Hashimi who enthusiastically was ready to discus and give his encouraging advice regarding my work. I also owe to Dr. Peter Wilson and Dr. Jeff Reeves for their constructive discussions regarding my progress reports and thesis. I am also thankful to my dear colleagues Abdolbaghi Rezazadeh, Mehdi Jafaripanah, Biswajit Mishra, Karthik Baddam, Noohul Basheer Zain Ali, Tack Boon Yee, Andrew Chapman, Kosala Amarasinghe, Marco Ochoa-Montiel and Donald Esrafili-Gerdeh for their helps and omnipresent friendship to establish an inspiring cross-cultural working atmosphere.

Last, but most important, I have to and I wish to acknowledge the people who are the most closest to me. I cannot even imagine that without helps of my wife, who supported me by all her means, it was possible for me to carry on with this work during the all tough times which we had here far from our relatives. She heroically carried the burden of our dear children. I cannot leave this acknowledgement without saying thanks to all my family in back home specifically my mum and dad who have devoted their life for my success and their prayers have been the greatest support.

Declaration

I declare that the work in this thesis is entirely my own unless otherwise stated.

Arash Ahmadi

Chapter 1

Introduction

1.1 Motivation

Electronic Design Automation (EDA) has a vital position in many challenges of electronic market demands, where high complexity, low cost, high speed and low power consumption are usual requirements. High Level Synthesis (HLS) has a key role in design automation by trying to fill the big gap between high level descriptions of the system requirements and low level specifications of the electronic system in specialised hardware languages like Register Transfer Level (RTL) Hardware Description Languages (HDLs). Accordingly, significant attention has been paid to HLS in academia and industry during the past decade which resulted in enrichment of the research community in terms of literature, tools, theories and methodologies. Figure (1.1) shows the basic work flow in HLS, which starts form an algorithm level specification of the design and results in the physical implementation. The algorithm level specifications are normally presented in the form of mathematical equations which need to be translated to a system level specification. As it is depicted in this figure, this system level specification should be divided into software and hardware parts both of which have a different synthesis and verification procedures, however after all they must be combined to work together. Testbenches in Figure (1.1) are testing routines for each level of specification to verify that the specification. It is observable that there are diversity of the problems which must be solved in HLS, which range from physical aspects of the circuit implementation to pure mathematical analysis and optimisation methods, has built up a scene of very widespread problems to be inspected. To tackle this complication, HLS has been categorised in different ways. From the designers' point of view, according to the major problems which have to be dealt with, it has been divided into: low power HLS, co-design HLS, parallel-design HLS, SOC HLS and so on. From another point of view, HLS methods can be classified based on the application domain of the target designs, Signal Processing (DSP) HLS for instance. The main focus of this study is Computational Hardware HLS.

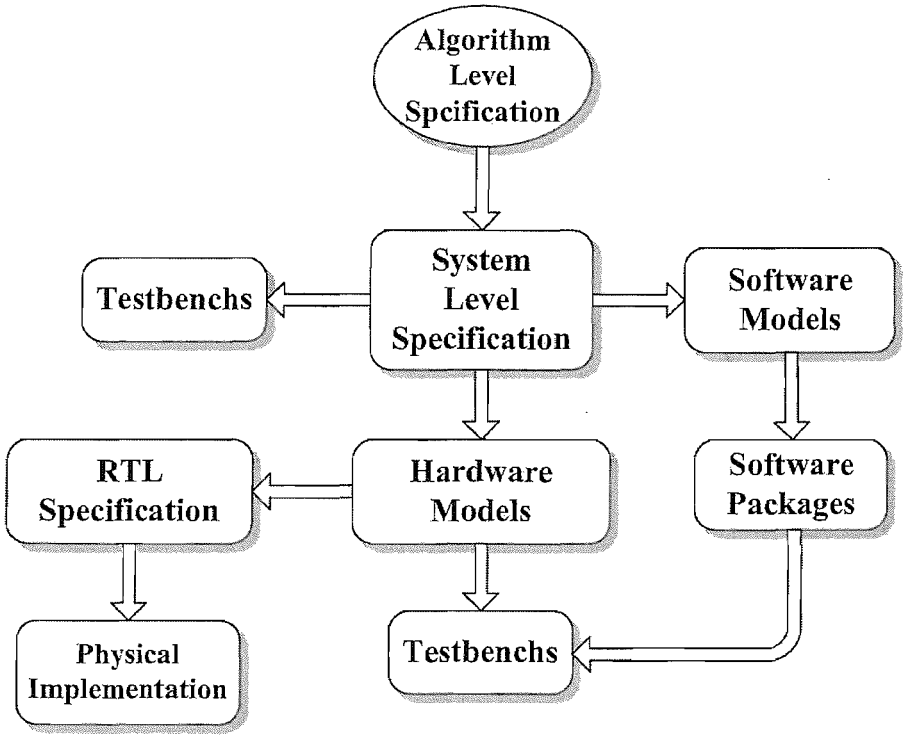


FIGURE 1.1: General system design flow chart.

Nowadays, in almost every engineering or science discipline, there is an enormous number of applications which are dependent on digital calculations. The algorithms of these calculations can be as simple as an algorithm to solve algebraic equations or more complicated such as a multidimensional stochastic partial differential equation solver. Furthermore, the implementation platform can also range from sophisticated supercomputers to low cost embedded calculation systems. Regardless of the calculation complexity and the target platform, most of these applications share some common characteristics which need to be taken into account during the design process.

There are some distinguishing differences between digital implementations of numerically intensive tasks and other kinds of digital systems that must be taken into account in HLS tools and methods specific to this domain. First, numerical algorithms typically are based on abstract representations of the mathematical operations in which most of the implementation issues are unseen. On the other hand, only a small number of mathematical operations can be implemented in digital systems directly, which means that there is a substantial gap between what is represented in the algorithm level of specification and what can be implemented in the digital system. In other words, most of the mathematical operations must be imitated by basic available operations in the digital world. These imitation operations should be based on mathematical techniques in conjunction with the hardware design limitations.

Calculation accuracy is another problem, where real variables ($x \in \mathbb{R}$) in algorithm level

specifications are dealt with an infinite-precision assumption; only a finite-precision calculation can be provided by digital implementations, which means that the input data to the system will be a set of numbers as an approximation of the real information instead of the exact values. Therefore in this sort of application, the existence of the computational error is inevitable. Nevertheless, the characteristics of the computational error are a matter of concern because of its destructive effects in computationally intensive applications. Minimising and modelling these effects are considered as major challenges in the hardware implementation of numerical tasks. These models need to be combined with the other hardware costs and parameters to create a consistent design and synthesis environment. Since these errors are data dependent and their behaviour is considered as a random phenomenon, their modelling in general, especially in presence of nonlinearity in the system, can be very problematic.

The third major difference is: numerical algorithms may need to be run over a massive amount of data, as in image processing for instance. Hardware implementation of these kinds of algorithms are datapath-dominant, which means that the datapath requires more resources in comparison with the controller. Furthermore, this fact suggests that the pipelining or parallelisation of the operations has a great impact on the system performance. Data communication synthesis, therefore, is likely to be a chief problem in datapath synthesis. HLS methods mostly use a datapath/controller model as a target structure. The synthesiser has to compile the high level description of the system to this target, where the datapath represents the computational parts of the algorithm and the controller is the hardware implementation of the controlling statements in the high level specification. This approach can give efficient solutions in some cases; but since it is based on compiler methodologies rather than hardware implementation methods, it cannot be expected that it presents the best solutions in this kind of applications because there are different restrictions in customisable hardware compared with classic computers.

Putting all the different issues together, a class of applications will be created which can be a centre of attention for HLS methods. These major differences recommend that general purpose HLS tools might not be able to give the best results in such cases. Having identified these differences, this study presents an investigation of these major problems to propose a HLS method focussing on hardware implementation of numerical algorithms to provide a more suitable framework for their synthesis.

1.2 Objectives

According to the discussion presented in section 1.1, regarding motivations of the work, the main objective of this thesis is to propose a synthesis method for computationally intensive hardware, which is application specific. To achieve this goal, several intermediate

objectives need to be met.

However, because computational error was investigated before digital computers, the majority of the works in hardware design concentrate on Linear Time Invariant (LTI) models of computation. Unfortunately these models are neither very exact in many practical cases, nor applicable to nonlinear functions and combinations. Accordingly, the first step is a comparative investigation of the models for computational error. A new approach to error modelling and calculation is the next step. This model uses a combination of current methods to provide a more comprehensive model which can be used with nonlinear systems as well as linear systems. Furthermore, more characteristics of the error can be extracted based on this method of error analysis.

The next intermediate objective is communication synthesis in datapath [33]. Bus-oriented datapath synthesis has been used previously. Although this structure is beneficial in terms of resources, it suffers from difficulties related to the speed and power consumption, see chapter 5 for more explanation. Modifications has been presented (in research community) to improve this structure, which need to be considered. Since heavy data traffic can be a major problem in computationally intensive hardware, even these modifications do not provide the required speed for the system. Accordingly a new structure for datapath synthesis is required.

1.3 Major Contributions and Thesis Overview

This thesis presents a novel method of computational error analysis , which is called Symbolic Noise Analysis (SNA), is a combination of the error range analysis and noise analysis methods, which provides more information from computational errors at different points in the system than other similar approaches. SNA considers dependency of the intermediate produced errors to avoid overestimation and also is capable of dealing with nonlinear operations. Furthermore, error Probability Density Function (PDF) does not need to be assumed with uniform distribution which is far from reality in many practical cases. Regarding datapath communication synthesis, a new bus structure, called Multiple-Width Multiple-Way Partitioned Bus (MW^2P-BUS), is introduced to provide a flexible and cost effective data communication medium for functional units in datapath level. This structure is generalised in the form of a predefined architecture to be used with the SNA method in a synthesis method for computational intensive systems. This synthesis tool produces RTL VHDL specification of the input algorithm based on the SNA method and MW^2P-BUS.

This thesis comprises nine chapters as follows:

Chapter 2 provides a brief review of datapath synthesis. First a general review of high level synthesis is presented, then bus-oriented high level synthesis is discussed and related

works are surveyed. Afterward, computationally intensive hardware related issues are investigated and major problems are discussed.

Chapter 3 provides a comprehensive overview of uncertainty in computing. Different methods and models are presented including interval-arithmetic, affine-arithmetic, Taylor-method, the noise model method and its nonlinear extension.

Chapter 4 describes the symbolic noise analysis method of error analysis. After motivating examples, and comparing different methods and their weak-points, SNA is presented in detail with implementation algorithm.

Chapter 5 introduces the MW^{2}P-BUS structure. Constructively, after motivation examples, single shared bus and partitioned bus (MWP-BUS) are presented first and then MW^{2}P-BUS is described as the evolution of the simpler structures.

Chapter 6 presents the target architecture for the synthesiser which is based on the MW^{2}P-BUS. This architecture is discussed in detail to show how a synthesised system can be implemented based on it. More design examples are provided based on this structure in appendix B.

Chapter 7 specifies the optimisation algorithm. This algorithm is based on a genetic search where genomes are designed to present the design including word-length and binding information. The corresponding algorithms and cost functions are explained in detail to provide a complete reference of the optimisation method.

Chapter 8 provides the case studies for evaluation of the methods. Using different designs it is shown that this method can offer improvements in design cost by trading accuracy with the other design costs.

Appendix A presents more details of the intermediate specification files which are used by synthesiser.

List of Publications

The work presented in this thesis has resulted in a number of original conference publications.

Presented Workshop and Conference Papers:

- *A Symbolic Noise Analysis Approach to Word-Length Optimization in DSP Hardware* Ahmadi, A.; Zwolinski, M.; International Symposium on Integrated Circuits (ISIC 2007), 26-28 September 2007.
- *MW^2P-Bus: A New Bus Structure for Datapath Synthesis* Ahmadi, A.; Zwolinski, M.; 3rd UK Embedded Forum, 2-3 April 2007.
- *Multiple-Width Bus Partitioning Approach to Datapath Synthesis* Ahmadi, A.; Zwolinski, M.; IEEE International Symposium on Circuits and Systems (ISCAS), 27-30 May 2007.
- *A New Structure for Datapath Synthesis* Ahmadi, A.; Zwolinski, M.; 12th International CSI Computer Conference, 20-22 February 2007.
- *Multiple-Width Bus Partitioning Approach to Datapath Synthesis* Presented at: Postgraduate Workshop on Embedded Systems,October 2006, Birmingham, UK.
- *Word-Length Oriented Multiobjective Optimization of Area and Power Consumption in DSP Algorithm Implementation* Ahmadi, A.; Zwolinski, M.; Microelectronics, 2006 25th International Conference on, 14-17 May 2006, Page(s):614 - 617.
- *Area word-length trade off in DSP algorithm implementation and optimization* Ahmadi, A.; Zwolinski, M.; DSPenabledRadio, 2005. The 2nd IEE/EURASIP Conference on, 19-20 September 2005, Page(s):8.

Chapter 2

Brief Review of Datapath Synthesis

2.1 Introduction

Owing to rapid development of microelectronics technology and design methods, it is possible to integrate millions of gates in a single chip. Design and verification of the complex systems which can be made up from such a complicated circuits, will not be possible without Computer Aided Design (CAD) tools and techniques. As a consequence of microelectronics progress and of demanding applications, design automation methods and tools need to be developed.

This chapter presents a comprehensive background knowledge related to the realisation of computationally intensive hardware, bus-oriented design and floating-point to fixed-point conversion to provide the preliminary information for the proposed method and implementation structure.

2.2 High Level Synthesis

Commercial developments in CMOS technology make it likely that 50 million gate ASICs will be feasible by 2010. ASIC design typically takes 18 to 24 months and costs from \$10M to \$20M [139]. An ASIC has a selling window of 6 to 8 months in the marketplace and, consequently, if the chip is delayed by much more than six months the customer is likely to move on to the next generation chip, which is possible to be cheaper, faster and have more features. Such competitive and highly time-restricted developments only can keep up with the growing size and complexity of designs if high-level design methodologies and accompanying tools are available that will allow complex digital systems to be realised by reasonably sized teams in a short time frame.

From the 1970s through the 1980s, research for CAD progressed as circuit complexity increased, with design abstraction advancing from transistor level circuit synthesis to logic level synthesis. With the assistance of logic level synthesis tools, designers could capture system specifications at a higher level of abstraction. The advantage of working at a higher level of abstraction is that it reduces the number of objects the designer has to manipulate, enabling the designer to design larger and more complex systems in shorter periods of time. High Level Synthesis (HLS) emerged in the late 80s, allowing the designer to capture system specification at an even higher abstraction level. The definition of high level synthesis was first given by McFarland et al. in [83]. They stated that:

> The synthesis task is to take a specification of the behaviour required of a system and a set of constraints and goals to be satisfied, and to find a structure that implements the behaviour while satisfying the goals and constraints. By behaviour we mean the way the system or its components interact with their environment, i.e., the mapping from inputs to outputs. Structure refers to the set of interconnected components that make up the system something like a netlist. Usually there are many different structures that can be used to realize a given behaviour. One of the tasks of synthesis is to find the structure that best meets the constraints, such as limitations on cycle time area or power, while minimizing other costs. For example, the goal might be to minimize area while achieving a certain minimum processing rate.

According to this definition, synthesis can take place at various levels of abstraction as designs can be described at various levels of detail. The primary data types at algorithmic level specification are numbers and/or bit strings and arrays. The input specification gives the required details for mapping from the sequences of inputs to the sequences of outputs in the form of mathematical functions. The specification should also constrain the internal structure of the system as little as possible.

In context of digital system design, High Level Synthesis is considered as the process of transforming a behavioural or algorithmic specification of a design into a synthesisable specification that consists of a datapath, a controller and memory elements. RTL specification is the generally accepted output of the HLS process [33, 39] which defines a structural model of a datapath, as an interconnection of resources, and a logic-level model of a control unit, that issues the control signals to the datapath according to the operations scheduling. The behavioural specification, on the other hand, refers to an abstract, purely algorithmic representation of the relationship between system inputs and outputs, with no explicit timing or structure information. This high level specification must consist three basic information: Circuit Behaviour (HDL Models), Building Blocks (library of resources) and Constraints (Timing, Silicon area, Power/energy etc.).

Any high level specification consists of several ordered operations. As it is depicted in Figure (2.1), the first task in high level synthesis is to capture the behavioural description in an intermediate representation that captures both control flow and data flow. This intermediate representation could be a Data Flow Graph (DFG) or a Control Data Flow Graph (CDFG) which are suitable data structures by which data dependencies between operations of the specified algorithm can be represented [33].

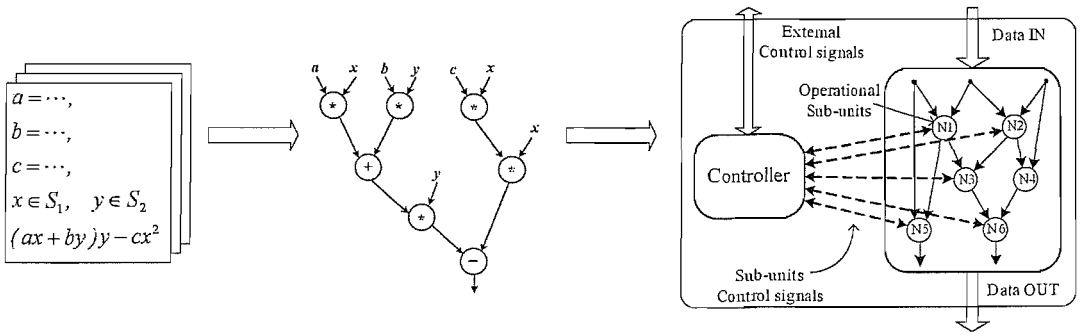


FIGURE 2.1: High level synthesis and structure of the target circuit.

High level synthesis is usually treated by dividing the problem into three overlapping sub-tasks: operation scheduling, resource allocation and resource binding [33, 39]. The scheduling assigns each operation to a time step in which it will be executed. Resource allocation determines the types (e.g., adder, multiplier, or register) and the number of these types of resources that should be included in the design and resource binding determines which resources that should be used to implement each specific operation. Depending on the Optimisation approach, these tasks could be achieved separately or in combination. The optimisation objectives or constraints most often are: timing (clock rate, throughput), silicon area (logic and interconnections), power dissipation and energy consumption.

As has been defined in the literature, for instance in [33], allocation consists of determining the number of resources that have been used in the design architecture. It also determines the clocking scheme, memory hierarchy and pipelining scheme. Allocation could be divided into sub-tasks like: arithmetic and logic units allocation, register and memories allocation and interconnection allocation. The latter is applicable when the system has the capability of a multi-bus structure.

Binding determines the mapping between the operations, variables and data (and control) transfers in the design and specific resources in the allocation. Hence, operations are mapped to specific functional units, variables to registers and data/control transfers to interconnection components. In other words, binding maps the operations to functional modules and variables to registers such that the cost of the necessary interconnect structure becomes minimal [33].

In some synthesising methods allocation and binding are combined and this combination is referred to as datapath synthesis, because regardless of the sequence of operation, allocation and binding build the datapath of the system. Algorithms for solving datapath synthesis problems can be found in [64, 33, 39].

Scheduling gives the information about the exact time of the each operation. This information is vital to the system performance in the next stages of design. In the design flow, scheduling could be done before or after datapath synthesis depending on synthesis constraints (time or resources or both) and design methodology. Having a great impact on the system performance parameters, scheduling algorithms have received significant attention in the literature, oriented to one or more cost optimisations. There are explanations for some basic algorithms in [33]. In this section, several commonly used scheduling algorithms are reviewed briefly: As Soon As Possible (ASAP), As Late As Possible (ALAP), list scheduling, and Force-Directed Scheduling (FDS).

As Soon As Possible Scheduling

If the timing given for an unscheduled DFG is to be minimum and resources are not restricted and assuming an unlimited number of functional units is available, the scheduling problem can be solved using the ASAP algorithm. An operation is scheduled in a control step as soon as all its predecessors are scheduled. These operations that do not have any predecessors are scheduled in the first control step. The ASAP scheduling algorithm can be described by the following steps.

1. Set control step $C = 1$;
2. For all operations v_i;
3. If v_i is not scheduled and (all its predecessors are scheduled or does not have any predecessors) schedule it at C else check v_i;
4. If there is unscheduled and unchecked v_i, go to step 2;
5. $C = C + 1$;
6. uncheck v_is;
7. If there are any operations not scheduled, go to step 2; otherwise stop;

As Late As Possible Scheduling

Under the same constraints as ASAP, the scheduling problem can also be solved using ALAP scheduling. It is similar to ASAP but schedules each operation into the latest possible control step. The algorithm can be described by the following steps:

1. If the timing constraint is given as T control steps, C is set to T;

2. For all operations v_i;
3. If v_i is not scheduled and (all its predecessors are scheduled or does not have any predecessors) schedule it at C else check v_i;
4. If there is unscheduled and unchecked v_i, go to step 2;
5. $C = C - 1$;
6. uncheck v_is;
7. If there are any operations not scheduled, go to step 2; otherwise stop;

Force-Directed Scheduling

Force Directed Scheduling (FDS) is a time constrained scheduling algorithm published by Paulin and Knight [102]. The principle of FDS is to reduce the operation concurrency in a control step and distribute the operations evenly to each control step in order to minimise the number of functional units required. The algorithm starts by calculating the time frames of each operation. The time frame is also commonly called the mobility range, which is the time interval in which an operation can be scheduled regarding its preceding and succeeding operations and scheduling time constraints. The mobility range of an operation can be obtained by scheduling the DFG using ASAP and ALAP. If an operation is scheduled at control step c_s by ASAP and is scheduled at control step c_l by ALAP, the mobility range is the difference between c_l and c_s. This method assigns probability values for each operation to be scheduled in a control step. If a control step is in the mobility range of an operation, the probability that the operation can be scheduled in this control step is the reciprocal of the operation mobility range. The probability that the operation can be scheduled outside the mobility range time frame is 0.

When the scheduling probabilities over control steps are calculated for all operations in the DFG, these information are used to produce a distribution graph. The distribution graph shows the sum of the probabilities of each type of operation in each control. For the distribution graph of each operation type, the distribution in a control step t can be represented as:

$$d(t) = \sum_{\text{all DFG nodes}} P(o,t), \tag{2.1}$$

where $P(o,t)$ is the probability contribution of operation o to control step t. The last step is to calculate the force for each operation in its time frame. Assume an operation o with a time frame from control step c_s to control step c_l. The parameter "force" for o in control step t, where $c_s \leq t \leq c_l$, is calculated as:

$$Force(t) = d(t) - \sum_{i=c_s}^{c_l} \frac{d(i)}{c_s - c_l + 1}, \tag{2.2}$$

The forces of o predecessors and successors must also be calculated by using Equation (2.1) whenever their time frames are affected by scheduling o to control step t. The forces from predecessors and successors are called indirect forces. The total force for o to be scheduled in control step t is the sum of the self force and indirect forces. After all the forces of o in its time frame are calculated, o is scheduled to the control step that has the smallest force. The algorithm can be summarised as follows:

1. Select operation o for evaluation;
2. Evaluate the time frame for each operation;
3. Update the distribution graph, using Equation (2.1);
4. Calculate self forces for every control step in the operation's time frame, using Equation (2.2)
5. Add the predecessor and successor forces to self forces;
6. Schedule o to the control step that has the lowest force. The time frame of o is set to the selected control step;
7. If there is any operation not scheduled, go to step 1; otherwise stop.

List Scheduling

List scheduling is a resource constrained scheduling algorithm. In many applications, implementation costs are the major concerns so the numbers of functional units are given as the constraints. Under the resource constraints, list scheduling tries to find a schedule that requires minimum latency. Basically, list scheduling is a variant of ASAP scheduling, which uses a greedy approach to schedule as many operations as possible into a control step, subject to the constraint of the availability of functional units [33]. The computational complexity for list scheduling is $O(n)$. The general list scheduling algorithm can be described as follows:

1. Set control step $C = 1$;
2. Get a list of operations that are ready to be scheduled;
3. Apply priority function to compute the priorities of the ready operations;
4. Schedule the first n operations and schedule them to C, where n is the available number of functional units at C;
5. $C = C + 1$;
6. If there are any operations not scheduled, go to step 2; otherwise stop.

2.3 Restricted HLS

In general, a high level synthesis method accepts an abstract specification of a system which is limited to a purely algorithmic representation of the relationship between system inputs and outputs. Nevertheless, this specification provides no explicit timing or structural information, in many cases there are some restrictions in system implementation which need to be considered prior to or during the synthesis procedure. These restrictions can be in the form of resource restriction, cost restriction and/or structure restriction.

Resource restriction is applied to the cases which the synthesiser has access to a limited number of certain resources. When the target implementation platform provides only a limited number of resources, in the case of programmable devices such as FPGA, synthesiser must be restricted to not violate this limit.

Cost restriction refers to the limitation in computation costs such as: power consumption, area and latency. These costs can be related and even combined with the resources restriction but they normally require a post synthesis evaluation whereas the resources restriction can be applied prior to or during the synthesis. Some costs, power consumption for instance, are data dependent and to be evaluated more precisely, a set of stimulus is required.

Structure restrictions are considered as restrictions which are applied to the implementation structures. When the synthesiser is configured to map all the designs to a predefined structure or architecture it is considered as a structure restricted synthesis. There are many different ways to design and implement datapath and controllers and restricting the synthesiser to a specific structure reduces the synthesis freedom, in returns it reduces the feasible space of the design search.

It is a basic observation about computing hardware that generality and efficiency are, to some extent, inversely related to each other in that the more general purpose a hardware is, and thus the greater the number of different tasks it can perform, the less efficient, in terms of computational cost, it will be in performing any of those specific tasks. Design decisions are therefore almost always compromises between generality and efficiency. As a basic strategy, designers at the first step try to identify the key features of applications for which the competitive efficiency is a necessity and then extend the application range as much as is practical without excessively damaging the performance of the main target application.

Concern about the computational efficiency is more significant when the computation cost is high in the target application. Two kinds of applications are recognisable in this regard: the first type are applications like embedded computational systems and SoCs which require a low cost, normally low power consumption and small area, design for

application specific circuits to be integrated into a system with some other applications. The second group contains applications such as DSPs or high accuracy scientific computations which need massive and high speed computation with accuracy restrictions.

An embedded system is a special-purpose system which is completely encapsulated by or dedicated to a device. Unlike general-purpose hardware an embedded circuit performs one or a few predefined tasks, usually with very specific requirements. Since the circuit is dedicated to specific tasks and needs to be designed for a specific application, the designer must optimise it to minimise costs. Embedded systems are often mass-produced, benefiting from economies of scale. Accordingly, design methods are required to be highly automated and adaptable with different design constraints.

Implementation of computationally-intense hardware, conversely, is not a new issue. Solutions are experienced to counter the problems for which general purpose hardware cannot achieve the necessary performance. Special purpose processors, attached processors, and coprocessors are well-known answers for this problem. The major downside of the fixed auxiliary hardware is their lack of flexibility which makes them weak in coping with changes in applications or development of new mathematical or algorithmic techniques for problems solving. As a solution, ASICs offer the opportunity of specialising a design for an application. Developing ASICs, however, is often expensive, because of the complexity involved in the design and fabrication processes. Reconfigurable hardware, on the other hand, such as Field Programmable Gate Arrays (FPGAs), is an attractive alternative to ASICs; however, their efficiency in general can be lower than ASICs [69].

The biggest challenge to using reconfigurable auxiliary computing systems is to develop an optimised implementation rapidly and to make it easy to verify and use. Circuit synthesis from a high level specification consists of many different steps and sub-tasks. Mapping the computationally intensive part of an algorithm to a reconfigurable device requires tools to translate this part of the algorithm into a synthesisable specification efficiently and automatically.

HLS in the general case consists of design optimisation. Several objectives can be included in optimisation, as it was mentioned, and also there are several parameters by which these objectives are controlled. The combination of these objectives with their controlling parameters forms a multidimensional space of possible solutions for the optimisation problem; this space is called the 'feasible space' in optimisation terminology. The basic duty of the optimisation task is searching this feasible space to find an optimal point. Regarding design complexity, optimisation objectives and controlling parameters, feasible space can be very large and it will be impractical to find an optimal solution in it in a reasonable time.

Apart from clever optimisation algorithms, the essential way out of this problem is limiting the feasible space, by restricting the optimisation tool to search only predefined regions of the feasible space. The major drawback of this approach is the possibility

of missing some optimal solutions which are located in the restricted area. This payoff has been accepted in the HLS community providing that an optimal point can be found in a reasonable time. One example of this restriction is to divide the target design into datapath and controller parts. To find an effective restriction which still contains optimal points, general characteristics of the target designs should be considered and categorised.

There are some characteristics which make intensively computational hardware implementation different from that of general purpose hardware. To have a more efficient design environment, computational algorithms can be inspected to classify their general features and accordingly a structure be proposed for their implementation. This predefined structure should be a flexible soft-architecture which provides the target structure for the datapath and controller. In this study, a hierarchically predefined target structure is presented which provides multidimensional freedom to the synthesiser-optimiser to find an optimal solution for design. This architecture is a bus-oriented design approach which is discussed in chapter 4. The following subsection gives a brief review of related work in bus-oriented datapath synthesis.

2.3.1 Bus-Oriented Datapath Synthesis

Datapath connectivity synthesis consists of defining the data connection among resources, steering logic circuits, memory resources, input/output ports and the control unit(s). Since a complete binding is required for all the system, connectivity synthesis refines the binding information by providing the detailed interconnection among all the system sub-blocks. Regarding the data communication structure in the system, different approaches have been introduced for datapath synthesis. The major types of connectivity models that have been reported are: the point-to-point connectivity model and the bus-oriented model [33]. The first approach uses multiplexers to steer data communications between hardware modules and manage the traffic, whereas the latter relies on a communication channel which is implemented by shared buses.

Multiplexer-oriented interconnection architecture is also called random topology or point-to-point interconnection architecture in which the interconnections of the functional units in the datapath are implemented by point-to-point connections. Each wire connects an input and an output of the functional units. In the case that more than one wire must be connected to the same port, a multiplexer is introduced to steer the data flow between them. Multiplexers are used by controller to select one of the multiple inputs to be transmitted to the output in a multiplexed time scheme. A point-to-point structure is the most popular interconnection topology in high-level synthesis because it simplifies the allocation algorithms [45]. Multiplexed datapaths have been compared with the bus-oriented architecture in [88], which results in large routing area requirements because of the large number of nets [91].

In a bus-oriented interconnection architecture, on the other hand, the variables are transferred via busses in which the interconnections can be time-shared between different variables and this can reduce the routing area requirements considerably. This is especially true if the datapath is floorplanned in a way that the data buses can be routed horizontally on top of the functional units [20]. However more complex control units are needed to handle the time-sharing of the bus [43]. Furthermore, since buses are supposed to connect several sub-blocks together over long distances, their average wire-length is expected to be longer than other signal wires (except the clock tree). Consequently, there are several considerations regarding the electrical characteristics of the shared buses such as bus driver requirements. Several works are presented on high level synthesis techniques for bus oriented datapath design and optimisation, which are reviewed in this section.

A bus which is partitioned into two or more segments is called a segmented bus. Each segment acts as a normal bus between modules that are connected to it and operates in parallel with other segments. Segments can be dynamically connected to each other, in order to establish connections between modules located in neighbouring segments. Due to the segmentation of this resource, parallel transactions can take place, thus increasing the performance. When parts of the segmented bus are not involved in transactions, or they only transfer intra-segment data, they become isolated from the rest of the bus.

A successful partitioned bus-oriented design is achievable by a precise organisation of the sub-blocks and bus segments. As a result, a higher degree of parallelism of data transfers and consequently the overall system performance are attainable. The success of a segmented bus implementation depends on the balance between parallelism and complexity of the system, therefore, the key parameters in this regard are the profile of the accesses between different functional units, the organisation of the segments, and the assignment of the units to the segments. The main idea is to organise the component devices and the segments in such a way that the number of parallel data transfers is maximised. At the other extreme, some frequently communicating units may reserve a long portion of the bus, thus causing the bus to act more like a non-segmented one.

In [43] Frank et al. have integrated placement into high-level synthesis by using an intermediate structure, called the communication graph. The nodes of the communication graph are labelled with tight estimates on the width and height of the hardware needed for each functional unit. They derive these estimates by scheduling the data transfers of the data flow on a restricted architecture prior to placement. After placement the optimal communication schedule is computed together with the allocation of register and interconnects. The tight estimates on physical design data allows a considerable reduction in design time, because the designs can be evaluated prior to interconnect and register allocation and assignment. This difficult and thus rather time consuming optimisation procedure only needs to be performed once for the best design. In [44] Frank et al. also address the problem of scheduling communications in a bus architecture

under memory constraints. They present a network flow formulation for the problem and obtain an exact algorithm to schedule the communications, such that the constraint on the number of registers in each functional unit is satisfied. An increasing number of architectures use multiple memories in addition to, or instead of, one central RAM. This technique relies on finding an exact solution to the problem.

De Micheli, in [33], suggests a sequencing graph model for a shared bus. From this viewpoint, buses act as transfer resources that feed data to functional resources, thus the operation of writing to a specific bus can be modelled explicitly as a vertex in the sequencing graph model. In this case, the compatible or conflicting data transfers are represented by compatibility or conflict graphs, similar to FUs in binding and scheduling. An alternative method is also offered in the same reference in which buses are not explicitly described in terms of sequencing graph where the optimum buses usage is derived by exploiting the timing of the data transfers. Consequently two problems are introduced: first, to find the minimum number of buses to accommodate all the data transfers and second, to find the maximum number of data transfers that can be done through a given number of buses. These problems are known to be analogous to the multi-port binding problem and can be modelled by Integer Linear Programming (ILP) constraints.

Seceleanu et al. in a series of works explored segmented bus synthesis algorithms [116, 113]. They consider the problem of allocating hardware units to segments of the bus in such a way that the traffic across segment borders is minimised and the potential for parallel transfers is maximised. In their work, the goal is to obtain an optimal distribution of the components on the segments, so that the performance is maximally increased. The objective here is to keep the inter and intra data transfers of each segment as low as possible. A fixed number of segments is considered as an initial setting to avoid trivial solutions. The inter-component traffic is defined by a unit-to-unit matrix of transfer frequencies.

The concept of segmenting the bus was proposed by Ewering, [40], in the context of single-chip devices. He proposed partitioned a bus as a target architecture for high level synthesis. For automatic synthesis of the architecture he adopts a sequential optimisation technique which performs scheduling, binding, segment ordering and communication scheduling/binding sequentially. A switch permits or inhibits of data flow from one segment to another. The direction of data flow is not under the control of the switch.

The basic structure of partitioned-bus architecture, which is illustrated in Figure (2.2) [40], consists of a specified number of parallel busses. In order to avoid performance degradation, each bus is divided into an equal number of segments. Two adjacent bus segments are connected with switches. The switches are placed symmetrically. A switch permits or inhibits data flows between segments, thus parallel data communications and processing are possible. If data moves a long distance, however, many divided busses are

possessed exclusively for data transfer which causes bus conflicts. Therefore equalizing the data transfer load of each divided bus is needed during the high-level synthesis process, both in the temporal and spatial domains.

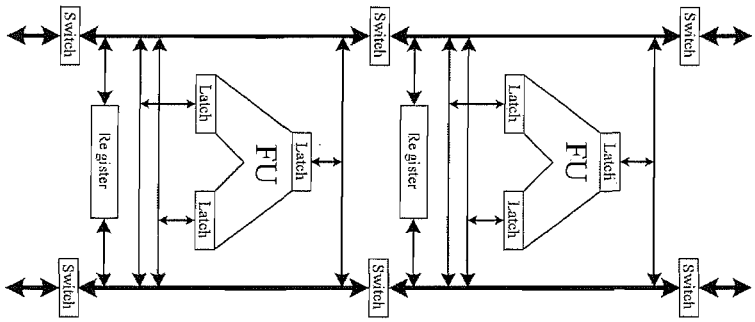


FIGURE 2.2: Basic structure of stacked partitioned bus.

In [91], a bus driven scheduling and register minimisation technique is proposed. After binding operations to segments, they solve the communication scheduling/binding problem by list scheduling. The main objective of their research is to reduce the impact of wiring. Their synthesis method, however, requires bus features to be considered as soon as possible in the design trajectory. A new binding model and two new algorithms for scheduling and register allocation for synthesis of bus-partitioned structures are proposed. The algorithms are actually one of the first attempts to incorporate bus-binding into the synthesis procedures. The experimental results demonstrate the reasoning of such an approach and its efficiency.

Jeon and Choi in [57] try to reduce the probability of bus conflicts by considering criticality, distribution and density of communication edges in the course of scheduling, binding and segment ordering. In [127] a data transfer model for formulating the classic HLS sub-problems is also proposed by Tarafdar et al. The model optimises the storage architecture of a design concurrently with the execution time.

In [135], Wang et al. discus a segmented bus from the point of view of memory usage and placement. Their work presents a methodology to reduce the bus power consumption in memory dominated systems. The proposed partitioning method, systematically, combines an activity driven placement of the memories and a bus segmentation approach to localise the wire switching activities and minimise the associated wire capacitive load of the memory bus. A reduction of 65% in bus power is reported.

In [61] a new structure extends the linear architecture of the partitioned-bus by stacking multiple layers for handling large datapath-intensive applications. In the high-level synthesis flow, we search the design space to determine the number of layers and the suitable length of each layer with a given aspect ratio. This work also performs operation clustering, segment binding, segment ordering, layer ordering, layer mirroring, and post processing to equalize the data transfer load of each bus.

In [58] Jone et. al. introduce the design theory and implementation issues of segmented bus systems based on a graph model and the Gomory-Hu cut-equivalent tree algorithm by which a bus can be partitioned into several (bus) segments separated by pass transistors. Devices with higher data communication are placed in adjacent bus segments, to reduce global data communication in the small portions of the bus segments. It is shown that the power consumption can be reduced significantly. The concept of tree clustering is also proposed to merge bus segments for further power reduction. The design flow, which includes bus tree construction in the register-transfer level and bus segmentation cell placement and routing in the physical level, is presented.

Regarding low power bus design, Chen et al. in [19] propose a bus-segmentation method to efficiently reduce the switched capacitance on the bus. The power consumed by the bus can, therefore, be substantially reduced. Highly communicating devices are located in adjacent bus segments, thus, most data communication can be achieved by switching a small portion of the bus segments. As a result, power consumption and critical path delay are both reduced. Experimental results obtained by simulating a delay model and a power model demonstrate that the proposed segmented bus system reduces bus power by about 60%-70% and improves critical bus delay by about 10%-30%.

2.4 Computational Hardware

General purpose hardware is designed with the primary goal of providing acceptable performance for a wide variety of tasks rather than high performance for specific tasks. The performance of such a machine ultimately depends on how well its capabilities are matched with the computational requirements of the applications. If an application requires more computational resources than general purpose hardware can provide, designers are often driven to application specific architectures in which fundamental machine capabilities are adapted to a particular class of algorithms.

In many engineering and scientific fields there are computationally intensive applications which consist dominantly of computational tasks. From an algorithmic point of view, these applications are normally repetitive and (preferred to be) calculated with a certain accuracy range. From the hardware design viewpoint, on the other hand, these applications should be implemented using basic arithmetic operations which are either shared over execution time and reused repetitively or included in the design to execute the required tasks in parallel (if possible). Consequently, the target hardware can be substantially improved in its performance by utilising specifically designed and optimised functional units, which can execute these fundamental operations with higher performance.

This section provides a basic view of the problem in different aspects to clarify the requirements and specialities of computationally intensive hardware.

2.4.1 Arithmetic Characteristic

Arithmetic units are the building blocks of any hardware implementation of numerical algorithms. Their performance and cost is, therefore, directly reflected in the final implementation cost of the algorithm. From a design style viewpoint, a large number of alternatives are available to realise arithmetic operations which offer different characteristics with different implementation costs. The way numbers are represented, for instance, has a great impact on the computation accuracy and the implementation cost. In addition to the number representation system, there are other characteristics which need to be decided for arithmetic operations in the target hardware, such as operand word-length, rounding method, overflow prevention and so on. Choosing the most suitable set of these characteristics for the arithmetic units has a vital role in the computational performance of the final hardware, they need, therefore, to be taken into account before or during the design and optimisation procedure.

To treat the issue more formally, we would like to encapsulate these characteristics as Arithmetical Characteristics (AC), which here means a set of arithmetic or computational characteristics which attribute the arithmetical structure of a functional unit. These characteristics include but not limited to:

- Number Representation System: fixed-point, floating-point, $\cdots$;
- Data Bit-Width;
- Rounding Method: Truncation, Rounding, $\cdots$;
- Overflow Prevention Method: Non-prevented, Saturated Arithmetic, $\cdots$;

Clearly choosing each set of characteristics is a tradeoff of costs and benefits. The designer also should consider the compatibility of these characteristics in a design and also the conversion cost of exchanging data between arithmetic units which have different characteristics. Normally, relying on experiences, design restrictions/requirements and available Intellectual Property (IP), the characteristics of the arithmetic units are chosen at the very beginning of the hardware design procedure and these characteristics, as a generally accepted view, are applied homogenously to all functional units in the hardware; however, it is a matter for discussion whether this is the best way of doing this in reality.

The following subsections provide a brief review of some basic characteristics of arithmetic operations and their impact on the efficiency and accuracy of the computation. In addition to primary arithmetic operations, there are mathematical functions which need to be used in some computational algorithms. Elementary functions such as logarithmic or trigonometric functions are examples which are widely used in many calculations.

Since these kinds of functions are more complicated in terms of their hardware implementation and might be used repeatedly, they need to be evaluated with more precise methods to consider different efficiency issues. Accordingly, a brief review of function evaluation methods is also provided.

2.4.1.1 Number Representation Systems

Computer arithmetic is concerned with numbers and the basic arithmetic operations. Accordingly, the number representation system is considered as a vital issue in computer arithmetic [100]. The choice of how real world values are going to be represented in a digital format has a great impact on the complexity and performance of the hardware that implements the computation algorithms. Design area, speed, power consumption, accuracy, compatibility and data interfaces are among the aspects to take into account when a representation system chosen. This section provides a brief overview of the most popular arithmetic systems.

In conventional digital computers, numbers are represented in the form of a finite and ordered set of binary values such that

$$X \longleftrightarrow (x_{n-1}, x_{n-2}, \cdots, x_1, x_0), \tag{2.3}$$

where X is a n-digit number and in a radix-r representation $x_i \in [0 \;\; r-1]$. Depending on how the original value is related to this ordered sequence of numbers and how can it be reconstructed from its representation, different arithmetic systems have been introduced. Arithmetic systems can be categorised into several subsets regarding the type of the numbers they represent namely: natural, integer and real numbers [35].

Natural numbers

Natural numbers are known as nonnegative integers, which are represented in the form of weighted systems or a Residue Number Systems (RNS). In a weighted representation system any natural number X can be represented in the form

$$X = \sum_{k=0}^{N} x_k \times r^k, \tag{2.4}$$

where the coefficients x_i are natural numbers $0 \leq x_i < r$. It can be proved that this representation is unique for X [35].

RNS representation is defined by a set of s moduli $\{m_i\}$. If the m_i are pairwise prime for all i, the RNS is called nonredundant. The RNS representation of a given natural number X is the vector $R(X)$, whose components x_i are the respective residues modulo

m_i, that is successive remainders of the integer division $\frac{X}{m_i}$ as in Equation (2.5).

$$x_i = (X \mod m_i), \tag{2.5}$$

The least common multiple (LCM) of $\{m_i\}$ is the range of the RNS, generally denoted by M. The greatest natural number that can be represented in the RNS, which is defined by $\{m_i\}$, is:

$$M - 1 = (m_1 - 1, m_2 - 1, \cdots, m_s - 1), \tag{2.6}$$

If the m_i are pairwise prime for all i, then M is:

$$M = \prod_{1}^{s} m_i, \tag{2.7}$$

Example of this system can be found in [65]

Integer Numbers

Since the only difference between natural numbers and integers is their sign, the most natural way of representing integers is the sign-magnitude representation system; nonetheless, it is not the most convenient for executing arithmetic operations [35]. Other common methods which should be mentioned here are Excess-E, r's complement and Booth encoding.

With sign magnitude, an integer can be represented in the form of $+X$ or $-X$, where X is represented as a natural number and the sign can be represented using an additional sign bit. This representation is very close to a human representation of integers.

Another way of representing negative numbers is associated with natural numbers by a defined relation such as $R(X)$, where $R(\cdot)$ is a one-to-one transform such as:

$$R(X) = \begin{cases} X & : \quad X \geq 0 \\ X + E & : \quad \text{otherwise} \end{cases} \tag{2.8}$$

where E is a predefined constant. In this system the range of the represented numbers is $-E \leq X \leq r^N - E$.

r's complement on the other hand, can be thought of as another form of $R(X)$ transform of the integer numbers, where $R(X)$ in this case is associating natural numbers to X as shown in Equation (2.9).

$$R(X) = \left(X \mod r^N\right), \tag{2.9}$$

in other words, X is represented as

$$R(X) = \begin{cases} \sum_0^{N-1} x_i \times r^i & : \quad x_{N-1} \geq \frac{r}{2} \\ \sum_0^{N-1} x_i \times r^i - r^N & : \quad x_{N-1} < \frac{r}{2} \end{cases} \tag{2.10}$$

here r is assumed to be even.

Booth's encoding is a signed digit method. Assuming a binary representation ($r = 2$), an integer X in this method is represented by a set of digits $X \leftrightarrow (y_{N-1}, \cdots, y_1, y_0)$ which are defined based on a 2's complement representation of X, $(x_{N-1} \cdots x_1 x_0)$ as in Equation (2.11)

$$\begin{aligned} X &= \sum_{0}^{N-1} y_i \times 2^i, \\ y_0 &= -x_0, \\ y_i &= -x_i + x_{i-1}, \qquad i = 1, 2, \cdots, N-1. \end{aligned} \tag{2.11}$$

clearly $y_i \in \{-1, 0, +1\}$ holds. The Booth representation can be extended to a Booth-r representation, more details are available in [35].

Real Numbers

Representation of real numbers, theoretically, requires an (unrealistic) infinite precision numbering system. To approximate real numbers there are two common numeration systems, fixed-point and floating-point. The fixed-point system is an extension of the integer representation system including a decimal point representation; it provides a relatively limited range of numbers with a constant precision. Conversely, the floating-point system allows the representation of a large range of numbers with relatively higher precision.

In a fixed-point numeration system a real number X is represented as in Equation (2.4) [65].

$$X = \sum_{k=-N_1}^{N_2} x_k \times 2^k, \tag{2.12}$$

Let assume that X_{min} and X_{max} are the minimum and maximum numbers which can be represented by N bits respectively. In a fixed-point representation any real number $\hat{X}$ belonging to the interval $r^{-p} \times X_{min} \leq X \leq r^{-p} \times X_{max}$ can be represented in the form of Equation (2.4) with an error equal to the absolute value of the difference between $\hat{X}$ and X. From another viewpoint, the distance between two succeeding numbers is one unit of the least significant position (ULP), see section 3.2. Accordingly the maximum quantisation error in this method is equal to $\frac{ULP}{2} = \frac{r^{-p}}{2}$.

Floating-point numbers provide a dynamic range of representation for real numbers without the need to scale the operands during operations. In a floating-point numeration system the representation consists of two numbers: a fixed-point number (the mantissa) and an integer (exponent). Formally, in a floating-point system of radix-r, mantissa length of p (precision) and exponent range of $[E_{min}, E_{max}]$, a number X is represented by a mantissa $M_x = x_0 x_1 x_2 \cdots x_{p-1}$ which is an p-digit number in radix r, satisfying

$0 \leq M_x < r$, a sign $S_x \in \{0, 1\}$, and an exponent E_x, $E_{min} \leq E_x \leq E_{max}$ such as shown in Equation (2.13) [2].

$$X = (-1)^{S_x} \times M_x \times r^{E_x}, \tag{2.13}$$

According to this representation, the ordered sequence of Equation (2.3) should be divided into two parts, one represents the mantissa (M_x including the sign bit) and the second part represents the exponent (E_x). For accuracy reasons, it is frequently required that the floating-point representation be normalised in a way that the mantissa is greater than or equal to '1', [65].

As explained in [2], the IEEE 754 standard defines binary representations for 32-bit single-precision and 64-bit double-precision numbers as well as extended single-precision and extended double-precision numbers. Similar to the definition of Equation (2.13), a single-precision floating-point number presentation consists of three parts: the sign bit, the exponent, and the mantissa. The division of the bit pattern into three parts is shown in Figure (2.3), where the sign bit is '0' if the number is positive and '1' if the number

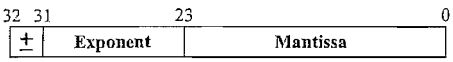


FIGURE 2.3: Floating representation in ANSI/IEEE single precision.

is negative. The exponent is an 8-bit number that ranges in value from -126 to 127. The exponent is not a typical 2's complement representation because this would make comparisons more difficult. Instead, the value is represented by an Excess-E system ($E = 127$), which makes it possible to represent negative numbers. The mantissa is the normalized binary representation of the number to be multiplied by 2 raised to the power defined by the exponent.

It is indicated in [136] that floating point arithmetic leads to a higher rate of SNR (ratio of the signal power to the noise power) than fixed point if the floating point mantissa is equal in length to the fixed-point word. It is also claimed that for first and second order filters the output SNR as a result of roundoff is dependent on the gain of filters and for fixed-point decreases faster than for floating point.

Boite et al. on the other hand, used fixed-point and floating-point realisations of digital filters in [12] to compare the output Signal-to-Noise Ratio (SNR) due to the internal noise. According to their results, for the same word-lengths, the peak value of the SNR in fixed-point is approximately equal to that observed in floating-point, however the later remains constant for a very large range of input signal levels.

Floating-point representation can provide more accuracy in computation, however, due to implementation constraints such as silicon area, latency and power consumption, many of the floating-point algorithms need to be converted to their fixed-point version for practical reasons. This requires data range evaluation of the intermediate variables in

the algorithm. Computing the exact ranges in a sequence of operations is a complex task especially when there are dependencies between variables and operations. The obvious method is an exhaustive simulation for all possible input values and tracing data range propagation through the computation tree of the algorithm. This approach is unfeasible in practice due to the size of the input space vectors, thus analytical methods have received more attention in the community so far. Related studies regarding Floating-point to Fixed-point Conversion (FFC) are briefly reviewed in section 2.4.3.

2.4.1.2 Data Bit-Width

The specification of a computation algorithm may not necessarily express the ranges of the data and intermediate results. However, such information is essential to determine the bit-widths of the hardware resources required for datapath synthesis because the computation cost including silicon area, latency and power consumption of a datapath depend directly on the bit-widths of the functional units, communication paths and storage resources. The input ranges or bit-widths may be part of the specification and the output is restricted to a minimum required accuracy, whereas the bit-widths requirements of the intermediate variables usually need to be selected by the hardware designer.

It is proved in [28] that there is a tight relationship between the hardware implementation cost of the functional units and their bit-width. It is shown in section 7.3 that there is an approximately linear dependency between implementation costs of basic functional units (register, adder, subtractor and sequential multiplier) and their bit-width. Computational accuracy, on the other hand, is directly defined by the bit-width of the operations. This contradiction between accuracy and implementation cost is the core idea of bit-width (word-length) optimisation.

The optimum selection of the bit-width (or word length) may imply range reduction due to implementation constraints which may result in loss of computation accuracy. However tradeoff is tolerable in many cases, the same accuracy might be achievable in most cases. This tradeoff and related works in this regard are reviewed in section 2.4.4.

2.4.1.3 Overflow

In a digital implementation of the arithmetic operations there are usually limits for minimum and maximum representable values which denote that all the valid inputs and outputs are limited to a fixed range between these limits. If the result of an operation lies outside of this range, it cannot be represented properly and an error will be produced which depends on the representation system and the operation type. However in some number representation systems, overflow does not intrinsically happen for some of

the operations, for example fixed-point multiplication. The result can, however, be a catastrophic loss in accuracy when it does occur.

Fixed-point addition/subtraction of two numbers which has the capability of overflow in the result, when the answer to an addition or subtraction problem exceeds the magnitude that can be represented with the prearranged number of bits. In the basic implementation of fixed-point addition/subtraction, overflow results in wrap-around phenomenon which represents the actual result when the MSB either is omitted or shifted to the sign-bit. In wrap-around, the functional unit attempts to represent positive numbers just outside of the limit range as large negative numbers, and vice versa. Since overflow discards the MSB part of the result, the consequential error has a devastating impact on computation accuracy, so the preferred method is to avoid and prevent arithmetic operations from overflowing [118].

Hardware designs are therefore usually either scaled appropriately to avoid overflow for all but the most extreme input vectors, or designed to use saturation arithmetic.In saturation arithmetic, if an operation results in a value greater than the maximum limit it will be clamped to the maximum value, while if it is below the minimum it will be clamped to the minimum. The value becomes "saturated" once it reaches the extreme values; further additions to a maximum or subtractions from a minimum will not change the result [27]. Formally, if the valid range of values is $[X_{min}, X_{max}]$, the operation "$\circ$" produces the following values:

$$O(a,b) = \begin{cases} a \circ b & X_{min} < a \circ b < X_{max} \\ X_{min} & a \circ b \leq X_{min} \\ X_{max} & a \circ b \geq X_{max} \end{cases}, \tag{2.14}$$

where $O(\cdot)$ is the saturated form of "$\circ$". Saturation arithmetic is advantageous in terms of overflow control in a way that may not significantly affect the computation accuracy. However, these advantages are provided at the cost of a somewhat larger and slower circuits compared to an implementation adopting standard fixed-point arithmetic for the same binary point locations and word lengths. Accordingly, care must be taken to select the appropriate points in the algorithm computation tree to saturate the signals [27].

2.4.1.4 Function Evaluation

The elementary functions (e^x, $\sqrt[n]{x}$, $ln(x)$, $cos(x)$, $cos^{-1}(x)$, $\cdots$) are the most commonly used mathematical functions which are differentiable and cannot be represented by finite degree polynomials without losing accuracy. Computing these functions with the required accuracy and a reasonable implementation cost is an important issue in hardware design. From a computer arithmetic viewpoint, this problem ranges from theoretical discussions in mathematics and algorithm design level to implementation difficulties

in software and/or hardware. Cost effective implementation of the elementary functions is often the performance bottleneck of many intensive computational applications [92].

Software implementations can be too slow for numerically intensive or real-time applications. For instance, more than 50% of the total run time may be devoted to function evaluation operations in some aircraft and missile simulation tasks [97]. The performance of such applications can therefore be improved considerably by utilising suitable hardware units for function evaluation modules. Nowadays, advances in hardware design automation enable development and exploration of low cost and high speed function evaluation hardware units customised for particular applications, within a practical design and implementation time [73].

Theoretically, the elementary functions are not much harder to compute than quotients in division algorithm. It is proved by [1] that elementary functions are equivalent to division with respect to Boolean circuit depth. This means that, roughly speaking, a circuits can be designed to compute N-digit elementary functions in a time proportional to $\log(N)$ [7]. In a more accurate way, as explained by [15], if it is assumed that $f(x)$ is an elementary function and $M(N)$ is the number of single-precision operations required to multiply N-bit integers; it is shown that $f(x)$ can be evaluated, with relative error $O\left(2^{-N}\right)$, by $O\left(M(N)\log(N)\right)$ operations as $N \rightarrow \infty$, for any floating-point number x (with an N-bit fraction) in a suitable finite interval [14].

In general, evaluation of an elementary function $f(x)$ consists of three stages. The first stage is range reduction (see [92]), in which the variation range of the argument x is reduced to an appropriate small interval $[a, b]$, resulting in a new argument $\bar{x}$ to limit $f(\bar{x})$ to a desired range of deviation. The second stage evaluates the function $f(\bar{x})$ using mathematical approximation methods. This approximation can be performed by well known numerical methods such as polynomial or rational approximations, see [47] for some examples. The third stage extrapolates $f(x)$ from $f(\bar{x})$.

Range reduction, as the first step of computing an elementary function, has a great impact on evaluation accuracy. Almost all approximation algorithms only give a correct result if the argument is within a given bounded interval. Accordingly, to compute a function $f(x)$ for an arbitrary argument x, one first must find an auxiliary value $\bar{x}$ such that $f(x)$ can easily be restored from $f(\bar{x})$ and $\bar{x}$ belongs to the interval where the approximation holds. A poor range reduction method may lead to catastrophic accuracy problems when the input arguments are large [92]. In practice, two kinds of range reduction are possible: Additive and Multiplicative. In additive range reduction $\bar{x}$ is equal to $x - kC$, where k is an integer and C is a constant which is chosen regarding the characteristics of $f(x)$. Multiplicative range reduction, on the other hand, means that $\bar{x}$ is equal to $\frac{x}{C^k}$, where k is an integer and C is a constant which is chose considering the characteristics of $f(x)$. Combinations of these basic methods with multiple C_i constant values also can be applied to improve the range reduction in special cases. Choosing the

range reduction method depends on the function characteristics, for example additive range reduction is normally applied for trigonometric functions whereas multiplicative range reduction is more suitable for exponential and logarithmic functions [92].

Even within a reduced range interval of the input argument, the elementary functions cannot be calculated exactly, apart from in a few cases, they need therefore to be approximated by numerical algorithms. Most algorithms consist either of piecewise polynomial or rational approximations of the function being computed, or of building sequences that converge to the result.

Accurate polynomial or rational approximation to a function encompasses tradeoffs between polynomial(s) degree and result accuracy. Normally, more accurate approximations are possible by increasing the degree of the approximating polynomials. Clearly, polynomials or rational functions with higher degree computationally cost more, which means that there is a contradictory relationship between accuracy and computation cost. An approach to deal with this drawback is the table based approximation. Nevertheless, this method can hypothetically evaluate functions with any desired accuracy. Tabulating a function for all possible input values is impractical for large word lengths (let us say greater than 16-bits). In consequence, a combination of polynomial and table-based methods is an attractive approach in many practical cases (see [73] and [86]). The basic method of computing $f(x)$, after preliminary range reduction, with this approach is to first locate in the table the value x_0 that is closest to x, then finding $f(x)$ as in Equation (2.15).

$$f(x) = f(x_0) + E(x, x_0), \tag{2.15}$$

where $f(x)$ is stored in the table and $E(x, x_0)$ is a correction function which can be approximated by a lower degree polynomial or fraction function. A compromise between the size of the table and degree of the polynomial(s) on one hand, and the result accuracy on the other is a matter of optimisation, which is very dependent on the case.

Convergence methods, which are also called shift-and-add methods, are another way of approximating functions. These methods are based on simple elementary steps such as addition and shifts (multiplication by 2^i). One of the most famous methods of this class is the CORDIC algorithm, introduced by [134], which has enabled pocket calculators to compute some elementary functions. Shift-and-add methods normally require less hardware than polynomial or fraction approximations but, as is discussed in [92], they can be slower and are still not generalised to all kinds of functions.

Since algorithm specification does not refer to the implementation method of elementary functions, in the hardware implementation of computational algorithms, usually, the designer needs to make a decision about the evaluation method of elementary functions or combination of them ($\sqrt{\log(x)\sin(x)}$ for example). Accordingly, different methods should be considered to trade off realisation costs of the algorithm with its accuracy and speed.

2.4.2 Arithmetic Characteristics As An Optimisation Parameter

Very Large Scale Integrated (VLSI) circuit design and implementation has profoundly changed the size and speed of computing structures by making available an immense amount of computational resources on a single chip. The rapid increase in the size of VLSI systems, and the need to reduce the circuit development time have resulted in a need for CAD tools that can help choose the most suitable design parameters at the early stages of the design process. Accordingly, a variety of methods and approaches are presented for VLSI design.

Generally speaking, VLSI design methods can be compared with each other by the computational power of their resulting chips. The computational power of the chip is often measured by the efficient use of the silicon area, required energy and the execution time. In other words, a design is more successful if it can perform the same computation abilities as others with less area, lower power consumption and in a shorter time. These implementation costs are referred to as 'complexity' in some studies [101].

Formally, with every machine M, a complexity function Φ_M can be associated in many different ways, where $\Phi_M(x)$ is interpreted as the complexity of the computation process of M on an input x. Whereas the complexity of a machine reflects the structure of the machine itself and is always represented by a single value (a natural number), the computational complexity associated with a program is a function, whose values can be understood as the complexity of the computations described by the program.

Often, complexity theory analyses the difficulty of computational problems in terms of the number of required computational resources. It is can also be expressed in terms of the necessary amounts of different computational resources, including time, space, power consumption and other measures. In accordance with the assumed definition, a complexity class is the set of all of the computational problems which can be solved using a certain amount of computational resource. Perhaps the most well studied computational resources in this regard are deterministic time and deterministic space. These resources represent the amount of computation time and memory space needed on a deterministic computer. These resources are of great practical interest, and are well-studied [137].

From a computational complexity view, the smallest units of the mathematical functions are the elementary arithmetic operations. These operations are considered as the atomic building blocks of the computation and are considered as the basic units of complexity measurement for computational algorithms. This assumption is very viable for the classic general purpose computer-software system of computation because normally in these systems fixed pre-designed hardware is provided which needs to be programmed by software to perform a specified algorithm. Since there is a predefined set of computational or arithmetic units available in the hardware part of the system, software codes need to utilise just these units to carry out all the required operations.

On the other hand, achievements in VLSI design technology, methods and tools make it practical to customise hardware designs for specific applications. Having customised a design for an application, the performance of the system can be improved by specialising its features for that specific usage. This customisation can be applied to different features of the hardware ranging from the architecture level to the lowest level of specification. Accordingly, the classic model of the computer-software for computational complexity is not applicable if the arithmetic units of the system have also been modified for a specific application. In other words, if the elementary arithmetic operations have been adjusted for a specific application, they cannot be assumed as the atomic building blocks for complexity evaluation of the implemented algorithm. The implementation cost of an adder, for instance, is strongly dependent on the word-length of the operands and the presentation system. This means that the cost of adding two numbers is not fixed for every implementation and is a function of the arithmetic characteristics of the operands, such as word-length.

Extensions have been proposed to expand the concept of computational complexity to the physical cost of the implementation of the algorithms. Thompson in [129] investigates the complexity of the Discrete Fourier Transform (DFT) with respect to a model of computation appropriate to VLSI implementation. This model focuses on two implementation costs, the silicon area and latency to perform the DFT algorithm (see [98] for more details) on the chip. Consequently, lower bounds are proved for area and time in relation to the number of points in the DFT algorithm.

Presenting more accurate bounds for computational complexity is beyond of the scope of this work but from a high level synthesis viewpoint, the characteristics of the arithmetic units must be taken into account in high level design evaluation, especially for computationally intensive systems, because their efficiency is very sensitive to the sub-task arithmetic operations. The following subsections present a brief review of the related work in this area.

2.4.3 Floating-Point to Fixed-Point Conversion

Most algorithms are typically first proposed in terms of mathematical equations. To be solved numerically, they are usually represented in a software format which is verified in digital computers using some carefully designed test values. Most often, the underlying data types in this software representation are either single or double precision floating-point. Either one of these floating-point representations has a limited number of bits for mantissa or exponent, however, both single and double floating data types are treated as infinite-precision in practice.

In a top-down design flow to implement these algorithms, the next step is to determine the system architecture, such as the data communication style of the sub-blocks or

parallelism scheme. The details of this architectural description should reach down to the level of arithmetic operators, such as adders, multiplexers, and delays. Since a wide variety of structures and sub-blocks are available to the designer, to choose one out of several promising architectures at this level of abstraction, requires an intensive search and comparison to find the most appropriate implementation. During the search, the architecture designers have to verify that structural refinements do not amend the algorithm functionality. This takes place in floating-point simulations using the same test values as previously.

After floating-point verification of the design, the immediate task of the hardware implementation is to determine the data types that are feasible in a final realisation. A large number of digital implementations rely on fixed-point approximations to reduce hardware costs while increasing throughput rates. The designer, therefore, needs to determine the arithmetic characteristics of the physical implemented data representation of each number in the specified algorithm including the word-length (both integer word length and fractional word length), truncation mode (either roundoff or truncation) and overflow mode (either saturation or wrap-around). This task is named as Floating-point to Fixed-point Conversion (FFC) in a literature [118].

In hierarchical system design and synthesis, the development of methodologies for the automatic implementation of floating-point algorithms in fixed-point architectures is required for the minimisation of the design costs including area, power consumption, latency, accuracy and time to market.

In [62] a method is proposed for the floating-point to fixed-point conversion after code generation. The input is an assembly program using a hypothetical floating-point instruction set and a floating-point data format. Then, a floating-point simulation of the assembly program is conducted for verifying and estimating the range of each internal variable for automatic translation into a fixed-point version. The scaling that is needed for the conversion of floating-point variables and data to fixed-point is conducted based on the range of the signal through the processing algorithm. This study targets floating-point algorithms in the TMS320C25/50 fixed-point DSP of Texas Instruments. Accordingly the proposed methodology is specialised for this particular architecture and to transpose it to other kinds of architecture requires more research.

A bit-width-optimisation system for hardware/software co-design, called FRIDGE, is introduced in [60] by Keding et al. This work provides range optimisation by interval arithmetic-based range propagation and simulation-based precision analysis. The tool allows an automated, interactive transformation from floating-point ANSI-C into a bit-true specification. Annotation is the first step in this method in which user defines the fixed-point format of some variables which are critical in the system or for which the fixed-point specification is already known. Furthermore, global annotations can be defined in order to specify some rules for the entire system such as maximal data word

length and casting rules. In the second step, the word-length of the integer and fractional parts are determined for each variable, this step is called interpolation. The fixed-point data formats are obtained from a set of data interval propagation rules and the analysis of the algorithm control flow. An entire fixed-point specification of the application can be produced in this step. This description is simulated in order to verify if the accuracy constraints are fulfilled. The commercial tool CoCentric Fixed-point Designer proposed by Synopsys is based on this approach. This procedure leads to a fixed-point specification of the application in terms of behavioural VHDL, C or assembly.

The aim of the method presented in [66] and [67] is to transform a floating-point C source code into an C code with integer data types in order to be independent of the target architecture. Moreover, a fixed-point format optimisation is done in order to minimise the number of scaling operations. At the first step, the floating-point data types are replaced by fixed-point data types with the scaling operations in the output code. The scaling operations and the fixed-point data formats are determined from the dynamic range propagation information obtained from an interval analysis and the statistical method introduced in [63]. The reduction of the number of scaling operations is based on the assignment of a common format to several relevant variables to allow the minimisation of cost function of the scaling operations. This cost function takes account of the number of occurrences of each scaling operation and depends on the scaling capacities of the target processor. For a processor with a barrel shifter, the cost of a scaling operation is equal to one cycle; otherwise the number of cycles required for a shift of n bits is equal to n cycles. In that work, the code execution time is not optimised under a global accuracy constraint. The accuracy constraint is only specified through the definition of the maximal accuracy degradation allowed for each variable. Specific elements of the architecture are not taken into account for optimising the fixed-point data formats. Moreover, the architecture model used to minimise the scaling operations does not take account of the specialised shift registers which allow specific shift operations without supplementary cycles. Furthermore, for processors with instruction level parallelism capacities, the overhead due to scaling operations depends on the scheduling step and cannot easily be evaluated before the code generation process. Some similar simulation-based work has been reported by Leong et al in [76].

In [17] Cantin et al. presented a method to determine the word length required by implementations of Digital Signal Processing algorithms. The method uses a C/C++ fixed-point simulation tool to model the impact of finite word length on overall accuracy. It finds a combination of quasi-optimum bit resolutions in the corresponding data flow graphs by computing dissimilarities between fixed-point and floating-point simulation results. The selected algorithm minimises these dissimilarities and finds a combination of word lengths that meets the objectives specified by the user. This method is a stimuli-based dynamic error analysis method similar to the method is presented in [126] and

[66]. The major difficulty of these dynamic error analysis methods is their optimisation time which makes them impractical for complex designs.

Menard et al. presented a method of implementation of Digital Signal Processors under accuracy constraints in [84] and [85]. In their method, the DSP architecture is taken into account for optimising the execution time under accuracy constraints. At the first stage, the method evaluates the variables dynamic range in the data flow gràph by applying interval arithmetic range analysis. The results obtained are used to determine the decimal-point of the data in order to avoid an overflow. Then, the word-lengths of each variable are defined. Finally, the data formats are optimised in order to minimise the code execution time as long as the accuracy constraint is fulfilled. The determination and optimisation of the data formats are made under accuracy constraint. The Signal to Quantisation Noise Ratio (SQNR) is used for evaluating the accuracy.

In [3] and [4] Banerjee et al. described a behavioural synthesis method which reads in high-level descriptions of digital signal processing applications written in MATLAB, and automatically generates synthesisable register transfer level models and simulation test-benches in VHDL or Verilog. Conversion of the floating point computations in MATLAB to a fixed-point MATLAB version of specific precision for hardware design is performed automatically. The RTL models can be synthesised using commercial logic synthesis tools. Experimental results are reported on a set of MATLAB benchmarks that are mapped onto the Xilinx Virtex II and Altera Stratix FPGAs.

Shi and Brodersen in [119] proposed a tool that automates the floating-point to fixed-point conversion process for digital signal processing systems. Their method optimises fixed-point data types of arithmetic operators, including overflow modes, integer word lengths, fractional word lengths, and the number representation systems. The approach is based on statistical modelling, hardware resource estimation and global optimisation from an initial structural system description. This technique exploits the fact that the fixed point realisation is a weak perturbation of the floating point realisation which allows the development of a system model which can be used in the optimisation process. In addition to word-length, this work adds overflow mode (wrap-round or saturation) and number system (2's complement or unsigned) to the arithmetic characteristic parameters which can be selected during the optimisation process. This work only tries to linearise the nonlinear system and does not provide any new model for error. It is also not clear what is the complexity of their method to find the Jacobean matrix of the linearised system. Furthermore, in the best case they can not only find variance of the computational noise in the output, error range nor errors PDF are not achievable with these methods.

Doi et al. in [38] presented a nonlinear programming method for floating-point to fixed-point conversion in high level synthesis. An Affine Arithmetic error model is employed for error propagation in the data flow graph which is integrated into a nonlinear problem

specification. In this optimiser, the bit-width of the functional units is considered as the implementation cost.

2.4.4 Word Length Optimisation

Over the past years, attempts have been made to demonstrate that using an optimal word-length for functional units, less than the worst-case assumption, at different points in the datapath, would dramatically save implementation costs.

In [21], a combined method of static and dynamic analysis was proposed. Cmar et al. employ interval propagation analysis for range width determination and a simulation-based method for precision bit-width optimisation. Their simulation is utilised by a concurrent program which performs the same calculation as a reference and as a custom fixed-point format, and compares the error between the two values. For precision evaluation, the first and second moments of the error at each signal point are examined. Since the C specification of the DSP algorithm needs major modifications for this analytical/simulation based method, the bit resolution analysis is performed only on one operand instead of on combinations of operands. The idea behind such an approach is that it sets an upper-bound on the word length of each variable, beyond which the least significant bits will be drowned in quantisation or external noise. No additional mechanism is proposed to automate the tradeoff of system area against error.

Kum and Sung [68] introduced several heuristic word length optimisation methods to trade-off system area against Signal-to-Quantization-Noise Ratio (SQNR). The techniques are heuristics based on bit-true simulation of the design under various internal word lengths. This method measures the performance of a fixed-point algorithm using simulation results. A reference system is designed without overflows or signal quantisation effects and it iteratively modifies the word length of a signal to find a set of optimum word lengths satisfying the fixed-point performance. Signal grouping is used to reduce the number of simulations to calculate the uniform word-length and minimum word length configuration. Starting from the minimum word length configuration, they perform an exhaustive or heuristic search to get the optimised configuration of word-lengths could then a high level synthesis is performed based on the minimum word length information, while the final word length optimisation is conducted using the synthesised hardware models. Since the design area is the only objective of the design, there is no suggestion in the work to use other design costs, power consumption for instance, as optimisation objectives.

Constantinides et al. focused on developing algorithms for word length optimisation in several works [24, 28, 26, 25]. These methods employ static analytical digital noise analysis for DSP applications applied to Linear Time Invariant (LTI) systems implemented as custom processing units. Optimisation techniques are proposed which allow the user

to trade off implementation area for arithmetic error at the system outputs. Optimality with respect to the area and error estimates is guaranteed through modelling using mixed integer linear programming; however, the optimisation time for complex designs is expected to be high.

Constantinides later extended the previous efforts to nonlinear components in a datapath by employing a small signal approach and investigating the effect of precision optimisation on power reduction as a by-product of the word length optimisation [28]. [120] also introduces a similar method based on perturbation theory for nonlinear systems. Again in these works, power consumption is not an objective in the optimisation heuristic.

Sulaiman and Arslan [125] presented a Multi Objective Genetic Algorithm (MOGA) for WL and power consumption in a Fast Fourier Transform (FFT) processor. The GA was used to find FFT coefficients which have optimum performance in terms of Signal-to-Noise Ratio (SNR) and power consumption. The results demonstrate that the GA can find solutions which are optimised for both objectives, but this work does not offer a general optimisation method for DSP algorithms. There are studies, on the other hand, which introduce methods to improve the speed or power consumption of the communication on the shared buses, including bus splitting.

Nayak et al. in [93] presented a compiler that takes high-level signal processing algorithms described in MATLAB and generates optimised hardware. Their precision analysis algorithm determines the minimum number of bits required by a model of errors through the data flow graph, where a set of error transfer functions determines the error contribution of each node. Data range optimisation is performed by a data range propagation technique. They concluded that this method reduces the required hardware resources significantly.

In [111] Sanghamitra and Banerjee presented an approach to automate the conversion of floating-point MATLAB programs into fixed-point MATLAB programs for mapping to FPGAs by profiling the expected inputs in order to estimate errors. The algorithm attempts to minimise the hardware resources while constraining the quantisation error within a specified limit. A binary search approach is utilised to arrive quickly at a coarse optimal point. The fine optimisation starts from the coarse optimal point to reduce the complexity to vary linearly with the number of quantisers. This method is based on a dynamic analysis which uses the stimuli for error analysis.

In [71] Le-Gal et al. proposed a methodology that employs an annotated formal model with bit-width information and dynamic range values in order to extract bit-wise information to optimise the area and power consumption of hardware architectures provided by high-level synthesis tools. This method is based on two steps. First, a bit-width analysis of the application according to the input information provided by the designer is performed. This bit-width information is propagated through a graph which models the

application to compute the lower-bound and upper-bound values to each computation and the required memory. The resulting annotated graph enables datapath structure optimisations for high-level synthesis with a processing time complexity of $O(n)$. Once all bound computation is performed, the necessary bit-widths to model data and to implement the operations are evaluated. This information is then used during the high level synthesis process. The second part of the method relates to the high-level synthesis process. High-level synthesis is used to formally transform the application into an architecture observing a set of constraints. Then an architecture optimisation stage is completed in order to adapt both possible operator and register bit-widths of the design.

Han and Evans [52] also reported a sensitivity and complexity approach to word length optimisation. In this work they discussed a pre-planned exhaustive search which utilises the sensitivity information of hardware complexity and the system output error with respect to the signal word length. Word length design case studies for a wireless demodulator were implemented to show a meaningful improvement in the optimisation speed. In their case studies, the optimised word-length was considered as the improvement measure.

Several works report applications of symbolic analysis in computational error analysis [94]. A basic implementation of this method is known as Affine Arithmetic (AA). In this method, unlike Interval Arithmetic (see section 3.3.1 for more details), noise source dependency is taken into account in a parametric representation of the error at different points in the DFG. In [72] Lee et al. implemented an AA-based method which categories the problem into two parts, range analysis and precision evaluation. The former gives the integer part of the data whereas the latter provides the fractional part of the variables at every point on the DFG. An Adaptive Simulated Annealing (ASA) heuristic is applied to find the near-to optimal points. Similar to this work, a study was reported by Pu and Ha [106] which applied AA with a different heuristic. In the later work, inspired by [42], by applying the central limit theorem, the first and second moments of the output noise are approximated from a symbolic representation of the output noise.

Since one way to reduce the power consumption in an Application-Specific Integrated Circuit (ASIC) implementation is to reduce the bit-width of the computation units, works have been presented focusing on this method for low power design. Mallik et al. in [80] and [81] describe algorithms to optimise the bit-widths of fixed-point variables for low power design in a SystemC-based ASIC design environment. An optimal bit-width allocation algorithm for two variables and a greedy heuristic is proposed in which converting the floating-point programs into ASIC synthesisable SystemC specifications is automated. The expected inputs are profiled to estimate errors in the finite precision conversions. Experimental results for the tradeoffs between quantisation error, power consumption, and hardware resources are reported which demonstrate that it is possible to reduce the power consumption by 50% on the average by allowing roundoff errors to increase from 0.5% to 1% in the benchmarks.

2.5 Summary

In this chapter, background knowledge related to the realisation of computationally intensive hardware as well as a comprehensive literature surveys on datapath synthesis, bus-oriented design and floating-point to fixed-point conversion is provided.

The computational complexity of hardware implementation is investigated and the effect of minimising the cost of the arithmetic operations. From this viewpoint, classic computational complexity is arguable in that computational cost of the algorithms cannot be taken down to the elementary arithmetic operations because their cost might vary depending on the implementation styles in the target hardware. Accordingly, the arithmetic characteristic is introduced as a design optimisation parameter, especially in the case of computationally intensive hardware. Floating-point to fixed-point conversion and word-length optimisation are comprehensively reviewed as closely related fields in which the arithmetic characteristic of functional units is the foremost optimisation parameter in high level synthesis.

The next chapter provides a focused review concerning uncertainties in computation and introduces different methods of how they can be modelled and treated in digital computation.

Chapter 3

Computing With Uncertainty

3.1 Introduction

Uncertainty in computation is always a possibility with the practical problems. From a computing system stand point, the origin of this uncertainty can be external or internal. External uncertainties are inherited from data acquisition or any preceding system which are imported together with the input data in the form of input error or noise. Internal uncertainties, on the other hand, arise from the current processing system because of computation limits and constraints. Since in many applications one or both kinds of these uncertainties are inevitable, the error effects of these phenomena on the computation process needs to be investigated.

Due to data dependency, computing the exact ranges of the output(s) variation in a sequence of operations is a complex task particularly when there are dependencies between intermediate variables. Theoretically the only general method to figure out the exact range is an exhaustive simulation of all the possible input vectors of the system. Since full cover of the input space is impossible in practice, usually estimates of the ranges are obtained through partial simulation of the input space. However, these estimates are not guaranteed to include all possible results and it must be assumed that the stimuli vectors truly represent the expected input values. Accuracy analysis of the computational systems has been investigated, with many different approaches which can be classified in many different ways, and one such classification is static analysis versus dynamic analysis.

Dynamic analysis methods are based on the use of a set of precisely chosen input stimuli to cover the worst cases of uncertainty in the data computation, whereas static analysis relies on analytical models of the error that functional units produce and the way that errors propagate through the system to the output. Since dynamic methods are very time consuming and based on simulation, this study is based on static error analysis and this chapter provides a review of this issue and the related methods and works.

Error models and the previous work in this regard are reviewed for different types of word-length refitting. These models either have a statistical specification or have an error-bound symbolic specification which makes it possible to keep track of the errors from their origin to eliminated or emerge those which come from the same source. Major symbolic error models are reviewed including Interval Arithmetic (IA) and Affine Arithmetic (AA) [94].

Statistical noise analysis is another approach which is inspired by analogue circuit noise analysis. In this approach, computational errors are considered similar to independent random noise sources. Each functional unit has an independent random noise generator in its output which produces a noise with magnitude that is dependent on the maximum computational error in that functional unit. These noise signals propagate through the system exactly like signals from independent sources. This method is very useful in LTI DSP applications, however, it does not consider the error dependency. It also requires some modifications for nonlinear systems.

Since our work is based on combining arithmetic characteristics with HLS parameters, we have to investigate the causes and effects of the computational errors. This chapter provides a brief review on computation error modelling and analysis methods.

3.2 Analysing Finite Precision Effects

Digital hardware can only provide a limited number of binary digits for all the numbers in computational algorithms. This limitation affects the represented values in two ways: discretisation and bounding. Discretisation embodies a precision limit which indicates the minimum distance between two neighbour values in the sequence of represented numbers, whereas the bounding limit refers to the maximum and minimum values which can be represented by this hardware. When real values must be fitted to this constraint the result will be the original value along with an unwanted, and normally unknown, value considered as error or noise. In most practical computation systems, there are many of this kind of error picked up and carried by data from point to point through the computation tree of the algorithm.

From a systematic viewpoint, uncertainty in computation may originate from two sources: external and/or internal sources. Since input data of every computational system either comes from another computational or a data acquisition system there is always a possibility that it carry some errors due to the finite precision of the previous system. Quantisation of the filter coefficients in DSP applications is a well known example of this type of error. Internal uncertainties, on the other hand, arise from the current processing system because of computation limits and constraints. Rounding or truncation of the results in arithmetic operations is an example of this kind of error. Since in many applications one or both kinds of these uncertainties are inevitable, the computation

process needs to be reinvestigated concerning error effects of these phenomena. Modelling and dealing with these uncertainties are subjects of variety of disciplines ranging from numerical analysis to computer software and hardware design.

As a matter of fact, it is not trivial to analyse the finite precision errors in actual designs. Computational errors in functional units are data dependent in that different input data sets can produce different patterns of errors in the system. Statistical distribution of the input data, therefore, can cause a great impact on the actual accuracy of the system. Furthermore, since finite precision effects are nonlinear, it is observable that computational errors in algorithms can be dependent on the sequence of the local functions such as arithmetic operations. It means that during the high level synthesis process (see section 2.2), subsystem allocation and scheduling might affect the predicted error in the output if this dependency has not been considered. This complexity of the problem has inspired the research community to introduce different methods of error analysis and design optimisation. There are a variety of approaches to word-length optimisation in high level synthesis in this regard, which can be classified in many different ways.

Dynamic analysis [52] relies on the use of stimuli input data sets. This method usually is performed by applying a set of possible inputs to the system and continuously comparing its calculated results with an auxiliary program which is performing the same algorithm but in high precision arithmetic. Since this approach directly compares the overall accuracy of the system with an ideal calculation, when compared to static-analysis techniques, it potentially provides bit-width(s) estimation for functional units close to the optimal set for those particular stimuli. However, in complicated systems, number of possible inputs (2^n vectors for an n input system) can be so high as to make it impossible to exhaustively search the input space for worst case errors. Hence this method can be problematic because a large set of stimuli inputs is required to analyse a complex design with sufficient confidence, possibly leading to prohibitively long simulation times and without guarantees for alternative input stimuli encountered in practice.

Static analysis is often more attractive than dynamic analysis especially for large designs, since only the characteristics of the input signals are needed, however, it is believed to give more conservative bit-width estimates than dynamic analysis [52]. This method is based on an analytical model of the errors and their propagation scheme in the computation tree of the algorithms. Errors can be modelled in the form of random values with a known probability Density Function (PDF), or conversely variation range of the data might be modelled as number intervals or ranges. Either way, the propagation scheme of these models must be inspected to find out the output value of the error or data range and its relationship with the local error sources. In spite of its simplicity, there are difficulties which this method needs to cope with.

In most of the computational noise analysis methods, such as DSP applications, error sources are considered as independent Wide-Sense Stationary (WSS) noise sources with

uniform PDF [98, 104]. This assumption has a great impact on the method efficiency; however, there are arguments about different issues in this regard. For instance, in many practical computations intermediate results are strongly dependent on each other which can violate the independency assumption of the error sources.

Error propagation model in static methods is also an important issue. Inspired by system theory studies, the primary works in this field were based on a linear Time Independency (LTI) assumption of the computational systems [99]. In practice, nevertheless, many of the computational algorithms result in nonlinear system specifications. Consequently, applying methods that are designed for LTI systems cannot be used. Furthermore, there are applications that are required to be implemented in the form of time dependent or adaptive systems. In these cases, the LTI model is not valid and cannot be applied to the error analysis method. And finally, considering HLS methods in combination with computational error analysis raises a new set of problems for these analysis methods.

To cope with aforementioned problems, different approaches have been introduced and investigated. The following sections provide a brief review on major works in this regard.

3.3 Error Bound Analysis Approach

Understanding of the required range of variable values, at different points of a given computation procedure, is vital to compute the minimum bit-width requirements in a hardware implementation of an algorithm. It is also gaining more importance in the customised hardware design methodologies, where underlying substrates allow construction of customisable variable bit-width datapaths.

Assuming that the variation ranges of the input variables are known, this approach tries to predict the variation range of the output data. Therefore, instead of a single value, every number is represented by a range of values between the upper and lower bounds. In other words, regardless of the real place of the number in the range, these methods are concerned with data range dilation and contraction by data propagation through the operations in the computation tree of the algorithm. These data ranges are normally presented in the form of symbolic specifications. Since this method is focused on value bounds, there is no information about how the actual value might be placed in the range. A simple representation of this method is depicted in Figure (3.1).

The following subsections provide a brief review of different variants of interval range analysis methods. Three methods are presented here: Interval Arithmetic (IA), Affine Arithmetic (AA) and the Taylor Representation of range.

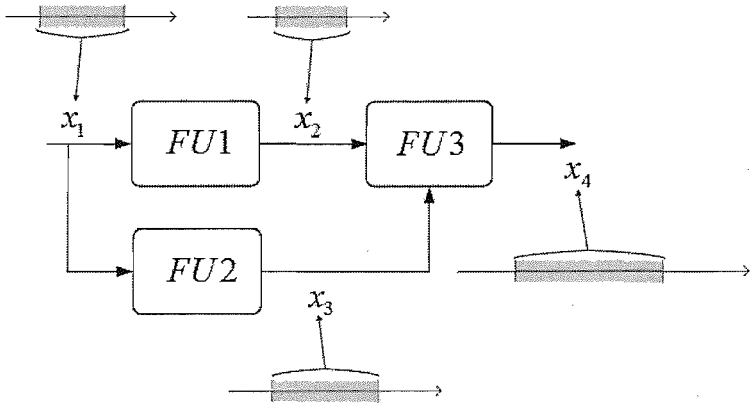


FIGURE 3.1: Variables bound propagation through the algorithm computation tree.

3.3.1 Interval Arithmetic

Interval Arithmetic (IA) was introduced in 1960's by Ramon E. Moore as a PhD thesis at Stanford University and later on, in his works [89, 90] as a tool for automatic control of the errors, in a computed result, that arise from input error rounding errors during computation, and truncation errors from using a numerical approximation to the mathematical problem. In his study not only the foundation of interval arithmetic has been laid; it also offers interval methods for a variety of problems, such as interval version of Newton's method [47] and automatic differentiation and algorithms for the validated solution of Ordinary Differential Equations (ODE).

IA is a data range analysis method in numerical computation from which every real value x is represented by an interval $\hat{x} = [x_l, x_h]$, where $x_l \leq x \leq x_h$. Arithmetic operations are performed on these intervals in such a way that each result interval $\hat{y}$ is guaranteed to contain the value of the actual quantity y. Semantically, IA representation starts from an closed interval of numbers. Consider a set S_I of real interval numbers defined as in Equation (3.1).

$$S_I = \{[x_l, x_h] | (x_l \leq x_h) \wedge (x_l, x_h \in \mathbb{R})\}\,, \tag{3.1}$$

where x_l is the lower bound of the interval and x_h is its upper bound, $\mathbb{R}$ is the set of real numbers and $\wedge$ represents the *logical AND*.

Arithmetic operations on real numbers can be extended naturally to real interval numbers. Let $\circ$ denote a two operand, real, arithmetic operation. If $\hat{x}, \hat{y} \in S_I$ then the formal specification of $\circ$ is as in Equation (3.2).

$$\hat{x} \circ \hat{y} = \{z | (z = x \circ y) \wedge (x \in \hat{x}) \wedge (y \in \hat{y})\}\,, \tag{3.2}$$

Specifically elementary arithmetic operations with two independent uncertain values represented in IA form as $\hat{x} = [x_l, x_h]$ and $\hat{y} = [y_l, y_h]$, have

$$\hat{x} + \hat{y} = [x_l + y_l, x_h + y_h], \tag{3.3}$$

$$\hat{x} - \hat{y} = [x_l - y_h, x_h - y_l], \tag{3.4}$$

$$\begin{aligned} \hat{x} \times \hat{y} = \Big[& \min\{(x_l \times y_l), (x_l \times y_h), (x_h \times y_l), (x_h \times y_h)\}, \\ & \max\{(x_l \times y_l), (x_l \times y_h), (x_h \times y_l), (x_h \times y_h)\} \Big], \end{aligned} \tag{3.5}$$

$$\frac{\hat{x}}{\hat{y}} = \left[\min\left\{\frac{x_l}{y_l}, \frac{x_l}{y_h}, \frac{x_h}{y_l}, \frac{x_h}{y_h}\right\}, \ \min\left\{\frac{x_l}{y_l}, \frac{x_l}{y_h}, \frac{x_h}{y_l}, \frac{x_h}{y_h}\right\} \right], \text{ assuming: } 0 \notin \hat{y}, \tag{3.6}$$

These formula ignore rounding, overflow and other computational errors. A comprehensive set of elementary functions implemented in IA can be found in [123].

In general, for every function $y = f(x_1, x_2, \cdots, x_n)$ where $f : \mathbb{R} \to \mathbb{R}$; a corresponding IA specification, $\hat{y}$, is defined such as: $\hat{y} = \hat{f}(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$. This operation returns the interval that contains all the values of $f(x_1, x_2, \cdots, x_n)$, where variables $\{x_1, x_2, \cdots, x_n\}$ are independent and range over the given intervals $\{\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n\}$.

Practically, it is observed that the computed error bound can be overestimated by IA because the self-dependency of the data values are not considered in IA. Thus interval arithmetic flils to identify different occurrences of the same variable bounds during the computation algorithms. In other words, since there is no trace of the error sources in the intervals, error sources which come from parallel branches in the computing tree might share some common components that are not considered in the interval computation. As a trivial example, consider $x - x = 0$ which holds for each $x \in [1, 2]$, but $\hat{x} - \hat{x}$ for $\hat{x} = [1, 2]$ results in [-1, 1] in the IA method. There is also another source of overestimation which is called *wrapping effect*. This effect appears when intermediate results of a computation are enclosed into intervals [96]. The following example makes this effect more clear.

Example 3.1. *Consider the function* $f : (x, y) \to \frac{\sqrt{2}}{2}(x + y, y - x)$. *The image of the square* $[0, \sqrt{2}]^2$ *is the rotated square with corners* $(0, 0), (1, -1), (2, 0), (1, 1)$. *On the other hand, interval computation yields* $f([0, \sqrt{2}], [0, \sqrt{2}]) = ([0, 2], [-1, 1])$, *these regions are depicted in Figure (3.2).*

Note that the observed overestimation (the area of the interval enclosure is twice the area of the range) is not a result of dependency, but of the enclosure of a non-interval shaped range into an interval. Overestimations of this kind are one of the major problems in the interval arithmetic treatment of differential equations [123].

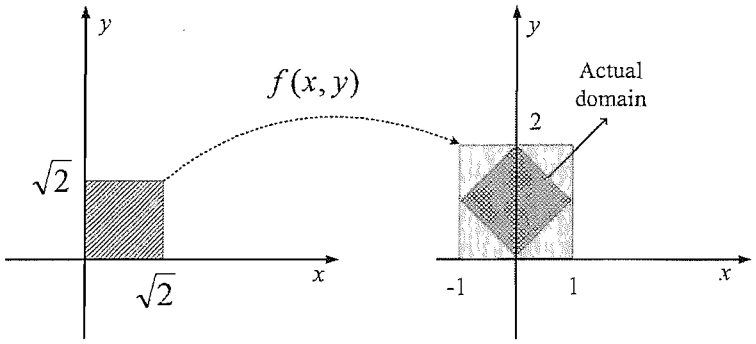


FIGURE 3.2: Wrapping problem in IA method.

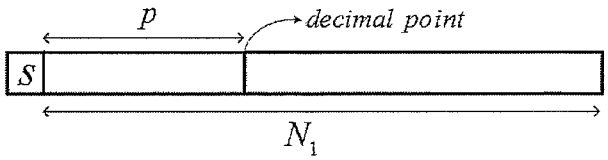


FIGURE 3.3: Standard fixed-point representation.

To build up a complete method to cope with computational error optimisation, finite precision errors also must be modelled in IA. Let us assume x is represented in fixed-point by N_1-bit word-length with decimal point positioned at $p > 0$, as depicted in Figure (3.3). If x comes from a quantisation unit, the quantisation error can be modelled in IA in Equation (3.7).

$$\hat{x} = \left[-2^{-p}, 2^{-p}\right], \tag{3.7}$$

To model the truncation error assume that the same x is refitted into an N_2-bit word-length register, thus the error are modelled as the IA range in Equation (3.8).

$$\hat{x} = \left[-2^{-p}(2^{-N_2} - 2^{-N_1}), 2^{-p}(2^{-N_2} - 2^{-N_1})\right], \tag{3.8}$$

IA computing, as a pioneering method in computing with uncertainties, is a very useful method with low computation overhead. It can easily be implemented by overloading elementary operations with IA operations and representing numbers in the form of ordered pairs. The major downside of the IA is its very conservative error range estimation. This weak-point especially can be exaggerated in practical computation tasks, where in long computation chains the computed intervals in one stage are inputs for the next stages. This conservatism is mainly due to this assumption that the unknown values of the arguments to the primitive operators vary independently over the given intervals. If this assumption is not true, which it is not mostly and then having correlation between possible results, not all the combinations of values in the given interval will appear in the results set. In that case, the interval obtained by IA can be much wider than the exact range of the result quantity, which known as "error explosion" [32]. Affine arithmetic is

an extension of IA to cope with some of these problems. This method is discussed in the following subsection.

3.3.2 Affine Arithmetic

Affine arithmetic (AA) is a method for numerical computation that aims to attack the dependency problem in interval computations. AA keeps track of first-order correlations between computed and input quantities; these correlations are automatically exploited in primitive operations, with the result that in many cases AA is able to produce interval estimates that are much better than the ones obtained with standard interval arithmetic. Moreover, AA also implicitly provides a symbolic representation for the joint range of related quantities that can be exploited to increase the efficiency of interval methods.

A partially known quantity x in AA is represented by an affine form $\hat{x}$, which is a first-degree polynomial representation as in Equation (3.9)

$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + ... + x_n\epsilon_n, \tag{3.9}$$

where the coefficients x_i are finite real numbers and ϵ_i are symbolic real variables whose values are unknown but assumed to lie in the interval $U = [-1, +1]$. x_0 is called the central value of the affine form $\hat{x}$. The coefficients x_i are partial deviations and ϵ_i are symbolic representation of the uncertainty in variables. In Equation (3.9) each ϵ_i stand for an independent component of the total uncertainty of the quantity x. The corresponding coefficients x_i represents the magnitude of each component. In this definition, the uncertainty source might be either internal (arithmetic rounding for instance) or external (indeterminacy of the input data such as quantisation error) [123].

To model the elementary operations this method divides all operations into two categories: affine operations and non-affine operations. The first one refers to those which result in a linear representation of the error symbols in the output whereas the later represents those which result in nonlinear combination of the symbols, multiplication for instance.

3.3.2.1 Affine Operations

If the operation $f(x, y)$ itself is an affine function of its arguments x and y, then f can be expanded and rearranged into an affine combination of the symbols ϵ_i. Specifically, for any given constants $\alpha, \beta, \zeta \in \mathbb{R}$, the computation $z \rightarrow \alpha x + \beta y + \zeta$ can be carried

out by affine expansion of x and y as in Equation (3.10).

$$\begin{aligned} \hat{x} &= x_0 + \sum_{i=1}^{n} x_i \epsilon_i, \\ \hat{y} &= y_0 + \sum_{i=1}^{n} y_i \epsilon_i, \end{aligned} \tag{3.10}$$

then $\hat{z}$ can be represented as in Equation (3.11).

$$\hat{z} = \alpha\hat{x} + \beta\hat{y} + \zeta = (\alpha x_0 + \beta y_0 + \zeta) + \sum_{i=1}^{n} (\alpha x_i + \beta y_i)\epsilon_i, \tag{3.11}$$

Except for roundoff errors and overflows, the affine form $\hat{z}$ above, together with $\hat{x}$ and $\hat{y}$, captures all the information about the quantities x, y, and z that can be deduced from the given affine forms $\hat{x}$ and $\hat{y}$, and the equation $z = \alpha x + \beta y + \zeta$.

An AA computation that uses only affine operations with known constant coefficients will usually give an almost-exact range for the ideal result. In fact, it will give an almost-exact explicit formula for the ideal result in terms of the input variables.

3.3.2.2 Non-Affine Operations

The most important primitive operations that are not affine are multiplication and division. If we write $\frac{x}{y} = x \times (\frac{1}{y})$, then we can concentrate on multiplication. Given two affine forms of Equation (3.10) their product is as Equation (3.12).

$$\begin{aligned} \hat{x} \cdot \hat{y} &= \left(x_0 + \sum_{i=1}^{n} x_i \epsilon_i\right) \cdot \left(x_0 + \sum_{i=1}^{n} y_i \epsilon_i\right), \\ &= x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + y_0 x_i)\epsilon_i + \sum_{i=1}^{n} x_i \epsilon_i \cdot \sum_{i=1}^{n} y_i \epsilon_i, \end{aligned} \tag{3.12}$$

which can be rewritten as the following affine form for the product:

$$\hat{x} \cdot \hat{y} = x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + y_0 x_i)\epsilon_i + z_k \epsilon_k, \tag{3.13}$$

where :

$$z_k \geq \left| \sum_{i=1}^{n} x_i \epsilon_i \cdot \sum_{i=1}^{n} y_i \epsilon_i \right|, \quad \epsilon_i \in U, \tag{3.14}$$

is an upper bound for the approximation error, as before. The simplest bound is:

$$z_k = \sum_{i=1}^{n} |x_i| \cdot \sum_{i=1}^{n} |y_i|, \tag{3.15}$$

which is at most four times the error of the best affine approximation, but is very easily computed.

AA can be used for precision analysis in a similar manner as for range analysis. Truncation and rounding can cause a maximum error of 2^{-p} (1 ULP) and 2^{-p-1} (0.5 ULP), respectively. Truncation chops bits off the least significant bits and requires no extra hardware resources. Round to nearest involves a small adder followed by truncation. Hence, the quantisation error of finite precision representation ($\hat{x}$) of a real value x is given in affine form as in Equation (3.16).

$$\hat{x} = \begin{cases} x + 2^{-p} \cdot \epsilon & \textit{Truncation} \\ x + 2^{-p-1} \cdot \epsilon & \textit{Rounding} \end{cases}, \tag{3.16}$$

And for data refitting from N_1-bit specification as depicted in Figure (3.3) into an N_2-bit specification, the AA error range is represented as:

$$\hat{x} = x + 2^{-p}(2^{-N_2} - 2^{-N_1}) \cdot \epsilon, \tag{3.17}$$

AA method solve the data dependency problem of the IA method, accordingly it can give better approximations for error bounds, however it requires more computations efforts because of its more complicated structure. On the other hand, since AA is based on a linear specification of the error symbols its offered method is restricted in the case of nonlinear operations, to solve this problem Taylor method provides a more sophisticated representation which is capable of modelling many nonlinear error symbols, this method is presented in the following subsection.

3.3.3 Taylor Method

For reducing both the dependency problem and the wrapping effect, interval arithmetic has been improved with symbolic extensions. Affine arithmetic is a simple method which provides a linear combination of the error symbols and can be used to model the elementary arithmetic operations, but in the case of more complicated elementary functions which are not linear and normally can be evaluated in the form of polynomial or fraction functions, see section 2.4.1.4, AA method cannot give a precise approximation. A rigorous multivariate Taylor arithmetic has been developed by Berz and his group since 1990s [79, 9] to cope with these difficulties.

A Taylor model of a function f on some interval $\hat{x}$ consists of the Taylor polynomial P_n of order n of f and an interval remainder term I_n. This model encloses the approximation error $|f - P_n|$ on x. In computations that involve f, the function is replaced by $P_n + I_n$. The polynomial part is propagated by symbolic calculations where possible and the interval remainder term is processed according to the rules of interval arithmetic. All

truncation and roundoff errors in intermediate operations are also enclosed into the remainder interval of the final result. The interval Taylor models for a differentiable function $f : x \subseteq \mathbb{R}^m \rightarrow \mathbb{R}$ can be derived in the form :

$$f(x) \in P_n(x - x_0) + I_n, \tag{3.18}$$

where $P_n(x)$ represents a degree-n polynomial in the m variables $x \in \mathbb{R}^m$, x_0 is the bias point and I_n represents the interval that compasses the interval error representation of polynomial truncation and also possible errors in calculation of the coefficients of P_n. Ideally, polynomial coefficients should be chosen in a way to minimise the error enclosure.

To represent the case more formally, let f be a function on m variables: $f : [-1, 1]^m \rightarrow \mathbb{R}$, a Taylor model of order n for f is a pair (P_n, I_n) where P_n is the Taylor expansion of order n for f at the point $x_0 = (0, \cdots, 0)$ and I_n is an interval enclosing the truncation error, I_n will also be called the interval remainder of the Taylor model. This interval remainder is required to satisfy the high order scaling property in that if the function f_h is defined for $-1 \leq h \leq 1$, by $f_h(x) = f(h \times x)$ and its remainder bound is determined as $I_{n,h}$, then as $h \rightarrow 0$, the width of $I_{n,h}$ behaves as $O(h^{\omega+1})$. I_n could be computed as a Lagrange remainder as:

$$I_n = [-\alpha, \alpha] \ \ with \ \ \alpha = \frac{1}{(n+1)!}||f^{(n+1)}||_\infty = \int_{-1}^{+1} f^{(n+1)}(t)\frac{(x-t)^n}{n!}dt, \tag{3.19}$$

where the $||\ ||_\infty$ norm is taken over $[-1, 1]^m$. However, determining I_n from a Lagrange remainder is in practice very difficult so it is not applicable in the most of the cases. It suffices that the scaling property and the following containment property hold as in Equation (3.20) [110].

$$\forall x \in [-1, 1]^m, \quad f(x) \in [P_n(x), P_n(x) + I_R], \tag{3.20}$$

To simplify the notations and algorithms, without loss of generality all considered Taylor models are considered as having the same order n, which must be in practice be less or equal to the minimum of their actual orders.

In addition to choosing the optimal coefficients for approximation polynomials for the range of a single function, Taylor models can be utilised to implement symbolic representation of the computational errors. Taylor arithmetic should also be defined on Taylor objects in such a way that evaluation of an expression for $f()$ in this arithmetic gives the Taylor polynomial with remainder term for $f(x)$. For an introduction to these concepts and techniques, see [110].

A Taylor model is a convenient way to represent and manipulate a function on a computer in [110]. Since any function that is obtained by a sequence of arithmetic operations is analytical which means that it and all its derivatives are differentiable, then it can

be expanded into Taylor series. Thus, it is reasonable to restrict the target functions to analytical functions [94]. Acordingly, various operations can be performed on Taylor models, such as elementary arithmetic operations $(+, -, \times, /)$, elementary functions $(\sqrt{}, log, sin, arctan, cosh, \cdots)$, composing Taylor models, integrating or differentiating them and so on. Several examples can be found in [79, 9, 110].

This method is the most sophisticated method among error range modelling methods with more precise error approximation and also higher computing overhead. To our best of knowledge it only has been implemented in scientific software applications [110]. The basic quantisation and refitting error models in this method are exactly the same for AA method as specified in Equations (3.16) and (3.17). Basic examples to compare these methods can be found in section 4.2.

3.4 Noise Analysis Approach

Error range propagation methods provide an expression of the error range at the output to evaluate the computation accuracy. This approach does not provide any statistical information about the error distribution over this range. Noise analysis approach, on the other hand, gives a statistical description of the error at the output in the form of noise energy or signal-to-noise parameter. This method is widely practiced in digital signal processing design and optimisation [99, 28].

Applying noise analysis consists of two parts: noise model and noise propagation. The first one refers to the error modelling of the functional units and the arithmetic characteristics under finite precision effects. Functional unit operation type $(+, -, \times, \cdots)$, arithmetic system (fixed-point/floating point), word-length, rounding method and overflow prevention method are the major parameters which need to be taken into account in noise modelling. In ideal case, every computational error in each functional unit is an independent random noise generator. To compute the overall effect of the errors at the output, therefore, these noise sources are considered to propagate through the system like signals and their spectrum in the output is calculated using noise sources and the system structure. The following subsections discuss these issues in more detail.

3.4.1 Modelling Digital Noise

From a mathematical point of view, computational error can be modelled in the form of an additive error, which means that adding or removing unwanted and indeterministic information to the input data, appears at the output as an added noise value (signal) to the original value (signal). In digital implementation of the (DSP) algorithms, depending on the way of interaction with these computational noises, they have been divided into two major categories: quantisation noise and round off error. However both of these

errors arise from the finite precision limitation, the way they affect the system are different and thus require different solutions. Quantisation noise is considered as an external source of the noise in that input data carries it before entering the computation system, whereas round off noise has internal sources which result from the finite precision restrictions of the arithmetic units which are trying to refit the data to the their finite word length. We prefer to call this kind of error as a "data refit error".

One of the input noise sources inherited from all previous computational operations or data acquisition systems. Every error imported to the system by input data will be carried processed along side with the original information. Depending on the type and characteristics, these errors might be strengthened or weakened compared with the original information through data processing. The most known and well studied type of error which appears almost everywhere in computational systems is quantisation error. This error is investigated widely in the field of DSP and coding [51, 50].

Another external source of the error comes from coefficient quantisation. Every system has a set of constant values representing the basic system characteristics, filter coefficients for instance. These values, however, are real numbers ($x \in R$) in reality but they must be used by a digital computing system. With a limited number of binary digits, truncation or rounding of these values is needed which causes an error that is considered also as quantisation noise. When parameters of the system function (or its corresponding difference equation specification) are quantised, the poles and zeros of the system function might move to new positions in the Z-plane, which in the case of a system implementation highly sensitive to the perturbation of the coefficients, may mean the resulting system might no longer meet the design specifications, or in the case of IIR system even might become unstable, detailed explanations can be found in DSP texts such as [98].

In additive noise model, a finite word-length unit can be modelled in terms of an ideal unit followed by an additive noise source as shown in Figure (3.4). Three distinct approximation schemes can be employed after fitting the output data into the available word-length of the functional unit by rounding, truncation or magnitude truncation. Each of these approximation methods gives a different noise source probability density function as depicted in the Figure (3.5).

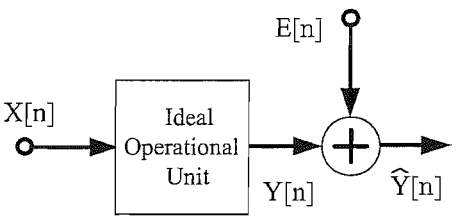


FIGURE 3.4: Additive computational noise model.

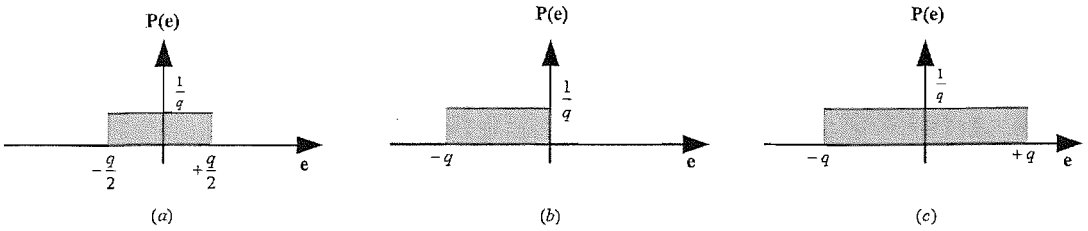


FIGURE 3.5: Probability density function for quantisation error a)Rounding b)2's complement truncation c)Magnitude truncation.

In the case of rounding, the output number must be fitted to a finite precision number whose value is the closest possible to actual number as depicted in Figure (3.5-a) [37]. The mean or expected value of the quantisation error due to rounding is zero and the variance of the error signal ($E(n)$) as in Equation (3.21),

$$\sigma_E^2 = \frac{q^2}{12} = \frac{2^{-2N}}{12}, \tag{3.21}$$

where q is the maximum value of the rounding error and N is the word-length of the target unit such that $q = 2^{-N}$. In the case of truncating a number, where the result is always less than the original value, as depicted in Figure (3.5-b), the mean value of the error is shown in Equation (3.22).

$$E\{E(n)\} = -\frac{2^{-N}}{2}, \tag{3.22}$$

and the error variance is as in Equation (3.23).

$$\sigma_E^2 = \frac{2^{-2N}}{12}, \tag{3.23}$$

If magnitude truncation is applied to 2's complement numbers, the probability density function has the form of the Figure (3.5-c). In this case the mean value of the error is zero and the variance is in Equation (3.24)

$$\sigma_E^2 = \frac{2^{-2N}}{3} \tag{3.24}$$

In addition to input noise, during computation with digital systems in most of the cases after every arithmetic operation the actual result needs to be fitted into the output word length. This operation might be a source of error if the result requires more precision than can be represented by the output.

The round off noise is generated by the rounding or truncation operation that follows the various arithmetic operations. It mostly happens in the case of multiplication where the result of the product of two N-bit fixed-point fractions is a $(2N-1)$ bit number and it must eventually be quantised to M-bits ($M < 2N-1$) by rounding or truncation. In

the systems with fixed word-length the truncation is done by built in units as shown in Figure (3.6).

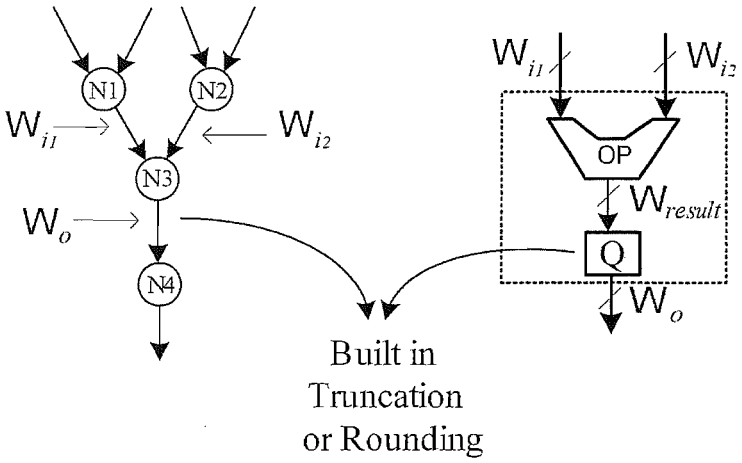


FIGURE 3.6: Built in operation output truncation off

According to discussions in [28] the model of Equation (3.22) needs correction for a multiple word-length implementation. For 2's complement fixed point arithmetic the truncation error cause by truncation from N_1-bit to N_2-bit has mean value and variance as shown in Equations (3.25) and (3.26) respectively.

$$E\{E(n)\} = -2^{p-1}\left(2^{-N_2} - 2^{-N_1}\right), \tag{3.25}$$

$$\sigma_E^2 = \frac{1}{12}2^{2p}\left(2^{-2N_2} - 2^{-2N_1}\right), \tag{3.26}$$

As a common assumption, these noise sources have been formulated as independent white noise in DSP design [99]. By this assumption, knowing the variance and mean value will be enough to provide an acceptable approximation for the noise produced in the system output(s), however, this assumption necessitates some preliminary conditions.

Sripad and Snyder in [122] present the required necessary and sufficient condition to model the output of a quantiser as an infinite precision input and an additive, uniform, white noise. In this work the statistical properties of the quantisation error are studied and it is proved that the characteristic function of the input random variable satisfies:

$$\Phi_x\left(\frac{2\pi n}{\Delta}\right) = 0 \qquad \text{for all} \quad n \neq 0, \tag{3.27}$$

if and only if the density function of the quantisation error is uniform such as:

$$f_\epsilon(E) = \begin{cases} \frac{1}{\Delta} & -\frac{\Delta}{2} \leq E < \frac{\Delta}{2} \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

The characteristic function is defined as in Equation (3.29), see [117] or similar basic probability texts for more details.

$$\begin{aligned} \Phi_x(u) &= E\left[\exp(jux)\right], \\ &= \int_{-\infty}^{+\infty} f(x)e^{jux}dx, \end{aligned} \tag{3.29}$$

where $E[\cdot]$ represents the expectation function [54] and $j = \sqrt{-1}$. Equations (3.27) and (3.28) imply that the quantisation error can be assumed as a uniform noise if the input signal satisfies the condition of the Equation (3.27). It is also shown in [122] that if this condition does not apply to the input signal, the model of Equation (3.28) can be used but its difference with the actual PDF of the quantiser is given by

$$\frac{1}{\Delta}\sum_{n\neq 0}\Phi_x\left(\frac{2\pi n}{\Delta}\right)\exp\left(\frac{-2\pi jnE}{\Delta}\right), \tag{3.30}$$

Furthermore, error noises can be assumed independent if and only if the joint characteristic function of the input random variables satisfies

$$\Phi_{x_1,x_2}\left(\frac{2\pi l}{\Delta}, \frac{2\pi k}{\Delta}\right) = 0 \qquad \text{for all} \quad l,k \neq 0, \tag{3.31}$$

Roundoff error after fixed-point multiplication is also commonly modelled as uniformly distributed white noise that is uncorrelated with the signal [28]. In [6] Barnes et al. present a statistical analysis of fixed-point roundoff error that identifies the conditions under which this model is valid, and examines the statistical behaviour of the roundoff error when these conditions are not satisfied. Their study shows that if the multiplier coefficient is expressed as $a = \frac{N}{M}$, where M is a positive integral power of two and N is an odd integer, then the errors generated by roundoff after multiplication can generally be modelled as uniformly distributed, white, and uncorrelated with the signal, if the signal has sufficiently wide bandwidth and has a dynamic range that extends over approximately M steps. For narrow-band low-level signals, the roundoff error statistics can differ significantly from the uniform, white, uncorrelated model. In addition, their results show that statistical behaviour of roundoff error can differ significantly from that of the quantisation error that is generated when a continuous random variable is quantised. In this study the dynamic range of the input signal is measured by $\frac{\sigma}{\Delta}$, where σ^2 is the variance of the signal and Δ represents the quantisation step size.

Also in this regards, a general statistical analysis is presented in [130] by Tokaji and Barnes for the roundoff error that is generated when a discrete random multiplicand, taking on only integer values, is multiplied by a real coefficient and the result rounded back to the nearest integer. Numerical results for different statistical joint moments verify that for signals with sufficiently large dynamic range ($\frac{\sigma}{\Delta}$) and bandwidth, the roundoff error after multiplication can be accurately modelled as white noise that is

uncorrelated with the signal, a result that is well known and widely used. However, at low signal levels, the white uncorrelated model may be significantly in error, since the roundoff error statistics are highly dependent upon the value of the multiplier coefficient, and deviations from the white, uncorrelated model are greatest for multipliers in the neighbourhood of integers and rational numbers with small denominators. These results are generalised by Vladimirov and Dimond in [133] which provides a mathematical discussions regarding noise dependency to the operation type and the system characteristics.

3.4.2 Noise Propagation

In addition to the noise model, the propagation scheme of the noise is required to calculate the noise effect at the output. From a system analysis viewpoint, this issue can be studied for two kinds of systems: linear and nonlinear. Linear systems, however, rarely exist in practice, but are the simplest model by which the input/output relationship is expressed with a linear relationship. It is also considered as the basis of many nonlinear systems analysis methods. Nonlinear models, on the other hand, in most of the cases provide more accurate but very complex models. The following subsections are devoted to the modelling of noise propagation through the linear and nonlinear computation trees.

3.4.2.1 Linear Time Invariant Systems

In mathematics, a linear function, $f(x)$, is one which satisfies both of the following properties which are referred to as the principle rules of superposition [98]:

- Additivity: $f(x+y) = f(x) + f(y)$
- Homogeneity: $f(\alpha x) = \alpha f(x)$

and a linear system is one which is specified by a linear relationship between its input and output. In other words, a linear system responds to any weighted sum of two input sequences with an output sequence equal to the corresponding weighted sum of the individual output sequences.

A time-invariant system is one whose output does not depend explicitly on time in that its response to a time-shifted input sequence is an equally time-shifted output sequence. For instance, if the input signal x produces an output y then any time shifted input, $t \mapsto x(t+\delta)$, results in a time-shifted output $t \mapsto y(t+\delta)$. A formal specification for time invariant system can be:

> If S is the shifting operator $S_\delta[x(t)] = x(t-\delta)$, then the operator T is called time-invariant, if $T\{S_\delta[x]\} = S_\delta[T\{x\}]$.

This property can be satisfied if the transfer function of the system is not a function of time except as expressed by the input and output. This property can also be stated in another way in terms of a schematic such as: "If a system is time-invariant then the system block is commutative with an arbitrary delay" [98].

From HLS viewpoint, most synthesisers use a DFG to represent the algorithms to be implemented. Accordingly, the most popular way to analyse the noise propagation through the system operations is independent of the noises produced in each operation in the system and to model them as white noise source which have been separated in the DFG as in [28] and [126].

From basic theory of system analysis it is known that in a Linear Time Invariant (LTI) system with M different inputs, the output $y[n]$ can be expressed as [99, 105]:

$$y_k[n] = h_k[n] * x[n] = \sum_{r=0}^{\infty} h_k[r] \cdot x[n-r], \tag{3.32}$$

and:

$$y[n] = \sum_{k=1}^{M} y_k[n], \tag{3.33}$$

where $*$ is the convolution operator and $h_k[r]$ denotes the impulse (unit sample) response for the k$^{\text{th}}$ source in the system. Therefore the output noise of the system can be computed by replacing the $x[n]$ with noise sources in the DFG and $h_k[r]$ with the unit sample response from the noise insertion point to the output(s). Figure (3.7) shows an example of a two-pole IIR filter with inserted noise sources. However, while this relationship is mathematically acceptable, its implementation for complicated cases is impractical.

One of the ways by which an adequate measure has been introduced is explained in [99] based on the above equation and L_p *norm* of $H(\omega)$ (the Fourier transform of the transfer function h[n]), where L_p is as in Equation (3.34).

$$L_P\{H(\omega)\} = \left[\frac{1}{2\pi}\int_{-\pi}^{\pi} |H(\omega)|^p \, d\omega\right]^{\frac{1}{p}}, \tag{3.34}$$

As it explained in [99], given a normal distribution assumption for all noise sources, the variance of the output noise, resulted from the k$^{\text{th}}$ noise source, can be approximated as:

$$E_k = \sum_{k=1}^{M} \sigma_k^2 \cdot L_2^2\{H_k(z)\}, \tag{3.35}$$

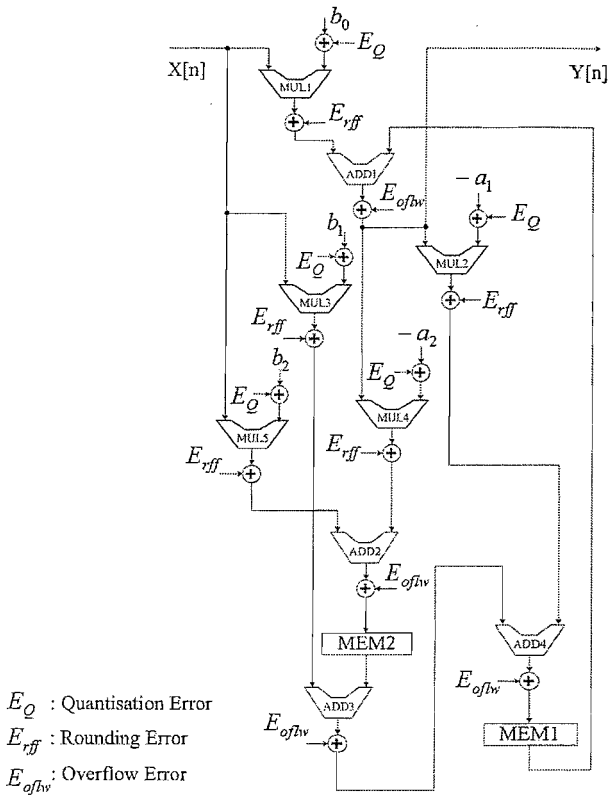


FIGURE 3.7: Two Pole IIR Filter structure with its Noise Sources.

where M is the number of noise sources, $H_k(z)$ is the Z transform, see [98] for more details, of transfer function from k^{th} noise source to the output. According to the definition, $L_2\{H(z)\}$ can be written as in Equation (3.36):

$$L_2\{H(z)\} = \left[\frac{1}{2\pi}\int_{-\pi}^{\pi} |H(\omega)|^2\, d\omega\right]^{\frac{1}{2}} = \left[\sum_{n=0}^{\infty} \left|Z^{-1}\{H(z)\}[n]\right|^2\right]^{\frac{1}{2}}, \tag{3.36}$$

In addition σ_k can be found according to [28] in a multiple word-length paradigm as shown in Equation (3.26).

3.4.2.2 Non-Linear Systems

In its most general form, the input-output relationship of a system can be expressed as

$$Y = F(X, t), \tag{3.37}$$

where $F(\,)$ specifies the input-output relationship, X is the input vector and Y represents the output vector both are as shown

$$Y = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{bmatrix}, \quad X = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_m(t) \end{bmatrix}, \quad y_k(t) = f_k(X,t), \tag{3.38}$$

$F(\,)$, unlike LTI systems, can be a nonlinear and/or time dependent function. Thus, in general, the superposition rule is not applicable any more and the input-output relationship can not be expressed as in Equations (3.32) and (3.33). Nonlinear models are inevitable in many different applications which necessitate their inspection in computational system analysis and optimisation. Since analysing nonlinear systems in the general form is very difficult in engineering, the first approach usually is to linearise; in other words, to try to avoid the nonlinear aspects of the problem.

In order to make some of the analytical results on computational error analysis for LTI systems applicable to nonlinear systems, the first step is to linearise these systems. The assumption is made that the quantisation errors induced by rounding or truncation are sufficiently small not to affect the basic behaviour of the system. Accordingly, any changes in word-length of the functional units in a nonlinearly specified computation algorithm cause a change in the relationship between X and Y of the form

$$Y = F(X,t) \longrightarrow Y' = F'(X,t), \tag{3.39}$$

$$Y' = Y + \triangle Y, \tag{3.40}$$

$$Y' = \begin{bmatrix} y'_1(t) \\ y'_2(t) \\ \vdots \\ y'_n(t) \end{bmatrix}, \quad y'_k(t) = f'_k(X,t), \tag{3.41}$$

where Y' represents the changed output in result of the word-length modification and is the output produced noise. These noise values can be expanded as

$$\triangle Y = G(X,E,t), \qquad \triangle Y = \begin{bmatrix} \triangle y_1(t) \\ \triangle y_2(t) \\ \vdots \\ \triangle y_n(t) \end{bmatrix}, \quad E = \begin{bmatrix} e_1(t) \\ e_2(t) \\ \vdots \\ e_n(t) \end{bmatrix}, \tag{3.42}$$

Since this produced error is a function of the input data and the word-length manipulation in the system, it can be modelled as a function of the input (X) and errors which have been produced in each node in consequence of arithmetic characteristic changes;

accordingly the new output is as in Equation (3.43).

$$Y' = F(X,t) + G(X,E,t) \tag{3.43}$$

The major problem with this method is specification of the function G in Equation (3.43). In [28] and [99] only LTI systems are considered, thus G can be represented by a linear relationship. This assumption is true for many systems, however, there are important classes of problems which do not comply with this condition (adaptive filters for instance). To overcome this restriction, two different methods are applied. Kim and Sung in [68] suggest a simulation based approach which bypasses the modelling problem, this approach is slow in the case of big designs. On the other hand, others, such as Constantinides in [25], propose a method which assumes that each component in the system can be locally linearised, see Table (3.1), in order to determine the output behaviour under a given rounding scheme. This approach approximates G based on the assumption that the noise signal is very small in comparison with the original signal. Equations (3.44) and (3.45) illustrate this approximation based on a Jacobean (Taylor) approximation of function G.

$$\triangle Y = G(X) \approx A \times X, \tag{3.44}$$

where A is the Jacobean of the $G(x)$ as in Equation (3.45).

$$A = J_G(x) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_m} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \cdots & \frac{\partial g_n}{\partial x_m} \end{bmatrix}, \tag{3.45}$$

This approach again has the difficulty in which the new (simplified) model elements, matrix A_{ij}, need to be found. We must bear in mind that the elements of A are dependent on input values and also time. Constantinidis, [25], suggests a simulation-based method to find the values. Shi, [119], proposes a similar method to find the noise expectation values at the output by applying a first order perturbation approach.

TABLE 3.1: **Linearity and Time Dependency of the building blocks.**

Operation	Representing FU	Function	$\frac{\partial}{\partial x_1}$	$\frac{\partial}{\partial x_2}$	$\frac{\partial}{\partial t}$
$+$	ADS	$f(x_1,x_2) = x_1 + x_2$	1	1	0
$-$	ADS	$f(x_1,x_2) = x_1 - x_2$	1	1	0
$\times$	MUL	$f(x_1,x_2) = x_1 \times x_2$	x_2	x_1	0
$\times$	MUL	$f(x_1,x_2) = a \times x_2$	a	-	0

3.5 Summary

Background knowledge related to the finite precision computing has been investigated in this chapter. Accordingly, a comprehensive literature survey on computational error modelling and analysis is provided to build up the required theoretical basis for the proposed method.

Finite precision computing always results in uncertainty in computation. However this uncertainty generally comes from digital system limitations, it can be categorised by its origin and behaviour. Different issues regarding these uncertainties are extensively studied in related disciplines of science and engineering. From a computing system point of view; there are two major approaches to model and analysis this problem in digital hardware design: error bound analysis and statistical error analysis, both of which are introduced and discussed in this chapter.

Error bound analysis relies on the fact that the basic arithmetic operations map a "bounded continuous interval" (or simply an interval) into a "bounded continuous interval". So, if an uncertain value can be modelled in the form of an interval, the output also will be an interval which definitely contains the actual value. By generalising this rule, this method calculates the output bounds of the result value. The value bounds can be represented in different ways such as: number intervals or symbolic representations of the ranges. The benefits and downside of each way are discussed in the chapter.

The statistical approaches, on the other hand, treat the errors emphasising the probability and statistical characteristics of the data and errors. This approach is very similar to noise analysis in analogue electronic circuits and consists of two basic steps: noise models and noise propagation structures. The first one refers to the PDF shape and statistical specifications of the error in every point of the computation tree, whereas the latter is concerned about how the noise is propagated through the system.

Next chapter presents a discussion with motivating examples to show weak points of these methods and a new method will be offered to cover these weak points.

Chapter 4

Symbolic Noise Analysis

4.1 Introduction

Chapter 3 provided an overview of the computation error modelling and analysis. As discussed, the current methods can be divided into two types: error bound analysis and statistically based methods, where the former are focused on the error bounds model and propagation in the system and the latter analyse the noise statistical expectation functions. Actually, each method provides a part of the information which can be useful in computational system design and optimisation. In this chapter a new method, called Symbolic Noise Analysis (SNA), is introduced that suggests a combination of both error bound and statistical analysis methods.

This method is based on symbolic modelling of the error bounds where the error symbols are considered to be specified with a probability distribution function over a known range. The noise symbol propagation scheme is quite similar to the error bound propagation structure in the error bound analysis methods, however, noise symbols can be combined and characterised in the form of different probability density functions. Accordingly at the system output the error can be portrayed in terms of the error bounds and noise spectra. This information provides a more comprehensive view of the error for a designer.

Although this method is based on a static analysis method, it can be used in combination with a dynamic approach as well. Since the method is based on noise symbol propagation, noise symbols for FUs or sub-blocks can be derived from a stimulus-based analysis in the form of a noise PDF. These stimuli-based noise symbols should be accommodated in the design library by the noise characteristics of the FU or sub-block to be used by the optimiser.

The chapter is organised as follow: first motivation examples are presented to show the limitations of the current known methods, then a new method of error analysis is

introduced and compared with the other methods. Symbol combination is discussed in the next section to show how the noise symbols can be merged to simplify the noise specification. Finally the implementation algorithm is provided to illustrate the method in practice.

4.2 Motivations

Nonlinear systems are real challenges for computational error analysis methods. Different methods have been modified and improved to deal with these difficulties, however, a perfect solution is still required (see chapter 3 for more details). Since most of the function evaluation algorithms are based on polynomial approximation [92], to highlight the problem and limitations let us consider the general form of polynomial functions and see how different methods might come up with different results for the computational error at the output.

Consider a degree-N polynomial as

$$z = \sum_{i=0}^{N} a_i x^i. \tag{4.1}$$

Uncertain input values are represented by the $\hat{x}$ and $\hat{a}_i$ respectively, which result in an uncertain output as

$$\hat{z} = \sum_{i=0}^{N} \hat{a}_i \hat{x}^i, \tag{4.2}$$

IA representation of the uncertainty, see section 3.3.1, in the input and coefficient values can be expressed in the form of $\hat{x} \in [x_l, x_h]$ and $\hat{a}_i \in [a_{il}, a_{ih}]$ and $\hat{z}$ bounds can be calculated from

$$\hat{z} \in \left[Min\left\{ \sum_{i=0}^{N} \hat{a}_i \hat{x}^i \right\} \ , \ Max\left\{ \sum_{i=0}^{N} \hat{a}_i \hat{x}^i \right\} \right], \tag{4.3}$$

Finding the exact output interval involves with the *Minimum-Maximum Finding* of the polynomials over the input and coefficients variation range, which is not trivial. Instead of searching for minima and maxima, the IA method performs a constructive bound approximation starting from interval arithmetic operations as discussed in section 3.3.1.

In contrast, the AA representation of the uncertainty, see section 3.3.2, starts form an affine representation of the x and coefficients (a_i) as shown in Equation (4.4).

$$\begin{aligned} \hat{x} &= \frac{x_l + x_h}{2} + \left(\frac{x_h - x_l}{2} \right) \epsilon_x = x_0 + x_1 \epsilon_x, \\ \hat{a_i} &= \frac{a_{il} + a_{ih}}{2} + \left(\frac{a_{ih} - a_{il}}{2} \right) \epsilon_{a_i} = a_{i0} + a_{i1} \epsilon_{a_i}, \end{aligned} \tag{4.4}$$

which results in $\hat{z}$ as:

$$\hat{z} = \sum_{i=0}^{N} (a_{i0} + a_{i1}\epsilon_{a_i})(x_0 + x_1\epsilon_x)^i. \tag{4.5}$$

After expansion, all the non-affine combinations of the noise symbols such as multiplications between them or powers must be modelled in the form of new noise symbols. This manipulation simplifies the error representation but reduces the accuracy of the error approximation. The Taylor model on the other hand, see section 3.3.3, keeps Equation (4.5) as it is to represent the uncertainty.

Example 4.1. *Consider the following quadratic equation,*

$$y = ax^2 + bx + c, \tag{4.6}$$

with the input and coefficient error bounds as $x \in [-1, +1], a \in [9, 10], b \in [-4, -6], c \in [6, 7]$ *and* $\Delta y = \hat{y} - y$ *with an uniform PDF over their ranges.*

IA analysis gives the following limits for the intermediate values and the output:

$$\begin{aligned} ax^2 &\in [-10, 10], \\ bx &\in [-6, 6], \\ y = ax^2 + bx + c &\in [-10, 23], \end{aligned}$$

In terms of AA symbolic error representation of the values, values of Equation (4.6) must be rewritten as in Equation (4.7) with this assumption that the center of the interval is the main value and and number variation over the range is represented in the form of symbolic values ϵ.

$$\begin{aligned} a &= a_0 + a_1\epsilon_a = 9.5 + 0.5\epsilon_a, \\ b &= b_0 + b_1\epsilon_b = -5 + \epsilon_b, \\ c &= c_0 + c_1\epsilon_c = 6.5 + 0.5\epsilon_c, \\ x &= x_0 + x_1\epsilon_1 = 0 + \epsilon_1, \end{aligned} \tag{4.7}$$

Accordingly, the output value will be derived in terms of exact and symbolic errors as

$$\begin{aligned} y &= (9.5 + 0.5\epsilon_a)(\epsilon_1)^2 + (-5 + \epsilon_b)(\epsilon_1) + (6.5 + 0.5\epsilon_c) \\ &= 6.5 + 0.5\epsilon_c - 5\epsilon_1 + 9.5\epsilon_1^2 + 0.5\epsilon_a\epsilon_1^2 + \epsilon_b\epsilon_1 \\ &= 6.5 + \underbrace{(0.5\epsilon_c - 5\epsilon_1 + 9.5\epsilon_{k1} + 0.5\epsilon_{k2} + \epsilon_{k3})}_{\Delta y}, \end{aligned} \tag{4.8}$$

which means that $y \in [-10, 23]$.

Nonlinear system noise analysis is also applicable here. This method starts from representation of y as:

$$y = f(x, a, b, c) = ax^2 + bx + c, \tag{4.9}$$

which expands to first order

$$\begin{aligned} y &\approx f(x_0, a_0, b_0, c_0) + \Delta x\frac{\partial y}{\partial x} + \Delta a\frac{\partial y}{\partial a} + \Delta b\frac{\partial y}{\partial b} + \Delta c\frac{\partial y}{\partial c}, \\ &\approx y_0 + \Delta y, \end{aligned} \tag{4.10}$$

where Δy can be modelled in terms of noise with variance of σ_y. As explained in section 3.4.2.2, noise variance is calculated in the perturbation method [25, 120] by simulation over a set of known data to find the differentiation terms in Equation (4.10). The error variance will be:

$$\sigma_{\Delta y} \approx 16.5 \tag{4.11}$$

however it does not provide any information about the PDF of the noise or the error bounds, explicitly.

The actual error bound can be achieved using mathematical methods and tools. In this case we found the maximum and minimum values of the function $y = f(x, a, b, c) = ax^2 + bx + c$ over the variation ranges of the x, a, b and c. In Figure (4.1) the maximum and minimum valued functions are depicted versus x using Maple. As is attainable from graph, the real range of the output is $[5, 23]$.

This example shows how the distribution of symbols over the predefined range can affect the final prediction of the error. It is also a matter of concern if the linear combination of the noise symbols is blindly used to calculate the error bounds, which might result in overestimated error bounds. In this example the refitting (truncating or rounding) error of the arithmetic operations is not considered, however it is very important in practical implementations.

Another problem with the current methods of error analysis is their ignorance of the probability distribution function of the computational noise. The uniform PDF assumption for all noise sources is arguable but still is acceptable with some conditions. Difficulties arise when even with this presumption the arithmetic combination of the noise values in the final result is very unlikely to be similar to uniform. The following example shows the effect of the elementary arithmetic operations on the output PDF. Results are in contrast with the naive assumption of noise analysis methods in which all the noise values are uniformly distributed over the range.

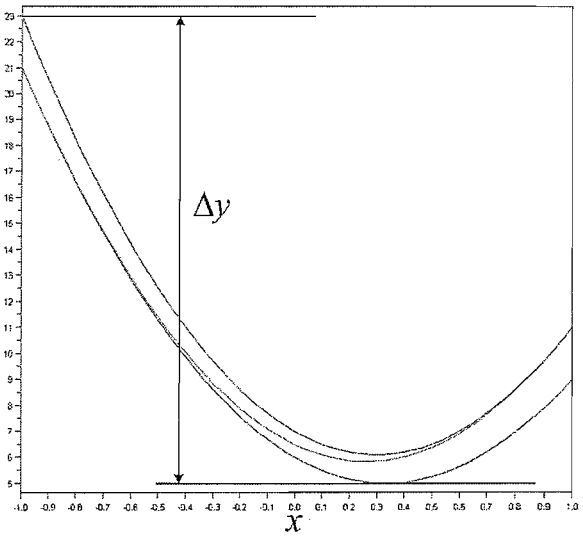


FIGURE 4.1: Real range of y deviation in for the indicate range of values in Example (4.1).

Example 4.2. *Assume $\hat{x}$ and $\hat{y}$ are two values with uncertainties which are abstracted in the affine representation as:*

$$\hat{x} = x_0 + x_1\epsilon_1 \qquad and \qquad \hat{y} = y_0 + y_1\epsilon_2, \tag{4.12}$$

where ϵ_1 and ϵ_2 are two values with interval specifications $\epsilon_1, \epsilon_2 \in [-1, +1]$. ϵ_1 and ϵ_2 are also assumed over these ranges with uniform PDF distributions. Three basic cases are considered, addition, multiplication and square:

$$\begin{aligned} \hat{x} + \hat{y} &= (x_0 + x_1\epsilon_1) + (y_0 + y_1\epsilon_2), \\ \hat{x} \times \hat{y} &= (x_0 + x_1\epsilon_1) \times (y_0 + y_1\epsilon_2), \\ \hat{x}^2 &= (x_0 + x_1\epsilon_1) \times (x_0 + x_1\epsilon_1), \end{aligned} \tag{4.13}$$

Referring to the basic probability theory [54] the PDF graph of the values in the output of the each operation are depicted in Figure (4.2).

Actually, the arithmetic combinations between noise symbols ϵ_1 and ϵ_2 cannot be assumed uniform, consequently the mean and variance of the PDFs after addition, multiplication and square have a similar shapes with Figure (4.2) and their mean value and variance are:

- Addition of ϵ_1 and ϵ_2: $\mu = 0 \quad \sigma^2 = \frac{2}{3}$,
- Multiplication of ϵ_1 and ϵ_2: $\mu = 0 \quad \sigma^2 = \frac{1}{9}$,
- Square of ϵ_1: $\mu = \frac{1}{3} \quad \sigma^2 = \frac{4}{45}$,

where these values for a uniform distribution over $[-1, +1]$ are:

$$\mu = 0 \quad \sigma^2 = \frac{1}{3},$$

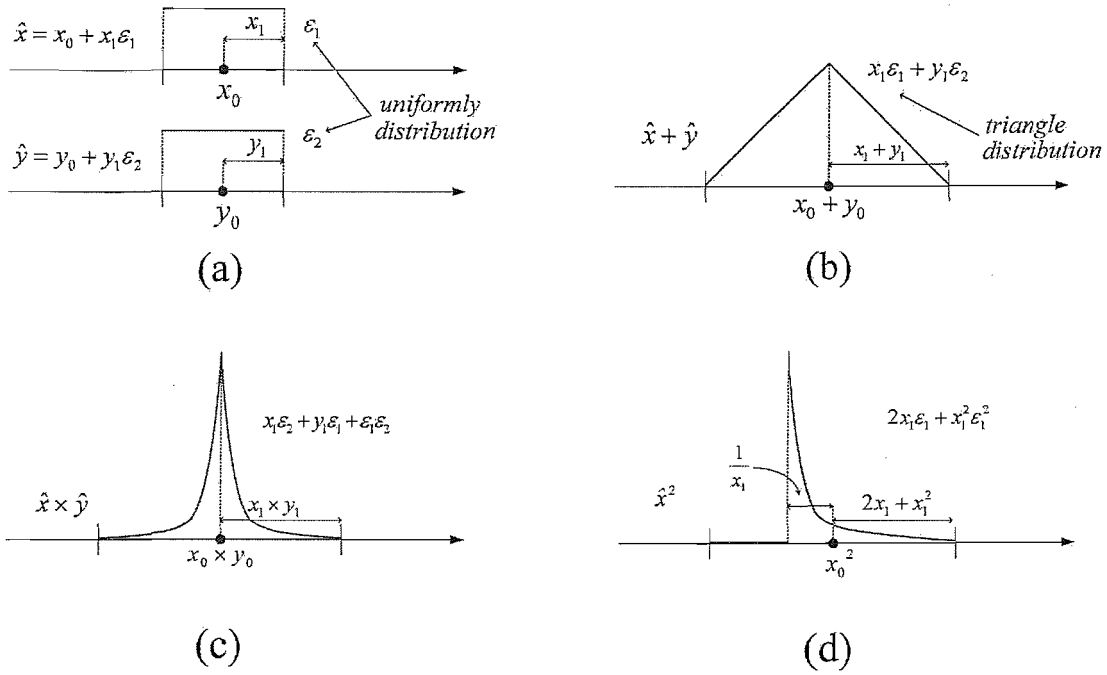


FIGURE 4.2: PDF after noise symbols combination a)Uniform PDF variables b)Addition of the variables c)Multiplication of the variables d)Square of the variable $(x_1 > \sqrt{2} - 1)$.

This example proves that it is not always an accurate assumption that the PDF of the noise sources are uniform after operations on them. It can be considered as a major weak point for noise analysis methods.

4.3 Symbolic Noise Analysis Approach

When accurate and precise numerical information is not available, the computation task is desired to be evaluated with the available information; however it might result in less exact results. Frequently, such information about the values is presented in the form of intervals bounding around the actual but unknown values. In many cases also, the information is in the form of a probability density function, which describes the relative likelihoods of the actual values.

Reconsidering the basic error analysis methods and the error range analysis methods, errors are modelled in the form of a set of error symbols which propagate through the computation tree of the system and the error variation range in the output can

be calculated but there is no useful information about the statistical characteristics of this error value. The noise analysis method, on the other hand, relies on statistical specification of the error. These two approaches have the capability to be combined to make a new approach which benefits from the positive points of both. Dealing with errors as symbols which have some statistical information with them is the core idea of the method.

To compare different methods of error representations in errors bound evaluation, a visual presentation of each method is useful. Consider a general function such as $y = f(x)$ as depicted in Figure (4.3) to perform a calculation on an uncertain input such as x. Figure (4.3-a) shows the case when the input uncertainty is represented in the form of a plain interval. Presumably, since the input representation does not support any more information than the uncertainty range, the output cannot be anything more than an interval in which the actual value will be placed. This is what IA method provides in the best case. What is depicted in Figure (4.3-b) is an AA representation of the uncertain value in terms of symbolic uncertainty. In spite of its improved error representation, again no information regarding the probability distribution of the error is provided.

Our method, on the other hand, can be represented as depicted in Figure (4.3-c) in which uncertainty is represented in the form of an algebraic combination of the noise symbols, where the noise symbols are considered as random numbers in the range $[-1, +1]$. Equation (4.14) shows uncertain values representation in this method.

$$\hat{x} = F_x\left((x_1, x_2, \cdots, x_N), (\epsilon_{x_1}, \epsilon_{x_2}, \cdots, \epsilon_{x_N})\right) \text{ where } \epsilon_{x_i} \in [-1, +1]. \tag{4.14}$$

In this representation $F_x(\cdot)$ is a fractional function of polynomials (which is called an algebraic function) and $(x_1, x_2, \cdots, x_N)$ are the coefficients of the polynomials (constants). $(\epsilon_{x_1}, \epsilon_{x_2}, \cdots, \epsilon_{x_N},)$ are *Noise Symbols* which carry the uncertainty of the represented value, $\hat{x}$, and each symbol ϵ_i is assumed to have a known PDF, P_{ϵ_i}. Unlike the AA method or the Taylor model, the PDFs of the noise symbols are taken into account, in which a PDF can be found for the output uncertainty to show the probability of the output taking each value inside the bounded interval. Figure (4.3-d) shows the general case where input noise symbols are also assumed to have an arbitrary PDF extracted from empirical data or simulation results. However the presented method provides more information about the computational noise and uncertainty, it requires more complex algorithms and methods to be implemented in the system analysis and optimisation procedures. One of the essential requirements is a method to deal with PDF functions. Therefore, the practicality of the error analysis method depends on the answers of two fundamental questions. First: how can noise symbols be represented in the synthesis method? and second: how can PDF of these noise symbols be used to find the output error PDF? The rest of this chapter is devoted answering these question regarding our proposed method.

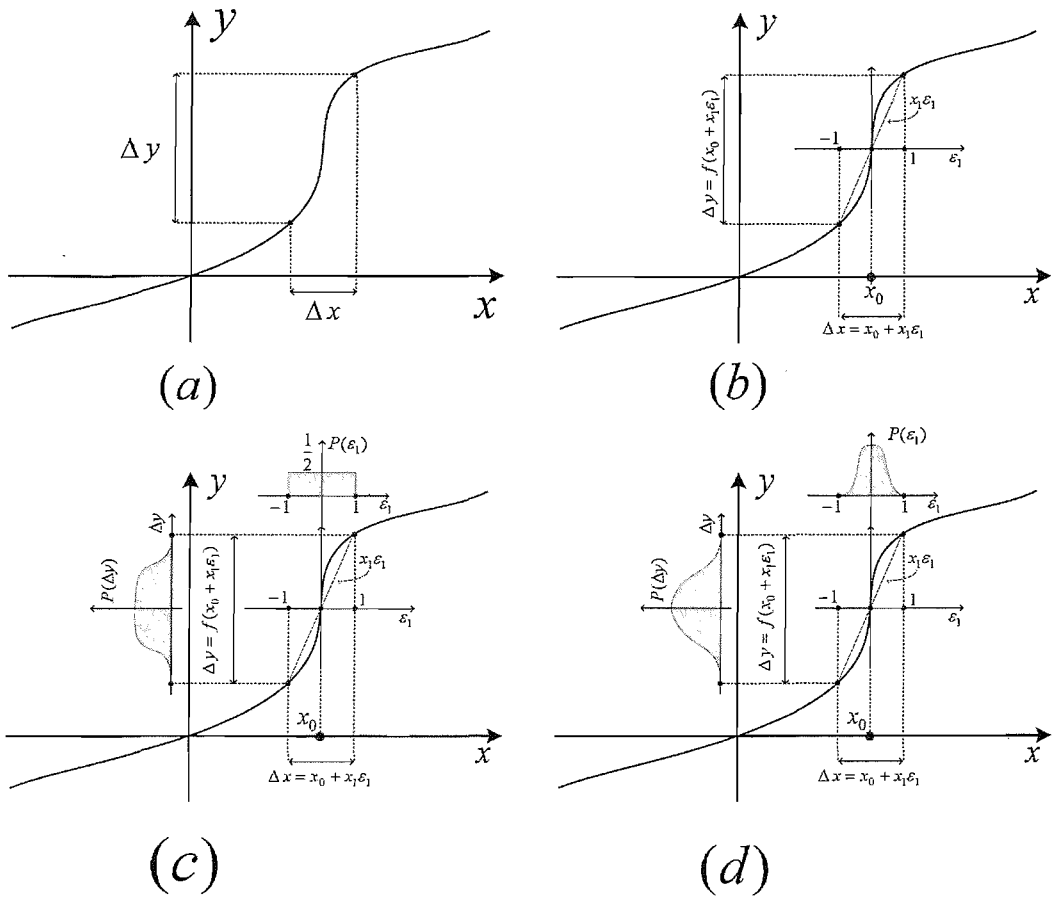


FIGURE 4.3: A visual view of symbolic noise representation of uncertainty a)IA method b)AA method c)SNA method assuming uniform PDF for noise symbols d)SNA noise symbols with arbitrary PDF

Berleant, [8], introduced a method to extend the automatically verified numerical inference to include combining operands when both are intervals, probability density functions or when one is an interval and the other a probability density function. This technique employs interval arithmetic techniques and forms a sharp contrast with traditional Monte Carlo methods. Our SNA method employs a similar method to calculate the output PDF of the function, which is discussed in section 4.4. Since the noise symbols are the basic elements of the method, an investigation of the computational noise PDF is required at the first place. The next subsection provides a brief discussion in this regard.

4.3.1 Statistical Model of The Noise Symbols

The probability density of the noise symbols are the initial information in the SNA method of error analysis. As discussed in chapter 3, the sources of these error symbols are different which means that they might have different characteristics. For instance, a very basic characteristic which is different for those are result of data or coefficients

quantisation and those are result of data refitting is the type of the distributions where quantisation noise has a *continuous probability distribution* and the data refitting noise is a *discrete random value* [54]. In most computational error analysis methods all kinds of noise are considered to be continuous and uniformly distributed.

In this study, all the errors are specified in the form of an algebraic combination of noise symbols, which are distributed over $[-1 + 1]$ with a known PDF. Since all the noise symbols are considered over a small range and they need to be combined or interact with each other, we consider all the noise symbols as continuous random values. However, input noise symbols are assumed to have an uniform distribution and it is also possible to extract the PDF of the noise symbols by simulation based on stimuli. In some cases where the input noise has another PDF other than uniform, the uniform distribution can easily be replaced by a simulation extracted PDF, which does not make any change to the overall algorithm. This option can be interpreted as a mixed method of dynamic and static analysis methods, see section 3.2, in which every subsystem can be used to model their noise symbols by simulation.

The other sources of noise symbols are arithmetic units and the data refitting process. In the case of data refitting noise, because almost all the function evaluation methods use basic arithmetic operations, an investigation of the noise models of them are required. Truncation of the result after addition and subtraction, provided no overflow happens, produces an error which can be modelled with a uniform PDF. Multiplication, however, behaves differently because of its sparse results distribution over the output range. The next subsection provides a brief investigation of the output truncation of fixed-point multiplication.

4.3.1.1 Multiplication Noise Model

The problem of the output PDF of multipliers must be considered in two separate categories of continuous and discrete distributions. Continuous distributions deal with real numbers in mathematics. Discrete distribution can be construed as integer numbers or fixed point representations in digital systems. Basically, the distribution of the product of two continuous random variables can be expressed as [54]:

$$\begin{aligned} F_{X\cdot Y}(a) &= P\{X \cdot Y \leq a\}, \\ &= \iint\limits_{X\cdot Y \leq a} f_{X\cdot Y}(x,y)dxdy, \\ &= \int_{-\infty}^{+\infty} f_{X,Y}(x, \frac{a}{x}) \cdot \frac{1}{|x|} \cdot dx, \end{aligned} \tag{4.15}$$

where X and Y are continuous random variables and $f_{X,Y}(x, y)$ is the probability density function of $(x \neq 0, y)$. Finding a closed form for this distribution function is not straightforward but some solutions have been addressed, [48], as depicted in Figure (4.4).

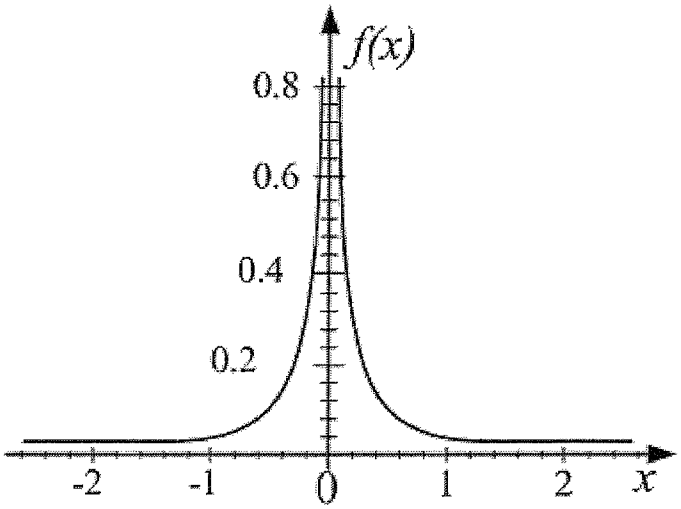


FIGURE 4.4: PDF of multiplier output with continuous normal distributed inputs. [48]

In the case of integer numbers this distribution is different because of the discrete characteristics of integer numbers. To see the similarity of the continuous distribution and discrete distribution for fixed-point numbers, a set of simulations over a large number of random numbers ($> 10^{15}$) are performed to determine the frequency of the output numbers of a 16-bit multipliers for inputs with uniform distribution. The output distribution is depicted in Figure (4.5).

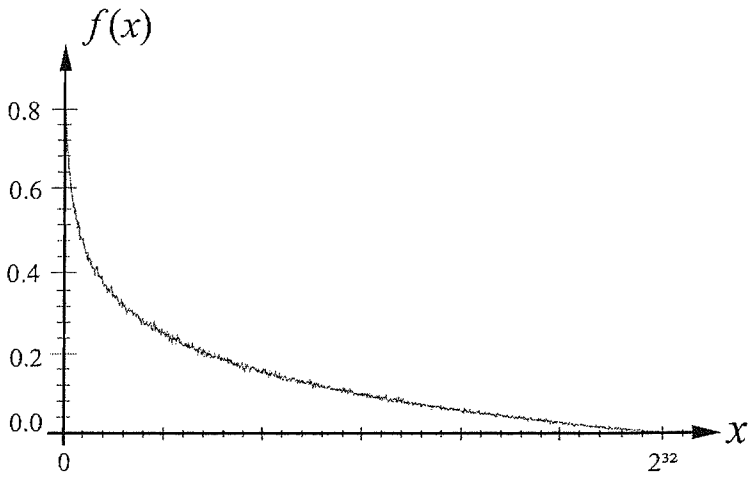


FIGURE 4.5: PDF of 16-bit multiplier output with integer uniform distributed inputs.

It is attainable form Figure (4.5) that multiplication results for integer numbers are more sparse in bigger numbers. This fact has been confirmed by our exhaustive search of the distinct multipliers output and it is observed that the output of multipliers does not cover the output bit-width range in full. The results of these investigations appear in

Table (4.1). In Table (4.1), the first column shows the multiplier bit-width, the second column shows number of distinct results, the third column indicates the total number of the numbers which can represented by output bit-width and the last column indicates what percentage of these possible numbers appear in the multiplier output. Clearly, by increasing the bit-width, the output will be more sparse.

TABLE 4.1: Multipliers output coverage for different word-lengths.

N-bit Multiplier	Number of distinct outputs	Total number of outputs	Output coverage %
1	2	4	50.00%
2	7	16	43.75%
3	26	64	40.63%
4	90	256	35.16%
5	340	1024	33.21%
6	1238	4096	30.23%
7	4647	16384	28.37%
8	17578	65536	26.83%
9	67592	262144	25.79%
10	259768	1048576	24.78%
11	1004348	4194304	23.95%
12	3902357	16777216	23.26%

Now, the question is how does this phenomenon affect the round-off noise in hardware implementations of the multipliers? A set of simulations with random numbers were performed which gives a bitwise view of the problem. Figure (4.6) gives the probability of each bit of the 16×16 bit multiplier being "1". According to this graph, this probability is not identical for all the bits of the output. Our empirical investigation of the Figure (4.6) shows that in m-bit truncation of the a multiplier with N-bit output, the probability of a "1" occurring in the the k^{th} place of the output noise is:

$$P_1(k) = \frac{2^k - 1}{2^{k+1}}, \tag{4.16}$$

and accordingly the probability of a "0" occurrence in the the k^{th} place of the output noise is:

$$P_0(k) = \frac{2^k + 1}{2^{k+1}}, \tag{4.17}$$

where in the both formulas $k \leq \frac{m}{2}$. Practical results confirm these equations. From this fact, a uniform PDF should not be expected from the output a multiplier.

Urabe in [131] presents a comprehensive discussion about the probability distribution of the truncation error of the fixed-point multiplication where he proves the following theorem:

Theorem 4.1. *In a fixed-point multiplication, divide the range of the roundoff error into 2^n equal intervals. Then the probability for the roundoff error falling into any one*

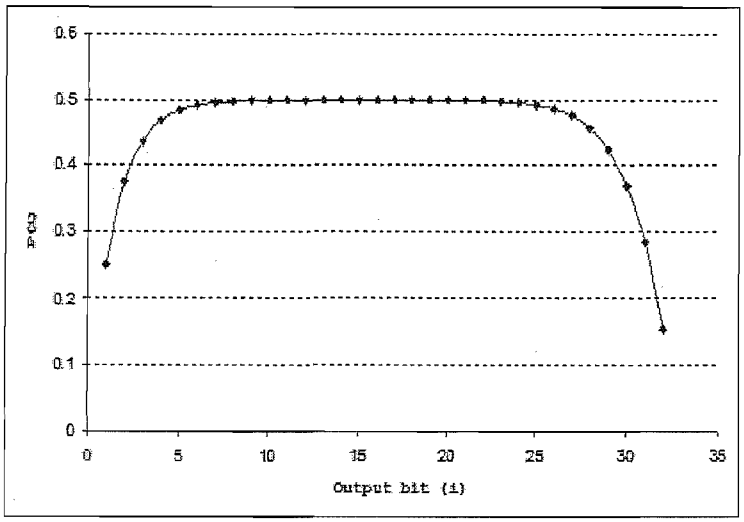


FIGURE 4.6: Probability of each bit being "1" in the output of a 32-bit multiplier.

of these intervals converges to 2^{-n} *as the number of digits of the factors is increased indefinitely.*

In our case, with an exhaustive simulation of the truncation output noise of the multipliers for random uniform distributed inputs, the PDF of the output noise for 8-bit truncation out of 16-bits is depicted in Figures (4.7). It can be observed that this distribution is not uniform.

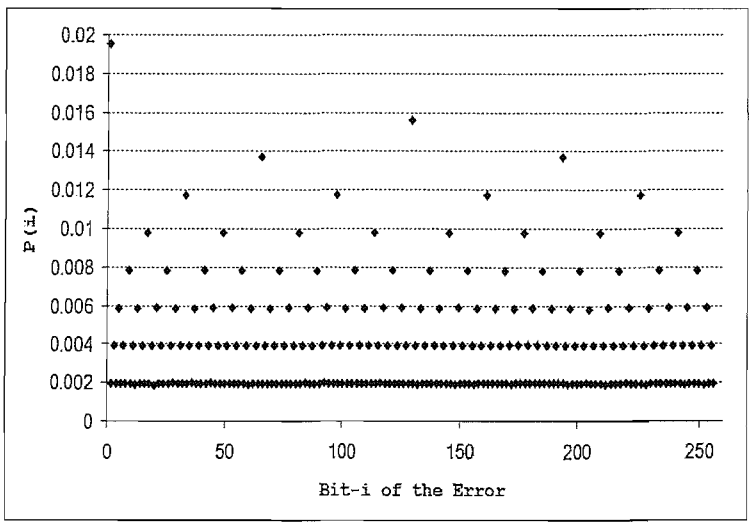


FIGURE 4.7: PDF of 8-bit truncation error for 16-bit multiplier output with uniform distributed input.

Applying Equations (4.16) and (4.17) and according to the symmetry of the graph, the probability function can be formulated as

$$\begin{aligned} P(x) &= \frac{P_1(k+1)}{2^{m-k}} \prod_{i=1}^{k} P_0(i), \\ &= \frac{2^{k+1}-1}{2^{m+2}} \prod_{i=1}^{k} \left(\frac{2^i+1}{2^{i+1}} \right), \\ &= \frac{2^{k+1}-1}{2^{m+2} \cdot 2^{\frac{k(k+3)}{2}}} \prod_{i=1}^{k} \left(1+2^i\right), \end{aligned} \tag{4.18}$$

where k is the biggest integer number in the range of $0 < k \leq m$ that 2^k divides x, which formally means:

$$k = Max \left\{ i \in \mathbb{N}, 0 \leq i \leq m, 2^i | x \right\},$$

where $\mathbb{N}$ represents the set of positive integer numbers. In another word, the probability of the occurring number x in the output depends on the place of the first "1", from the right hand side, in the base-2 representation of x. The mean value and variance of the truncation noise for different numbers of truncated bits are compared in Table (4.2) . According to the table , by increasing the number of truncated bits, uniform distribution model gets closer to the actual model. This result confirms the results in [5], where it is suggested that the roundoff noise of the fixed-point multiplication can be approximated by a uniform distribution for a large number of m, number of truncated bits, and N, word-length of the multiplier output.

TABLE 4.2: Mean value and variance of uniform PDF and our model in Figure (4.7)

m-bit	Uniform Distribution		Proposed Distribution	
Truncation	Mean	Variance	Mean	Variance
1	0.5	0.083333333	0.25	0.1875
2	1.5	0.75	0.999969	1.249939
3	3.5	4.083333333	2.750153	5.687454
4	7.5	18.75	6.501007	23.004912
5	15.5	80.08333333	14.252628	90.197059
6	31.5	330.75	30.005785	353.203323
7	63.5	1344.083333	61.769383	1392.149538
8	127.5	5418.75	125.486034	5518.875694
9	255.5	21760.08333	253.234952	21960.62686
10	511.5	87210.75	508.976108	87624.61602
11	1023.5	349184.0833	1020.686764	350021.5419
12	2047.5	1397418.75	2044.548389	1399220.573
13	4095.5	5591040.083	4092.004452	5594182.905
14	8191.5	22366890.75	8188.349327	22372861.38
15	16383.5	89473024.08	16379.13195	89482935.01

Truncation noise for multipliers with inputs with different PDFs is still matter of question. We believe that the truncation noise is the same for a big category of the input distributions because according to [122], quantisation noise for a wide range of number distributions can be approximated by a random uniform distributed input. So in the case of multiplication, as shown in Figure (4.8), the m-bit truncation error of the output is only dependent on the m-bits LSB of the inputs, and since m-bits of the inputs can be approximated by uniform distributed inputs, the multiplication truncation noise for a wide range of the inputs can be modelled as in Figure (4.7).

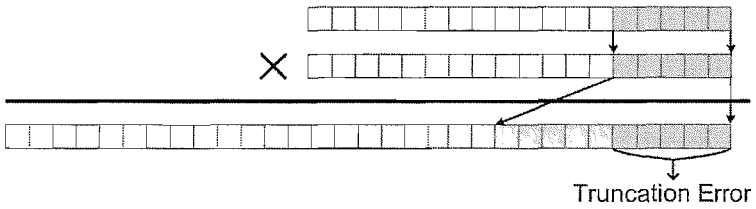


FIGURE 4.8: Truncation error depends only on the LSB part of the inputs.

It can be concluded that in word-length optimisation, PDF of truncation error in fixed-point multipliers are better to be assumed uniform where number of truncated bits, m, is greater than or equal to 8. For truncation with smaller number of bits ($m < 8$), the PDF s different which needs to be considered for more precise error analysis.

4.3.2 Noise Symbols Propagation

In the SNA method a partially known quantity $\hat{x}$, which is known analytically or has been extracted from empirical or simulation-based information, is initially represented in SNA form as in Equation(4.19).

$$\hat{x} = F_x(\vec{E}), \tag{4.19}$$

where $F_x(\cdot)$ is a fraction of polynomials with M known coefficients $(x_1, x_2, \cdots, x_M)$; and $\vec{E}$ is an array as in Equation (4.20).

$$\vec{E} = [\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_m], \tag{4.20}$$

where ε_i are symbolic representations of random values.

This algebraic representation [94], covers a big range of nonlinear relationships which can be expressed as algebraic relations. Equation (4.19) is an extension to the ordinary Taylor Model for representation of the uncertainties. Furthermore, the AA representation can be achieved with a first order Taylor Model:

$$x = x_0 + \sum_{i=1}^{m} x_i \cdot \varepsilon_i, \tag{4.21}$$

where x_0 is the original value, x is the rounded value, $x_i \in R$ are constants and $-1 \leq \varepsilon_i \leq +1$ are noise symbols in the range $[-1\ +1]$.

Every noise symbol has a known Source (S) in the computation data flow graph and a known probability density function (P), $\epsilon_i = (S, P)$. In the proposed method, modelling the effects of word-length manipulation takes place in two basic steps: the first is a noise model for computational errors for every operation in the DFG, and the second is a model of noise propagation through the DFG.

To evaluate the noise propagation through the DFG, a specification of the datapath polynomial algebra is applied and also all the non-algebraic operations are approximated by fraction functions. Consequently, since all the error models for linear and nonlinear operations are represented in the form of fractional (algebraic) functions, the output representation of the error will be a fractional function of the corresponding noise symbols, which means that y can be written in the form:

$$\hat{y} = F_y(X, \epsilon_{k1}, \epsilon_{k2}, \cdots, \epsilon_{kn}), \tag{4.22}$$

where $F_y(\cdot)$ is a fractional function and X is the input vector. Polynomial operations and methods are explained comprehensively in related works such as [72] and [94].

A relationship between output and noise symbols will be useful providing it is possible to derive an explicit specification for the output uncertainty. Consequently, the next problem regarding noise symbol propagation is noise symbol combination. Because noise symbols are random variables with a known PDF, to extract useful information, the algebraic operations on probability functions needs to be investigated in more detail.

$$F_y(X, \epsilon) = \frac{p(X, \varepsilon)}{q(X, \varepsilon)}, \tag{4.23}$$

where $p(\cdot)$ and $q(\cdot)$ are polynomials and X and ε are vectors of input and noise symbols respectively. The next section presents the method of finding PDF of the $F_y(X, \varepsilon)$ knowing PDF of noise symbols (ε).

4.4 Symbols Combination

To utilise symbolic noise analysis, one needs to combine the output noise PDF from Equation (4.23) and PDF of the noise symbols. In other words, we are given random numbers $\epsilon_1, \epsilon_2, \cdots, \epsilon_N$, such their the probability distribution from which are known and we are interested in determining the distribution of $F(X, \epsilon_1, \epsilon_2, \cdots, \epsilon_N)$. Finding a closed form for output PDF of the system in the general case is very attractive, however, it is well known that analytic solutions may only be derived for simple functions [109].

Basic examples can be found in distribution theory which employ characteristic functions as an auxiliary tool for PDF extraction of the functions of the random variables [117].

In probability theory, the characteristic function of any random variable completely defines its probability distribution, where it is denoted by $\phi(t)$ and is defined as the Fourier transform of the probability function given by Equation (4.24), where X is any random variable [117].

$$\phi_X(t) = E\left(e^{jtX}\right). \tag{4.24}$$

t is a real number, j is the imaginary unit, and E denotes the expected value. If f_X is the probability density function, then the characteristic function is

$$E\left(e^{jtX}\right) = \int_{-\infty}^{+\infty} e^{jtx} f_X(x)dx, \tag{4.25}$$

Characteristic functions are particularly useful for dealing with functions of independent random variables. For example, if $\{X_1, X_2, \cdots, X_N\}$ is a sequence of independent (and not necessarily identically distributed) random variables, and

$$S_N = \sum_{i=1}^{N} a_i X_i, \tag{4.26}$$

where the a_i are constants, then the characteristic function for S_N is given by:

$$\begin{aligned} \phi_{S_N}(t) &= E\left(e^{jt(a_1X_1+a_2X_2+\cdots+a_NX_N)}\right), \\ &= E\left(e^{jt(a_1X_1)}e^{jt(a_2X_2)}\cdots e^{jt(a_NX_N)}\right), \\ &= \phi_{X_1}(a_1t)\cdot\phi_{X_2}(a_2t)\cdots\phi_{X_n}(a_Nt). \end{aligned} \tag{4.27}$$

Clearly the independence of X_i are required to establish the equality. PDF of the S_N can be driven by the inverse Fourier transform of Equation (4.27).

Example 4.3. *Consider a set of random variables X_i which have a uniform distribution on the interval $[0,1]$ the distribution for the S_N can be found directly as in Equation (4.28) by a Fourier transform table [98].*

$$\begin{aligned} f_{S_N}(u) &= \mathcal{f}^{-1}\left[\left(j\frac{1-e^{jt}}{t}\right)^N\right](u), \\ &= \frac{1}{2(N-1)!}\sum_{k=0}^{N}(-1)^k\binom{N}{k}(u-k)^{N-1}\mathrm{sgn}(u-k), \end{aligned} \tag{4.28}$$

where the f^{-1} *represents the inverse Fourier transform and* $\mathrm{sgn}(\cdot)$ *is the sign function. The first few values of* f_{S_N} *are given as in Equation (4.29).*

$$
\begin{aligned}
f_{X_1}(u) &= \frac{1}{2}\Big[\mathrm{sgn}(1-u)+\mathrm{sgn}(u)\Big], \qquad (4.29)\\
f_{X_1+X_2}(u) &= \frac{1}{2}\Big[(-2+u)\mathrm{sgn}(-2+u)-2(-1+u)\mathrm{sgn}(-1+u)+u\mathrm{sgn}(u)\Big],\\
f_{X_1+X_2+X_3}(u) &= \frac{1}{4}\Big[-(-3+u)^2\mathrm{sgn}(-3+u)+3(-2+u)^2\mathrm{sgn}(-2+u),\\
&\quad -3(-1+u)^2\mathrm{sgn}(-1+u)+u^2\mathrm{sgn}(u)\Big],
\end{aligned}
$$

Since the analytical solution for the PDF extraction of the output in the closed form is not a trivial task and the general form is limited to simple cases, approximation methods are used in practice. Monte Carlo simulations are the best known and most commonly applied numerical methods for solving this kind of problem. According to [109], however, Monte Carlo simulation is often an unsatisfactory and misleading treatment of compounding uncertainty such as in our application.

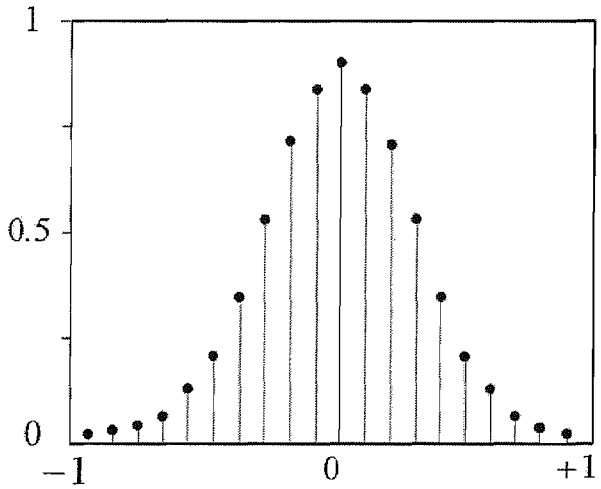


FIGURE 4.9: Sampled PDF represented in the form of grid.

A very basic method to approximate a PDF is discretising, where all the PDFs are approximated by sets of numbers which are samples of the real PDF, as depicted in Figure (4.9). In this way the PDF is reduced from a continuous distribution to a discrete function. The convolution operation between PDFs, therefore, can be performed by a finite number of elementary operations. The accuracy of the method depends on the sampling rate where a higher number of samples increases the accuracy and computation overhead. To compromise between accuracy of the method and computational overhead, modifications are possible. The method which we use in this study, therefore, is based on a method which combines range error analysis and probability propagation, called the histogram method, explained in the following subsection.

4.4.1 Histogram-Based Method

From a probability viewpoint, the interval methods implicitly have made the assumption that the modelled values are situated within the bounds of the specified intervals with a probability of "1". Accordingly, providing an operation, such as $\circ$, is applied to values $x \in \hat{x}$ and $y \in \hat{y}$, where $\hat{x}$ and $\hat{y}$ are intervals, to get a result $z = x \circ y$, it can be said that $z \in \hat{z} = \hat{x} \circ \hat{y}$, see section 3.3.1, where the probability $P(z \in \hat{z})$ conforms to:

$$P(z \in \hat{z}) = P(x \in \hat{x}) \cdot P(y \in \hat{y}) = 1 \times 1 = 1, \tag{4.30}$$

assuming $\hat{x}$ and $\hat{y}$ are independent. It is also assumed that all the numbers in the interval have the same probability. Interval representation of the uncertain values, therefore, implies that the uncertain value is a random number with a uniform distribution over the specified interval. This probabilistic view of the interval operations is the core idea of the proposed method in [8], called the *Histogram Method* for doing operations on probability density functions. Informally, a histogram is defined as a finite set of numbers representing histogram values (bins) in the corresponding non-overlapping subintervals. In other words, a histogram is the graphical version of a table which shows what proportion of cases fall into each of several or many specified subintervals.

The histogram method has extended the elementary arithmetic operations to include histograms as the basic data type. It employs the interval arithmetic method to do the operations on the bins of the histograms to create a new set of bins for the output histogram. The widths of the bins are the output intervals where the corresponding probabilities of the output intervals are assigned based on the probability rule of Equation (4.30). These values are the heights of the bins on the output histogram. From another standpoint, the arithmetic operations on histograms can be envisaged as a discretised convolution of the functions which have been approximated in the form of the histogram operations.

To approximate the density functions, the input range of the function is divided into a certain number of non-overlapping intervals. The standard interval data type is also extended with a probability mass distributed inside the interval to form a histogram bar. This probability is the same as the value of the function over the range. Providing that a proper number of subintervals are chosen, a common approximation to obtain an estimate of the output PDF is to assume that the distributions inside the bars are always uniform. The uniformity approximation allows interval analysis to be used for probabilistic function characterisation where the computation is typically performed in terms of standard interval arithmetic.

To operate on a pair of PDFs of $\hat{x}$ and $\hat{y}$, the output histogram can be calculated as follows [8]:

1. Split the PDFs of $\hat{x}$ and $\hat{y}$ into subintervals to create their corresponding histograms $H_x = \{\hat{x}_i\}$ and $H_y = \{\hat{y}_i\}$.

2. Compute the Cartesian product of the bins of the histograms describing $\hat{x}$ and $\hat{y}$.

3. For each combination $(\hat{x}_i, \hat{y}_j)$ in the Cartesian product, produce an intermediate result interval employing the corresponding IA operation over the ranges:

   (a) Execute the corresponding interval arithmetic operation on $\hat{x}_i$ and $\hat{y}_j$ to get $\hat{z}_{ij} = \hat{x}_i \circ \hat{y}_j$;

   (b) Associate with $\hat{z}_{ij}$ the probability $P(\hat{z}_{ij}) = P(\hat{x}_i) \cdot P(\hat{y}_j)$;

4. The intermediate result intervals must be combined to get the bins of the output histogram:

   (a) Decide on a set of intervals partitioning the domain of $\hat{z}$. This partitioning determines the placement of the bins in a histogram approximating the distribution function. The particular partition is unspecified by the algorithm, but few bins will tend to provide coarse results;

   (b) Calculate the area for each histogram bin of $\hat{z}$ defined by the partition as follows:

      i. Any intermediate result interval, $\hat{z}_{ij}$, that falls completely within one member of the partition has its entire probability mass assigned to the bin corresponding to that member;

      ii. Any intermediate result interval that overlaps more than one member of the partitions, has its probability mass divided between them, with mass assigned to each partition member in proportion to the fraction of the intermediate result interval it overlaps.

      iii. All of the probabilities assigned to a partition member are accumulated to give the total probability of the number. This is done for each partition member;

   (c) The probability of each bin equals its area, so the height of each bar is:

$$h = \frac{probability}{width}.$$

Since noise symbols in SNA method are assumed to be continuous random numbers, to employ the histogram method on them, their PDF needs to be segmented into subintervals to build up histograms. The number of these segments represents the accuracy of the histogram methods. Therefore, a better approximation for PDF is possible by increasing number of the segments, however, it costs more computation overhead. Each histogram bin is characterised both by an interval describing its placement on the real number line and by a probability density function which shows the placement probability inside the interval. In this way, the standard interval data type is extended with

a probability mass distributed inside the interval to form the height of histogram bins. A common approximation to obtain an estimate of the output PDF is to assume that the distributions inside the bars are always uniform. The uniformity approximation simplifies using interval analysis for probabilistic function characterisation where the computation is typically performed in terms of standard interval arithmetic.

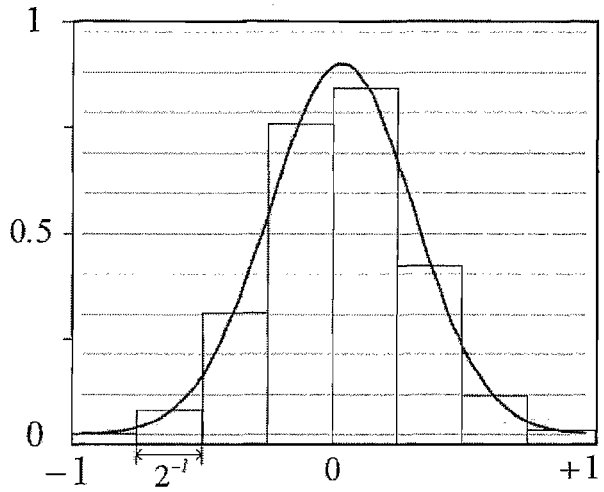


FIGURE 4.10: Histrogram approximation of a PDF.

Noise symbols in our method (ϵ_i) are bounded random values in a fixed predefined range of $[-1, 1]$. Accordingly, all the operations on them must also result in PDF over $[-1, 1]$ to be applicable in the succeeding operations as new symbols. Furthermore, these symbols have continuous PDFs over the range and we need to choose how to divide them into subintervals. It seems better, in terms of the computation overhead, to use the same standard of discritisation for all the PDFs. We divide all the noise symbols into 2^{l+1} subintervals, as depicted in Figure (4.10), for the sake of simplicity and compatibility. In this study, a histogram (H) can be defined formally as in Definition (4.2), where corresponding to each interval I_i in the H, a probability p_i is defined which represents the PDF value in the interval I_i.

Definition 4.2. A histogram H is a partition of the domain ϵ in terms of intervals I_i and local probabilities p_i where:

$$H = \left\{ (I_i, p_i) \middle| I_i = \left[-1 + i \cdot 2^{-l}, -1 + (i+1) \cdot 2^{-l} \right), \; i = 0, \cdots, 2^{l+1} - 1, \; 0 \leq p_i \leq 1 \right\}.$$

where l in the Definition (4.2) represents the granularity of the histogram H. This definition also holds the basic characteristics of the set partitioning as:

$$H = \left\{ I_i \middle| (\bigcup_{\forall i} I_i = \epsilon) \bigwedge (\bigcap_{\forall i} I_i = \emptyset) \right\},$$

and also the basic probability rule ($P(x) \leq 1$), which implies that:

$$\sum_{i=0}^{2^{l+1}-1} p_i = 1,$$

With this definition, the histogram method splits the noise symbols and their corresponding PDF into fixed length subintervals, as depicted in Figure (4.10), and uses these subintervals to generate intermediate results and then constructs the result PDF of the arithmetic combination of the symbols .

Assuming interval operands are represented according to H, the new operators return the result of the operation also represented in terms of H. When this result includes more than one interval, the operator distributes the probability, p_i, among the output intervals depending on the behaviour of the specific arithmetic operation. In order to have consistent input and output data types, this new arithmetic can be formulated as histogram arithmetic, as inputs can be viewed as histograms having a single interval.

Using our definition the computation model of the previous section is modified as:

1. Consider the input space is the set of intervals describing the histogram of the input i in terms of $H_i = \{(I_{ij}, p_{ij})\}$, where interval I_{ij} of H_i can be represented as: $I_{ij} = [a_{ij}, b_{ij}]$.
2. For each vector $[\cdots, ([a_{ij}, b_{ij}], p_{ij}), \cdots]$ of the input space:
   (a) Compute the probability $p_k = \prod_{i=1}^{I} p_{ij}$;
   (b) For each operation with input histograms H_i (with one or several intervals), and for each combination of intervals from the Cartesian product of the intervals of histograms H_i do:
      i. Obtain a histogram result using histogram-based arithmetic;
      ii. Proceed with the next operation if there is any more (step c) otherwise return;
      iii. Set the probability of each resulting histogram interval I_k by its calculated p_k;
      iv. Collect the result histogram to produce the output histogram (H_{out});

According to the algorithm, all algebraic operations on histograms can be expanded to the interval operations. Consider a set histograms $H_i = \{(I_{i,j}, p_{i,j})\}$ and function F which is applied to them to produce another histogram as in Equation (4.31).

$$H_{out} = F(H_1, H_2, \cdots, H_m) = F\left(\{(I_{1,j}, p_{1,j})\}, \{(I_{2,j}, p_{2,j})\} \cdots, \{(I_{m,j}, p_{m,j})\}\right), \quad (4.31)$$

where $H_{out} = \{(I_{out,k}, p_{out,k})\}$ is the result histogram. Applying the algorithm means that F must be applied to all the Cartesian combinations of the input histograms as in

Equation (4.32).

$$\{(I_{out,k}, p_{out,k})\} = \{F((I_{i,j}, p_{i,j}))\}. \tag{4.32}$$

All the operations between intervals of the histograms are exactly the same as in the IA method in section 3.3.1. These operations on the input intervals produce a set of overlapping intermediate intervals with the corresponding probability values which need to be mapped into the output histogram bins. Recalling the fact that all the histograms must have the same structure of Definition (4.2), let assume that intermediate intervals are:

$$(I_{out,k}, p_{out,k}) = F((I_{i1,j1}, p_{i1,j1}), (I_{i2,j2}, p_{i2,j2})) = \left[-1 + x2^{-l}, -1 + y2^{-l}\right], \tag{4.33}$$

where x and y indicate two ends of the intermediate interval which are calculated by applying F to the intervals $(I_{i1,j1}, p_{i1,j1})$ and $(I_{i2,j2}, p_{i2,j2})$. The place of the intermediate interval and its overlapping with the final intervals $\{(I_{out,k}, p_{out,k})\}$ can be extracted by comparing x and y with k. If the intermediate interval has a share in the output interval $I_{out,k}$, then the value of this share depends on the relative positions of x and y on k axis and also the probabilities multiplication $p_{i1,j1} \times p_{i2,j2}$. This positioning can be categorised as follow:

$$\begin{aligned}
x < k \quad &\rightarrow \quad \begin{cases} y < k & p_{out,k} = p_{out,k} \quad \text{(No overlap)} \\ k \le y < k+1 & p_{out,k} = p_{out,k} + (p_{i1,j1} \times p_{i2,j2}) \left(\frac{y-k}{y-x}\right) \\ k+1 \le y & p_{out,k} = p_{out,k} + (p_{i1,j1} \times p_{i2,j2}) \left(\frac{1}{y-x}\right) \end{cases} \\
k \le x < k+1 \quad &\rightarrow \quad \begin{cases} y < k+1 & p_{out,k} = p_{out,k} + (p_{i1,j1} \times p_{i2,j2}) \\ k+1 \le y & p_{out,k} = p_{out,k} + (p_{i1,j1} \times p_{i2,j2}) \left(\frac{k+1-x}{y-x}\right) \end{cases} \\
k+1 \le x \quad &\rightarrow \quad p_{out,k} = p_{out,k} \quad \text{(No overlap)}
\end{aligned} \tag{4.34}$$

x and y can be found for basic arithmetic operations to implement the algorithm.

In fact, the interval methods have been described as having more advantages than traditional random sampling approaches (i.e. Monte Carlo simulation) [8, 18]. Exhaustive exploration of the input data space is possible when represented in terms of intervals but is infeasible when considering individual numeric values. On the other hand, two problems appear when applying the histogram method to PDF estimation [18]. First, the computation of the Cartesian product of the input histogram bins yields a set of output bins that must be merged into a single output histogram and the complexity of this merging can increase to infeasible levels. For instance, merging two intervals with a non-empty intersection, as depicted in Figure (4.11), produces three smaller intervals, so every new merge is bound to deal with more intersections as the computation progresses. Second, it has been argued, [18], that assuming uniform distributions inside the histogram bins can be a problem with some operations that significantly increase the

size of the output interval but that causses sparse distributions, integer multiplication for example.

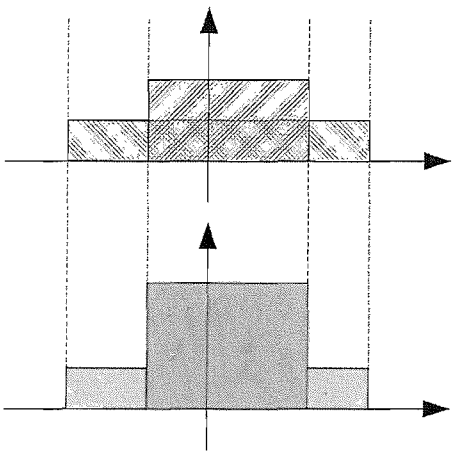


FIGURE 4.11: Merging to overlapping bins in the output histogram.

There are similarities between the method which we have employed in this study and a modified approach based on the definition of grids is proposed by Carreras, [18], to minimise these problems and better control the accuracy of the enclosures in the output histogram obtained through interval calculations. Histogram grids by forcing a specific representation on input and output histograms, and grouping the PDF samples in them, allow control of the sizes of their bars. However, the computation is typically still performed in terms of standard interval arithmetic. Such grids are analytic partitions of the numeric ranges of interest that force a specific representation, and are the basis for the definitions of new abstract domains and abstraction functions. Furthermore, it is possible to associate a probability with each element of the abstract domain associated with a given grid, resulting in the notion of histogram grids and the definitions of interval operations on such grids. This allows the application of this class of abstractions to the probabilistic characterisation of functions.

4.4.1.1 Mean and Variance of the Combined Variables

In the noise analysis approach, the noise spectrum is the key factor for accuracy comparison between different designs. After noise symbols propagation through the computation tree of the systems, the output error can be derived in the form of the algebraic combination of the noise symbols. Having known the statistical distribution of the noise symbols the noise spectrum can also be calculated in the output as well as the error ranges. In almost all the previous works, truncation or rounding errors are considered to be independent and uniformly distributed. Based on these assumptions and using stationary random signals characteristics in the time invariant system the output noise spectrum is calculated by p-norm as illustrated in Equation (3.34) [99].

In the simple cases of linear systems where output noise can be represented in terms of a linear combination of the noise symbols, the expectation and variance of the sum of independent random variables, such as ϵ_i in Equation (4.21), can be calculated by Equation (4.35) and Equation (4.36).

$$E\left(\sum_{i=1}^{m} x_i \cdot \epsilon_i\right) = \sum_{i=1}^{m} \left(x_i \cdot E(\epsilon_i)\right), \tag{4.35}$$

$$Var\left(\sum_{i=1}^{m} x_i \cdot \epsilon_i\right) = \sum_{i=1}^{m} \left(x_i^2 \cdot Var(\epsilon_i)\right) + 2\sum\sum_{i<j} \left(x_i \cdot x_j \cdot Cov(\epsilon_i, \epsilon_j)\right), \tag{4.36}$$

where $E(\cdot)$, $Var(\cdot)$ and $Cov(\cdot)$ stand for expectation, variance and covariance respectively. In addition, based on the Central Limit Theorem, the symbolic noises in Equation (4.21) be merged so that the distribution of the replacement symbolic noise is approximately normal for large m.

There are works which try to combine the abilities of statistical methods with symbolic analysis. [42] gives this basic idea that uncertainty symbols can be combined by the Central Limit Theorem [54] to reduce the symbol array size. This idea is expanded in [41] and applied to interconnection approximation by [77]. This method is improved to provide a method for statistical interval analysis by [121] and is utilised for VLSI and DSP design tools, however, their proposed method is only applicable to the simple cases of PDF combinations.

From a probability theory point of view, if the probability distribution of x admits a probability density function $f(x)$, then the expected value can be computed as:

$$E(x) = \int_{-\infty}^{+\infty} x f(x) dx. \tag{4.37}$$

The expected value of an arbitrary function of x, $g(x)$, with respect to the probability density function $f(x)$ is given by:

$$E(g(x)) = \int_{-\infty}^{+\infty} g(x) f(x) dx. \tag{4.38}$$

If $\mu = E(x)$ is the expected value mean of the random variable x, then the variance is:

$$Var(x) = E\left((x-\mu)^2\right) = E(x^2) - (E(x))^2. \tag{4.39}$$

For a histogram representation of the noise symbols this formula can be rewritten as:

$$\sigma_\epsilon^2 = Var(\epsilon) = \sum_{k=0}^{2^l-1}\left[\int_{1-k2^{-l}}^{1-(k+1)2^{-l}} p_k x^2 dx - \left(\int_{1-k2^{-l}}^{1-(k+1)2^{-l}} p_k x dx\right)^2\right], \tag{4.40}$$

$$= \sum_{k=0}^{2^l-1}\left[\frac{p_k^3}{3}\left((1-k2^{-l})^3-(1-(k+1)2^{-l})^3\right)-\frac{p_k^2}{2}\left((1-k2^{-l})^2-(1-(k+1)2^{-l})^2\right)^2\right],$$

$$= 2^{-l}\sum_{k=0}^{2^l-1} p_k^3 - 2^{-2l}\sum_{k=0}^{2^l-1}\left(p_k^2\left((2k+1)p_k+2\right)\right)$$

$$+2^{-3l}\sum_{k=0}^{2^l-1}\left(p_k^2\left((k^2+\frac{1}{3})p_k+4k-2\right)\right)+2^{-4l}\sum_{k=0}^{2^l-1}\left(p_k^2\left(2k-\frac{1}{2}\right)\right)$$

This computation may be combined with the histogram calculation in the system analysis and optimisation algorithm.

4.5 Implementation Algorithm

The overall algorithm for implementation of the symbolic noise analysis method consists of three basic major steps:

1. Build up a Noise Symbol representation for signals in the computation tree and their relationship with the arithmetic characteristics of the nodes.
2. Find the symbol propagation through the tree and its relationship in the output node(s).
3. Find the histogram representation of the output PDF and the corresponding bounds and noise powers.

Obviously each step in this algorithms contains many sub-steps.

The first step of the algorithm refers to the fact that based on the previous discussions in chapter 2 and 3, the errors in each computational node are a function of the arithmetic characteristics of the node such as word-length, arithmetic system and so on. Accordingly, the noise symbols assigned to each node are dependent on these characteristics. Noise symbols are created in a data structure which contains their source and probability distribution types (uniform, triangle ...) in the form of histograms with suitable granularity. The volume of this data base increases with the number of nodes in the system data flow graph. For computationally intensive systems which mostly require a repeated routine this volume should not be intractably big. Furthermore, it is possible to merge some of the noise together into new noise symbols to reduce the number, however

this approach causes inaccuracy and overhead in the optimisation process. To improve this merging method one might investigate the DFG of the system to find those noise symbols which only propagate through single paths and their merging does not decrease the optimisation accuracy.

The second step consists of polynomial operations to build up the output error relationship with the noise symbols sources from different points in the computation tree of the datapath.

The third step is based on the algorithm discussed in section 4.4.1, by which the output PDF and the corresponding bounds can be calculated. Applying the histogram requires choosing a proper granularity of the histograms discritisation which has a great impact on the result precision and also the algorithm complexity. For a histogram calculation of an G-point sampling of the PDF (granularity=G), the total number of the Cartesian combination of the grid points between M distinct noise symbols is G^M. For example, computational complexity of SNA method to evaluate the output noise PDF of a design with M functional units is $O(G^N)$, assuming G is the same for all noise points. With $G = 1$, evaluation time is the same as AA method and it increases with G polynomially (order M).

In comparison with the other methods, the proposed method has much more comprehensive information about the output error, however it has more computation overheads. Furthermore, in respect of AA error symbols, the noise symbols in the proposed method carry more information which means that their storing and manipulation requires a more complicated data structure and also more computation resources. However, the complexity of the method gives the freedom to the designer to adapt the required precision with available computation resources in an optimisation procedure. The following examples provides some results to compare the different methods.

Example 4.4. *Consider the quadratic equation of Example (4.1). Apply the SNA method results in the histograms of Figure (4.12), which is depicted over the error range* $[x_l, x_h]$. *Different histograms are represented which are calculated with different granularities. Table (4.3) also presents the mean , variance, lower bound and upper bound for these calculations. According to Equation (4.8), the output range can be calculated as*

$$\begin{aligned} y &= y_0 + \Delta y, \\ &= 6.5 + [x_l, x_h], \end{aligned} \tag{4.41}$$

where x_l *and* x_h *can be found in Table (4.3). In comparison, our method can provide better approximations and also the noise PDF at the output.*

This example shows that the higher granularity produces higher precision results but with more calculation overheads. This flexibility is especially useful in the optimisation process, where the low granularity calculations can be used for preliminary analysis to

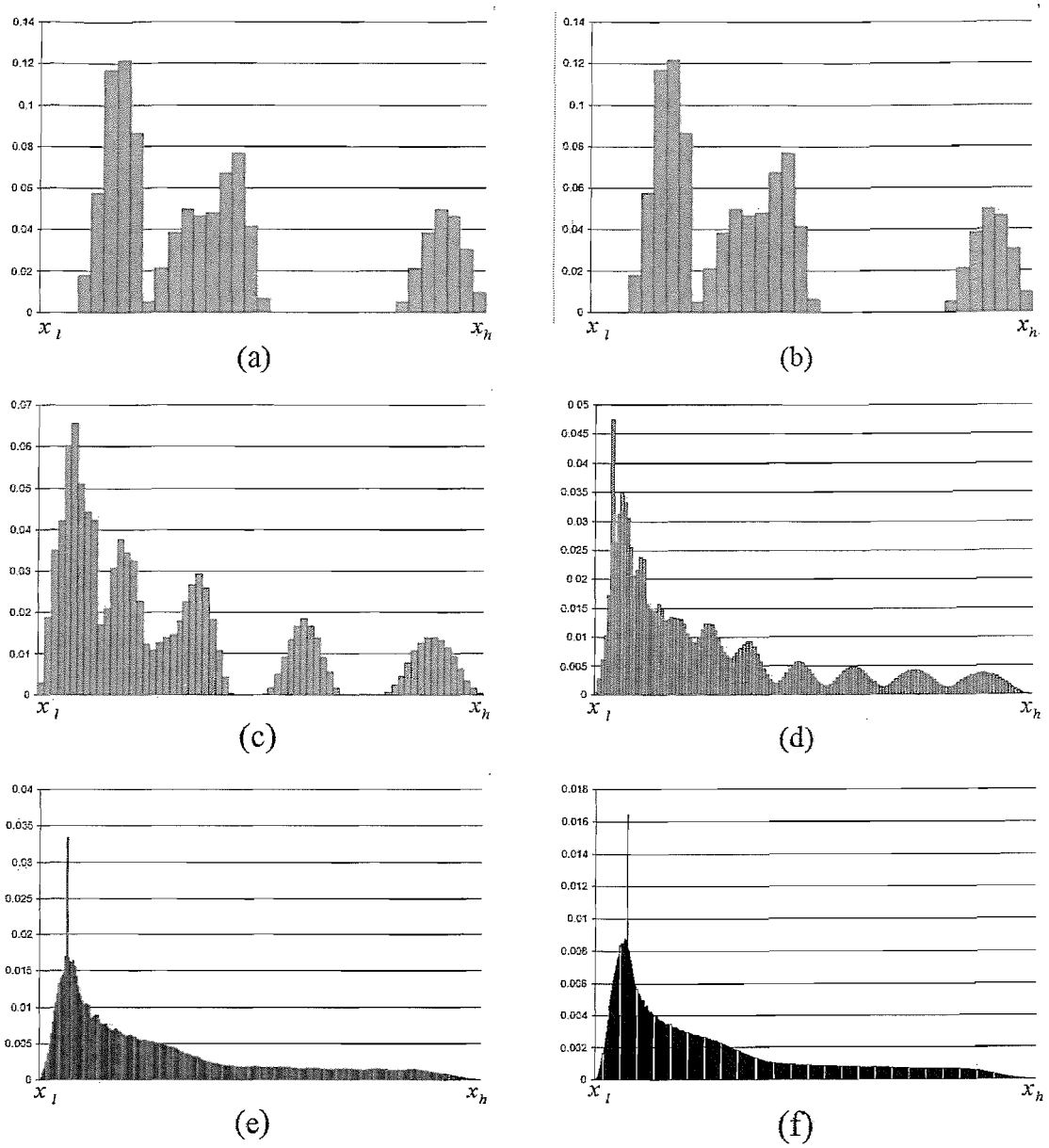


FIGURE 4.12: Output histogram over error range $[x_l, x_h]$ for Example (4.4) with different granularities a)g=2 b)g=4 c)g=8 d)g=16 e)32 f)64.

limit the feasible space of the optimisation search. Applying SNA method, execution time for this example was lower than one second for granularity of $g < 20$ and 4 second for $g = 32$ and approximately 50 seconds for $g = 64$ on a AMD Opteron 2GHz machine.

Example 4.5. *Let us try 32-bit fixed-point implementation of a polynomial approximation of the exponential function, as written in Equation (4.42) [47].*

$$\begin{aligned} y &= \big(a_1x^5 + a_2x^4 + a_3x^3 + a_4x^2 + a_5x + a_6\big)x^3 + 1.0, \\ &= (((((a_1x + a_2)x + a_3)x + a_4)x + a_5)x + a_6)x^3 + 1.0, \end{aligned} \tag{4.42}$$

TABLE 4.3: Estimated parameters with histogram method for Example (4.1)

Granularity	Mean	Variance	x_l	x_h	x_c
2	6.148148	36.274348	0.0	16.0	0.0
4	4.6688	28.173507	-1.0	16.5	-0.5
8	3.908208	22.467426	-1.25	16.5	-0.75
16	3.534689	19.489651	-1.375	16.5	-0.875
32	3.349956	18.014349	-1.4375	16.5	-0.9375
64	3.258123	17.286116	-1.46875	16.5	-0.96875
Actual Values	3.17	16.57	-1.5	16.5	-0.81

Table (4.4) shows the coefficients before and after quantisation and their quantisation noises. It is also assumed that the input value is limited to a range of $-0.5\ln 2 \leq x \leq 0.5\ln 2$ and is not carrying any error from previous computations. Error analysis of this equation is based on different methods for a fixed-point hardware with 32-bit maximum word-length. To focus on the accuracy effect, it is assumed that overflow does not happen in the computations because of the proper scaling in the input data, however, it can be analysed by SNA if we know the overflow PDF model of the arithmetic operations.

TABLE 4.4: Quantisation error of the coefficients in Example (4.5).

a_i	Coefficient Exact values(Dec)	Quantised Coefficient a_{i0} (Hex)	Error Weight a_{i1}	Noise Symbol
a_1	1.9875691500	$1.FCD154F1$	$9.17956E-11$	ϵ_1
a_2	1.3981999507	$1.65F06E95$	$6.11544E-11$	ϵ_2
a_3	8.3334519073	$8.555D1AAC$	$7.47763E-11$	ϵ_3
a_4	4.1665795894	$4.2AA4F5C0$	$7.61482E-11$	ϵ_4
a_5	1.6666665459	$1.AAAAA8A3$	$1.13829E-10$	ϵ_5
a_6	5.0000001201	5.00000203	$9.61093E-11$	ϵ_6

After coefficient replacement, the total error can be represented in the form of the Equation (4.43).

$$\begin{aligned}\Delta y &= (a_{11}x^8)\epsilon_1 + (a_{21}x^7)\epsilon_2 + (a_{31}x^6)\epsilon_3 + (a_{41}x^5)\epsilon_4 + (a_{51}x^4)\epsilon_5 + (a_{61}x^3)\epsilon_6, \\ &= (((((a_{11}\epsilon_1 x + a_{21}\epsilon_2)x + a_{31}\epsilon_3)x + a_{41}\epsilon_4)x + a_{51}\epsilon_5)x + a_{61}\epsilon_6)x^3, \end{aligned} \quad (4.43)$$

Interval Arithmetic

In this approach all the numbers are represented in the form of range vales as:

$$\hat{a}_i = [a_{i0} - a_{i1}\epsilon_{ai}\ ,\ a_{i0} + a_{i1}\epsilon_{ai}], \quad (4.44)$$

and as a consequence of applying IA rules, the output error range is:

$$\Delta y \in \left[-6.21628550436 \times 10^{-12} \ , \ +6.21628550436 \times 10^{-12}\right], \tag{4.45}$$

Affine Arithmetic

Affine arithmetic approach uses exactly the same terminology as shown in the problem and the final calculation is in the form of Equation (4.46).

$$\begin{aligned} \hat{y} &= y_0 + \Delta y, \\ &= y_0 + (a_{11}x^8)\epsilon_1 + (a_{21}x^7)\epsilon_2 + (a_{31}x^6)\epsilon_3 + (a_{41}x^5)\epsilon_4 + (a_{51}x^4)\epsilon_5 + (a_{61}x^3)\epsilon_6, \end{aligned} \tag{4.46}$$

and the error part of the output can be represented in an affine format as:

$$\Delta y = 0 + \left(6.21628550436 \times 10^{-12}\right)\epsilon_y, \tag{4.47}$$

where $\epsilon_Y \in [-1, +1]$ is the noise symbol of the output error.

Noise Based Analysis

In noise-based method these errors are considered as independent WWS noise sources which are propagating through the system. Since this example represents a linear system relationship between noise sources as in Equation (4.48) the output error variance can be calculated applying superposition.

$$y = \left(a_1x^5 + a_2x^4 + a_3x^3 + a_4x^2 + a_5x + a_6\right)x^3 + 1.0, \tag{4.48}$$

and the error variance at the output point is:

$$\sigma_{\Delta y} \approx 2.258 \times 10^{-22}, \tag{4.49}$$

Proposed Method

In the proposed method, values are represent in the form of a (not necessarily linear) combination of the quantised values and error symbols. The final representation is

$$\hat{y} = y_0 + (a_{11}x^8)\epsilon_1 + (a_{21}x^7)\epsilon_2 + (a_{31}x^6)\epsilon_3 + (a_{41}x^5)\epsilon_4 + (a_{51}x^4)\epsilon_5 + (a_{61}x^3)\epsilon_6. \tag{4.50}$$

Similar to last example, using an unform distribution for ϵ_i over $[-1, +1]$ and applying histogram calculation algorithm of section 4.4.1, the output error and its PDF can be found. PDFs of ϵ_i are divided to g (granularity) subsections then IA arithmetic is applied to calculate value Equation (4.50) over each Cartesian combination of the subsections. The histogram PDF of the output error is depicted in Figure (4.13). It can be observed

from this histogram that the output error is uniform only in the $\frac{1}{3}$ of the error range. This result is also compatible with the theoretically calculation of the weighted sum of uniform random variables as discussed in [59].

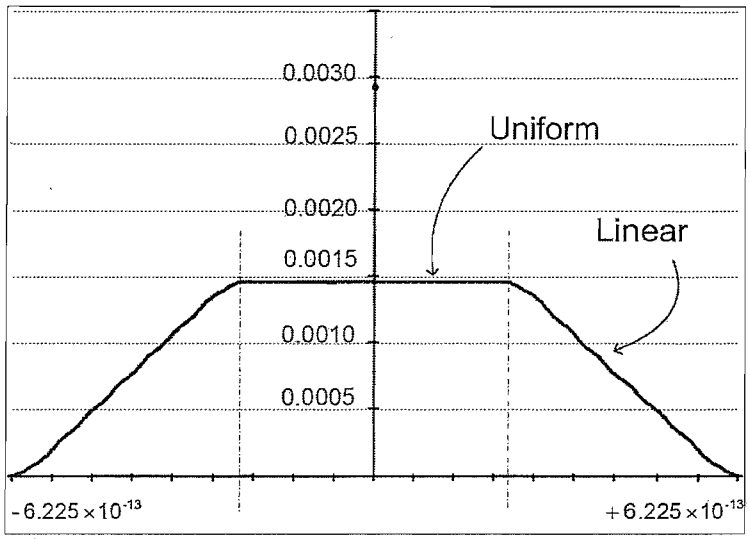


FIGURE 4.13: PDF histogram of the output error over the error range for Example (4.5).

Comparing results

Applying different methods to the computation the output error in each case is as in Table (4.5). It is observable from the table that SNA method provides more comprehensive information about output error compared with the other methods. IA and AA only give the output range of the result, when noise analysis method can only provide noise variance in the output. Although estimated range of the error is more accurate by SNA method its calculation overhead is higher. With SNA method, calculation time for this example was 1 second for $g = 8$ and was approximately 60 seconds for $g = 16$ on a AMD Opteron 2GHz machine.

TABLE 4.5: Error result in the output of the Example (4.5).

Analysis Method	Error Bound	Noise Power	PDF
IA Method	$[-6.216 \times 10^{-12}, +6.216 \times 10^{-12}]$	×	×
AA Method	$[-6.216 \times 10^{-12}, +6.216 \times 10^{-12}]$	×	×
NA Method	×	2.258×10^{-22}	×
SNA Method	$[-6.225 \times 10^{-13}, +6.225 \times 10^{-13}]$	$6.291 \times 10{-}24$	✓

4.6 Summary

In this chapter, a new method for modelling computational error is presented. There are works in which the computational error is modelled as an independent random noise source which behaves like white noise. There are other approaches which deal with the error as a solid symbol which represents the error range. Our model is a combination of both, where every error bound is represented by symbolic noise which consist of error bounds as well as the statistical characteristics. Using this method enables the analysis to be done, based on both methods. In addition the noise spectra at the output can be evaluated with controllable accurately to see the error distribution over the bounds. This method is applied to several examples and results show its power in practice.

the next chapter introduces the data communication structure for a hardware platform which is suitable for computational datapath implementation when the arithmetic characteristics of the functional units can be taken into account as an optimisation parameter.

Chapter 5

Multi-Way Multiple-Width

5.1 Introduction

From a system level point of view, every digital circuit consists of three basic parts: Functional Units (FU), controllers and interconnections. The first provides computational resources for the algorithm execution, the second part controls the sequence of the operations in the circuit and the third provides the communication channels for internal and external data and control signal exchanges. Technically, interconnection refers to all the wires in the design, whereas here we only focus on those which are responsible for data communication between units. Thus the control signals and inter-units wires are considered as a part of the controller and functional units respectively. Since computationally intensive hardware is supposed to deal with a large amount of data processing and exchanges, the data communication can be a bottleneck in many cases.

Providing an optimal communication channel for the functional units, which creates the required data communication paths with lowest implementation cost is a vital issue in today's design framework. Although this issue is mostly considered at the architecture level of design, in designs in which the datapath is dominant, it is necessary to consider this problem at a lower level of specification such as Register Transfer Level (RTL). Two main approaches have been used in communication synthesis at RTL, namely the peer to peer method and the bus-oriented method. This chapter describes a modified bus-oriented method and the corresponding implementation.

5.2 Motivating Examples

Bus-oriented datapath synthesis relies on sharing communication paths to reduce the cost of implementation. In return, due to shared data transfer channels, delays are likely during data exchange between functional units. To evaluate the viability of the bus-oriented method and compare its implementation capabilities with multiplexer-based methods, some basic examples are presented. The simplest structure of the bus-oriented method, namely "single shared bus", is compared with a peer-to-peer structure. These examples compare the cost efficiency of the methods in terms of latency and the number of required switches for steering logic.

Example 5.1. *Let consider an example of a behavioural specification of a first order differential equation solver for an architectural design as presented in [103]. The specified circuit has been designed to solve the second order differential equation of Equation (5.1).*

$$\frac{d^2y}{dx^2} + 3x\frac{dy}{dx} + 3y = 0 : \tag{5.1}$$

To solve this equation, [103], a forward Euler method is applied in the interval $[0, a]$ *with step size of* dx*. The corresponding algorithm and DFG are quoted in Listing (4.1) and Figure (5.1).*

Listing 4.1: Solution algorithm for Example (5.1)

Set Initial Conditions: y, $\frac{dy}{dx}$
while $(x < a_3)$ *repeat:*
  $x_f = x + dx;$
  $u_f = u - (a_1 \times x \times u \times dx) - (a_2 \times y \times dx);$
  $y_f = y + (u \times dx);$
  $x = x_f;\ \ u = u_f;\ \ y = y_f;$
end ;

To clarify the position of the data transfer units, consider a structural view of the circuit as depicted in Figure (5.2) in which FU stands for functional units. In this structure, MUL represents a multiplier, ALU is an arithmetic and logic unit, memory represents all memories (volatile and non volatile) and steering represents the data transfer logic and buses. Our discussion is focused on the steering block, which provides data communication media for functional units, and can be implemented with a multiplexer and/or bus based method.

Considering an ideal allocation and binding with two multipliers and ALUs, the data communication between these functional units will be as depicted in Figure (5.3). It can

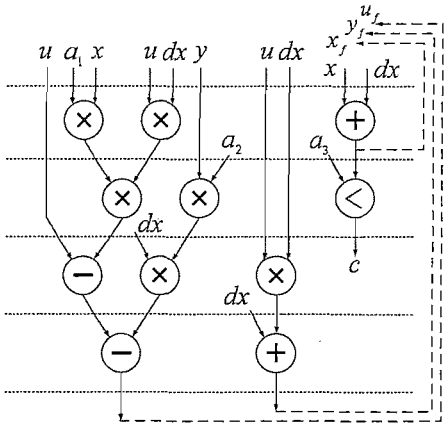


FIGURE 5.1: Corresponding DFG of the Example (5.1).

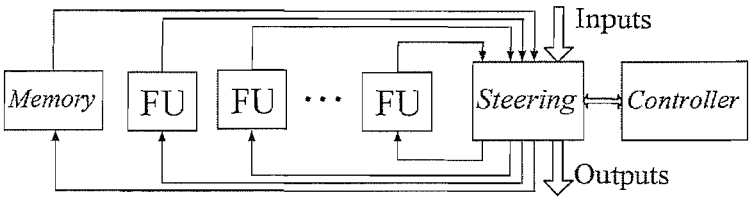


FIGURE 5.2: Structural view of the Example (5.1).

be seen that the implementation requires three 2-1 multiplexer (MUX-2) and one 3-1 multiplexer (MUX-3). In addition, with a smaller number of functional units (MULs or ALUs), the number of required multiplexers will increase. In comparison with the multiplexer-based method, a shared bus construction of the circuit requires only six bus switches as in Figure (5.4).

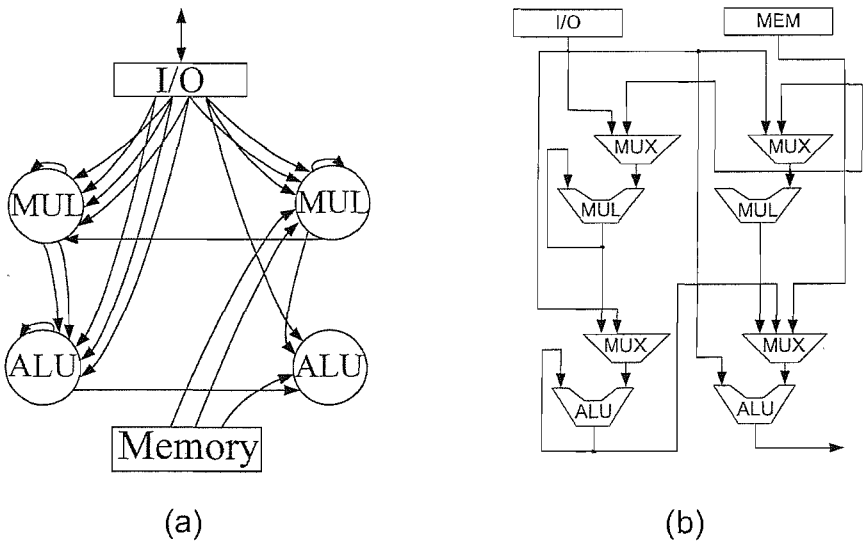


FIGURE 5.3: Data communication scheme after binding and allocation of the Example (5.1) a)data communication scheme b)multiplexer-based implementation.

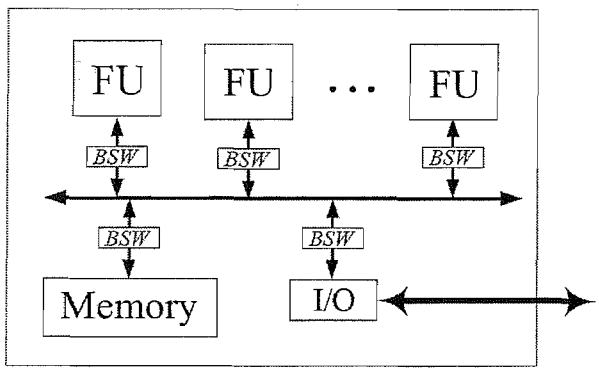


FIGURE 5.4: A shared bus oriented implementation of the Example (5.1).

Similarly, for a different number of resources (MUL and ALU), after scheduling and allocation by hand, the number of the required multiplexers is shown in Table (5.1). In this table T_{MUL}, T_{ALU} and T_{DT} stand for the delays of multiplication, ALU operation and Data Transfer in terms of clock cycles, respectively, and the number of multiplexers represents the required number of 1-2 multiplexers (MUX-2). It can be seen that the shared bus structure can be implemented with a fewer number of switches, but its delay is higher. The next example investigates this problem with a more complicated design.

TABLE 5.1: Comparing multiplexer-based and bus-oriented methods for Example (5.1).

Binding	**MUX-Based**		**Bus-Based**	
(ALU,MUL)	MUX No.	Latency	BSW No.	Latency
(1,1)	6	$6T_{MUL} + 2T_{ALU}$	2	$6T_{MUL} + 2T_{ALU} + 18T_{DT}$
(2,1)	5	$3T_{MUL} + 2T_{ALU}$	3	$3T_{MUL} + 2T_{ALU} + 18T_{DT}$
(1,2)	6	$6T_{MUL} + T_{ALU}$	3	$6T_{MUL} + T_{ALU} + 18T_{DT}$
(2,2)	5	$3T_{MUL} + T_{ALU}$	4	$3T_{MUL} + T_{ALU} + 18T_{DT}$

Example 5.2. *The Fast Fourier Transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) [105]. FFT has a great importance to a wide variety of intensively computational applications such as digital signal processing (DSP), solving partial differential equations and algorithms for quickly multiplying large integers [105]. Figure (5.5) shows the basic block diagram of a radix-2 butterfly cell in the form of Dissemination In Time (DIT) [98], which is the building block of the FFT algorithm. This block diagram is implemented by adders and multipliers and the corresponding scheduling diagram is depicted in Figure (5.6). See appendix A for more details.*

In Figure (5.6) it is assumed that there are two multipliers and two adder/subtractors available. Numbers next to the connection lines indicate the Control step (C-step) in which data transfer takes place on that interconnection, these values are extracted from the scheduling table. This design is implemented and compared with peer-to-peer and shared-bus methods and the results are in Table (5.2). Similar to the previous example,

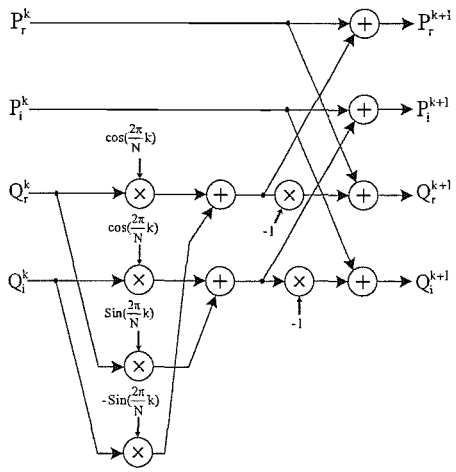


FIGURE 5.5: Radix-2 FFT Butterfly Cell block diagram DIT form.

it can be observed from the results that data transfer delay is very high in the shared-bus method, however, the data steering cost shows reduction.

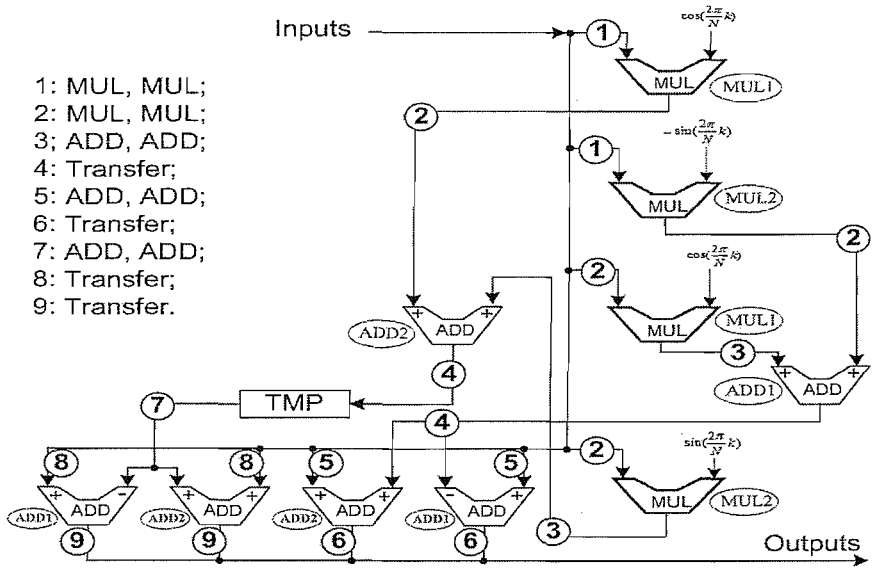


FIGURE 5.6: Scheduling diagram for radix-2 butterfly DIT form.

TABLE 5.2: Comparing multiplexer-based and bus-oriented methods for radix-2 DIT FFT.

Binding	**MUX-Based**		**Bus-Based**	
(ALU,MUL)	MUX No.	Latency	BSW No.	Latency
(1,1)	6	$4T_{MUL} + 6T_{ALU}$	2	$4T_{MUL} + 6T_{ALU} + 21T_{DT}$
(2,1)	6	$4T_{MUL} + 3T_{ALU}$	3	$4T_{MUL} + 3T_{ALU} + 21T_{DT}$
(1,2)	6	$2T_{MUL} + 6T_{ALU}$	3	$2T_{MUL} + 6T_{ALU} + 21T_{DT}$
(2,2)	6	$2T_{MUL} + 3T_{ALU}$	4	$2T_{MUL} + 3T_{ALU} + 21T_{DT}$

These results suggest that data steering can be implemented by bus-oriented structures with fewer switches but data transfers will take more time. However, using a shared-bus structure can cause a considerable delay in data transfer in comparison with a multiplexer based method, where, because of a limited number of communication paths, every data transfer competes with others to occupy a path. Consequently, scheduling and bus binding for data transfers are as important as the operation scheduling, because an operation cannot be executed unless its operands are transferred to the input latches of the corresponding functional unit [128].

The most trivial way to improve this structure is to utilise a partitioned bus with a local communication data transfer scheme [40]. In the partitioned-bus architecture, the shared bus is divided into a limited number of bus segments in the hope that the communication path shortage problem will be alleviated due to the localisation of data transfers. However, this approach needs more careful scheduling and binding for data transfers [116] [75].

5.3 Bus-Oriented Datapath

Initially, a system specification consists of functional units and data communication between them. Data flow graphs for instance, as a broadly accepted representation of the datapath, are traditionally focused on the functional units and data transfer operations are considered as a secondary concern. Consequently, design optimisation steps are focused on functional units and their implementation costs [128].

Regarding the data communication structure in the system, different approaches have been introduced for datapath synthesis which form a top-level point of view, two extreme cases could be realised for system interconnection: a single shared bus and a fully connected peer to peer structure, both of which are depicted in Figure (5.7) in terms of DFG over C-steps. The shared bus is accessible for all the sub-blocks without any priority whereas in the peer-to-peer structure every data transfer has a dedicated interconnection. The first one has a very simple implementation, however in massive data processing systems with many sub-blocks it can be severely slow. In the fully connected peer to peer structure, on the other hand, the result by system could be faster but more costly in implementation.

By more investigation these two extreme cases, from a high level synthesis viewpoint, can be annotated on the DFG representation of the datapath in the form of the data transfer restrictions. Consider Figure (5.8) which shows a data flow graph representation of an example datapath, in which 'T' represents data transfer. In Figure (5.8-a), the DFG depicted using a C-steps time scale, assumes no restriction in data communications between operations. In other words, it supposes to there is a data communication channel available for every transfer whenever it is needed. Nevertheless there is no

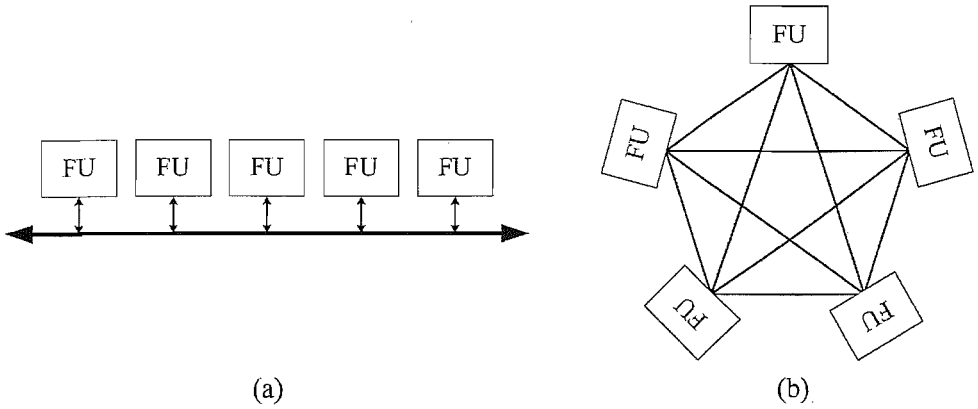


FIGURE 5.7: Sub-Blocks bus structures a)Single shared bus b)Fully connected peer to peer.

assumption concerning the implementation of the DFG, normally multiplexers are used for implementation of such a data transfer structure in the datapath synthesis.

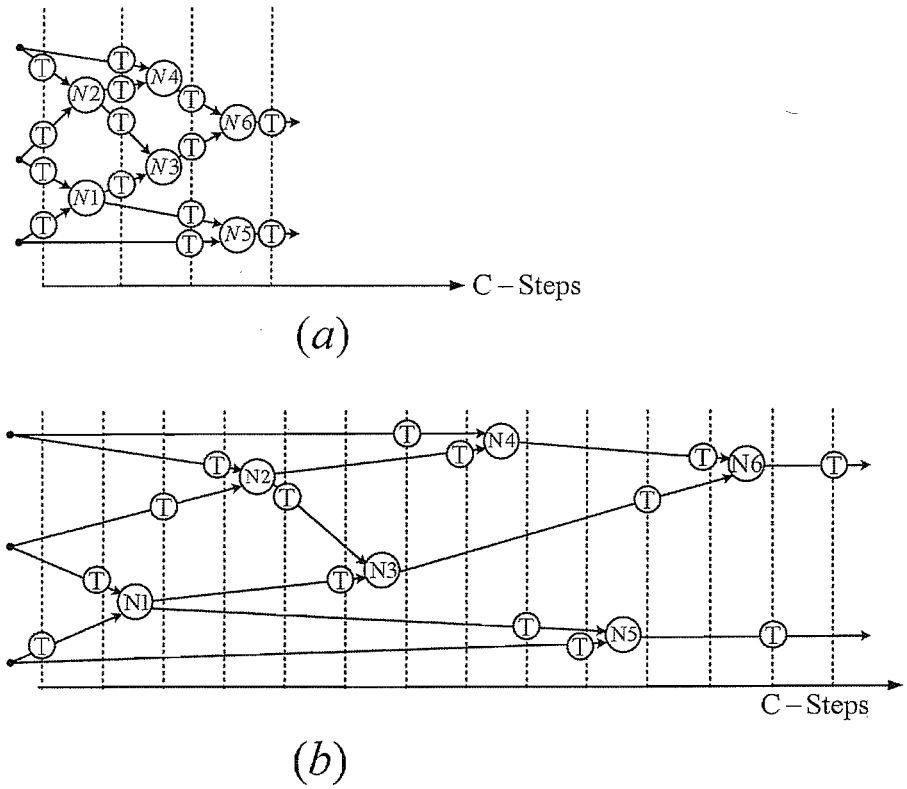


FIGURE 5.8: Data transfer in DFG a)Without limit in data transfer in each C-step (Fully connected peer to peer) b)Restricted to one data transfer in each C-step (Single shared bus).

In Figure (5.8-b) on the other hand, an extreme restriction is applied to the data communication structure which assumes only a single shared data communication channel or bus for all sub-systems. As depicted in Figure (5.8-b), in this case only one data transfer can take place in each C-step which means that the DFG must be expanded along

the C-step (time) axis to avoid any data collision between sub-systems. This structure can be realised in the form of a single shared bus in which one set of interconnections are shared among all the functional units as a data bus. In comparison with the first structure, the main advantages of the shared bus architecture include: simple topology, low cost, and extensibility [114]. It is, however, slow and requires more control overhead in comparison to directly connected units.

To compromise between these two extreme communication synthesis models, we start with a single shared bus and then expand it to more flexible structures. The following subsections provide more details in this regard.

5.3.1 Multiple-Width Partitioned-Bus

In the single shared bus, the costs are directly dependent on the number of functional units connected to the bus. For example, it is observable that the average data transfer delay increases with the number of functional units or bus power consumption is a function of the interconnection capacitance which also is a function of the number of connected functional units and the bus wire-length [55]. Accordingly, it can be concluded that the shared bus is more efficient in cases in which there is a small number of units, that need to be connected to the bus. Thus dividing functional units into smaller groups connected by detached shared buses is a rational approach to improve the data transfer performance.

In a partitioned bus structure, as depicted in Figure (5.9), each bus is divided into segments which are functionally connected by switches. Neighbouring segments, when disconnected, can work independently and convey different data simultaneously. Functional units which do not share the same segments can, therefore, operate in parallel. This structure makes routing and placement easier [75] and also reduces the power consumption of the bus [56]. The concept of segmenting the bus appeared in [40] in the context of single-chip devices. The basic structure of the partitioned-bus architecture consists of a specified number of busses connected in serial, where two adjacent bus segments are connected with symmetrically placed switches. A switch permits or inhibits data flow between segments, thus parallel data communications and processing are possible when two segments are disconnected.

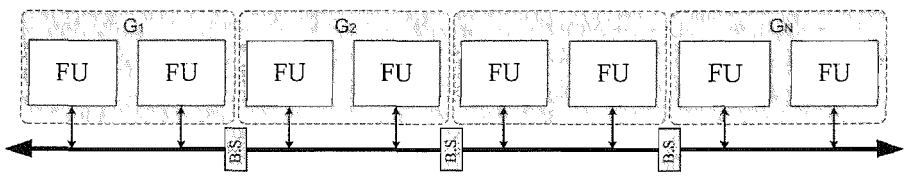


FIGURE 5.9: Partitioned bus structure.

In addition to basic HLS parameters, see section 2.2, the performance of this bus structure is dependent on the grouping of functional units and their allocation to the bus segments. For instance, if data transfer takes place between functional units which are positioned in non-neighbour segments all the intermediate buses are possessed exclusively during the data transfer. Therefore, equalising the data transfer load of each divided bus is needed during high level synthesis, both in the temporal domain and in the spatial domain to take the data transfer into account as a synthesis parameter. This bus structure can also be implemented in the form of a ring connection in which the end segments of the bus are connected by a bus switch.

Choosing the bit-widths of the bus segments is another issue which needs to be considered in the bus structure. Originally, a partitioned bus is considered as a single bus with a uniform bit-width, which is split into segments by bus switches and it is generally agreed to have the same bit-width for all the segments. Bearing in mind that the word-length optimisation has a considerable impact on the overall performance of the system, it is arguable whether all the bus segments should have the same bit-width or not. Assuming all the functional units which are grouped together have the same word-length and are connected to a bus segment, this bus segment must also have the same bit-width as functional groups. Other groups and their corresponding bus segments may or may not have different bit-widths, we called this structure Multiple Width Partitioned Bus (MWP-Bus).

It is need to be emphasised here that the the MWP-Bus is just a structure for datapath and its cost and benefits must be considered overall with the functional units. However, generally speaking, reusing the number of wires in design can reduce design cost but in our study, number of segments, functional units allocation to the segments in combination with bit-width of the segments are utlised by optimisation tool to find the optimum design for datapath. Actually, optimizer makes decision whether bus segments should have different widths or not based on the functional units grouping and scheduling and systems overall implementation cost.

The word-length optimisation method is based on the computational error sources and propagation models as explained in chapter 3 and 4. Accordingly, in addition to functional units which are potential sources of computational error, data exchange between bus segments can also cause a truncation error if the sender bus has a larger bit-width than the destination segment. Since the MSB parts of the bus segments are always connected and truncation takes place in the LSB part, see Figure (5.10), so when data transfer from left to right in Figure (5.10), it results in an error which can be modelled as a truncation error along side other error sources in the system.

This modified structure increases the flexibility of the bus and gives more freedom to the optimiser to find the optimum design, however, it requires more effort in terms of word-length optimisation problem as well as grouping functional units. Form the word-length

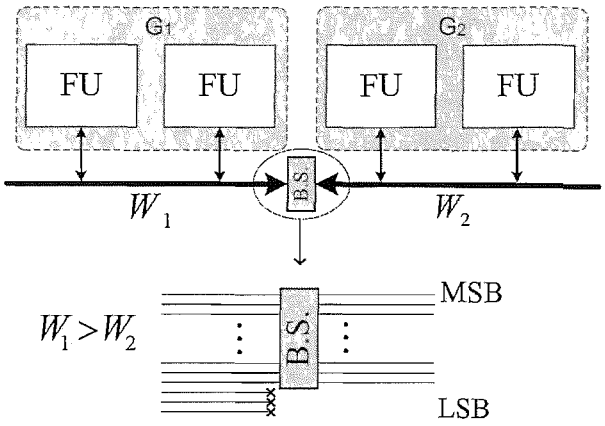


FIGURE 5.10: Bus segments with different widths.

optimisation point of view, the MWP-Bus structure, however, is an attempt to allocate optimum word-lengths to groups of functional units, which results in reduced feasible space of the optimisation search. Since MWP-Bus segments are connected in serial, if two segments communicate, all the segments which are physically placed between them will also be occupied. To improve this, the structure can be extended to multidimensional structures, which is discussed in the following subsection.

5.3.2 Multiple-Width Multiple-Way Partitioned-Bus

The MWP-Bus structure offers only one possible path for each data transfer, so the controller only needs to activate the required Bus-Switches (BSW) just in time to connect functional units. This structure needs a simple control unit and improves data communication speed considerably for a small number of functional units, however, there are difficulties when the number of bus segments increases.

In buses with long length, traffic in the middle part of the bus will be dramatically higher than in the ends of the bus. Let us assume a partitioned bus with N serial segments, where the probability of the segment m sending data to segment n in time t is $P_{m,n}(t)$, then the probability that segment k has been occupied by data transfer of other segments communication at the time t is $P_k(t)$, which can be calculated as

$$P_k(t) = \sum_{i=1}^{k}\sum_{j=k}^{N}(P_{i,j}(t)\cdot q_{i,j}(t)) + \sum_{i=k}^{N}\sum_{j=1}^{k}(P_{i,j}(t)\cdot q_{i,j}(t)), \tag{5.2}$$

where $q_{i,j}(t)$ is the probability of the path from segment i to segment j is not occupied by other segments data exchange at time t. According to Figure (5.11), the path from segment i to segment j is not occupied by other segments data exchange if data would not be sent:

- Case 1: Sending data from region A to B or C,
- Case 2: Sending data from region B to A or B or C,
- Case 3: Sending data from region C to A or B,

which means that $q_{i,j}(t)$ can be calculated as

$$q_{i,j}(t) = \overbrace{\prod_{l=1}^{i-1}\prod_{m=i}^{N}(1-P_{l,m}(t))}^{\text{Case 1}} \cdot \overbrace{\prod_{l=i}^{j}\prod_{m=1}^{N}(1-P_{l,m}(t))}^{\text{Case 2}} \cdot \overbrace{\prod_{l=j+1}^{N}\prod_{m=1}^{j}(1-P_{l,m}(t))}^{\text{Case 3}}, \quad (5.3)$$

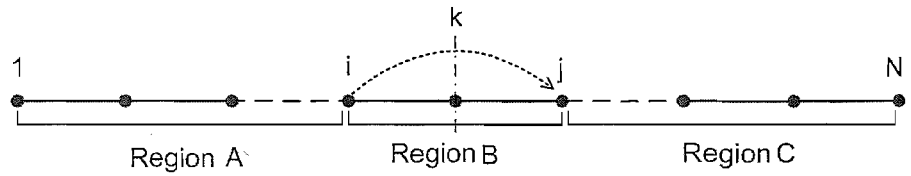


FIGURE 5.11: Sending data from segment $i(< k)$ to segment $j(> k)$ divides the MWP-Bus into three parts.

In a real implementation of the bus structure, however, the probability functions $P_{m,n}(t)$ are dependent on each other because whenever there is no available segment to create a path, which is called path congestion, data communication between segments needs to be postponed to the time at which a path is available for data transfer. Even regardless of this fact, Equation (5.2) shows that in designs with a large number of functional units and consequently, a large number of bus segments, data communication latency will inevitably increase in MWP-Bus. Table (5.3) shows these probabilities for buses with different numbers of segments in the simplified case that assumes $P_{i,j} = p_0$ and $P_{i,i} = 0$ for all the segments. It can be observed from Table (5.3) that data traffic in the central part of the bus increases with the number of segments if p_0 is $<< 1$. In other words, if the probability of data exchange between bus segments is low, data traffic in the middle part of the MWP-Bus is expected to be the bottle-neck of data transfer between functional units which are placed in different bus segments.

TABLE 5.3: Comparing occupation probabilities for the segments of the buses with different number of segments.

N	Place of the segment in MWP-Bus				
	1	2	3	4	5
3	$4p_0(1-p_0)^3$	$6p_0(1-p_0)^5$	$4p_0(1-p_0)^3$	-	-
4	$6p_0(1-p_0)^5$	$10p_0(1-p_0)^9$	$10p_0(1-p_0)^9$	$6p_0(1-p_0)^5$	-
5	$8p_0(1-p_0)^7$	$14p_0(1-p_0)^{13}$	$18p_0(1-p_0)^{17}$	$14p_0(1-p_0)^{13}$	$8p_0(1-p_0)^7$

A possible extension, to avoid this problem, is a bus structure which provides more than one path between some points in the datapath using parallel paths across the bus

segments. Data transfer schemes and the related controller will be more complicated but this extension improves the data transfer latency significantly, see Figure (5.12) for simple examples of this proposed extension which is called Multiple-Way Multiple-Width Partitioned Bus (MW^2P-Bus). FUs in the Figure (5.12) are considered as single

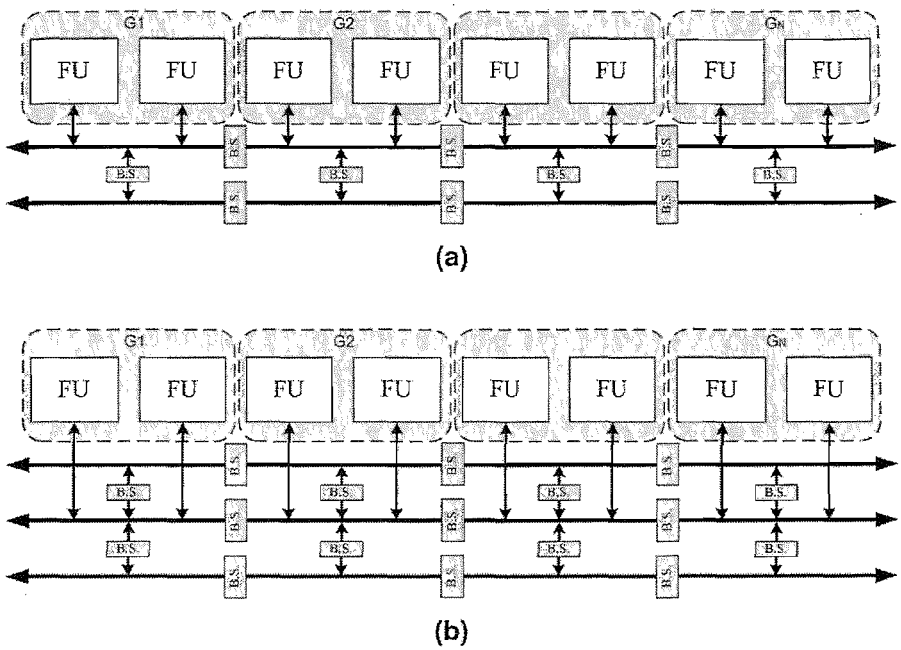


FIGURE 5.12: Multiple-way partitioned bus a)Two parallel line partitioned shared bus c)Three parallel line partitioned shared bus.

input units which does not reduce the generality of the method because in the case of multi-input FUs, each input or output can be connected to the same or different bus segments.

These MW^2P-Bus can be envisaged as a set of parallel MWP-Buses with segments connected by bus switches. For more than three parallel buses, the structure forms three dimensional shapes, as shown in Figure (5.13), where the resulting structures are polygonal prism lattice buses in which the vertices are bus segments and the edges are bus switches. In the Figure (5.13), M and N represent the number of parallel lines and the number of serial segments respectively. This prism bus structure can be extended to a torus shape by connecting the ends together, as depicted in Figure (5.14), to reduce the average communication path lengths.

The major problem of these structures is the categorisation of the sub-blocks according to their data traffic. In other words, to find which sub-blocks must placed together in a group or which bus structure must be used, a profile of sub-block data transfer during system operation is required from which the synthesiser/optimiser can make a decision to put which sub-blocks in which group and what bus structure should be used.

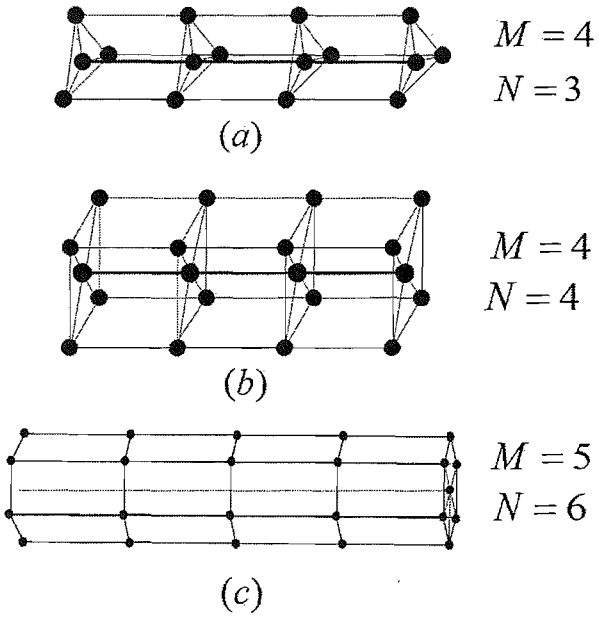


FIGURE 5.13: The proposed bus structure in higher dimension a)Four-line structure b)Five-line structure c)General prism structure.

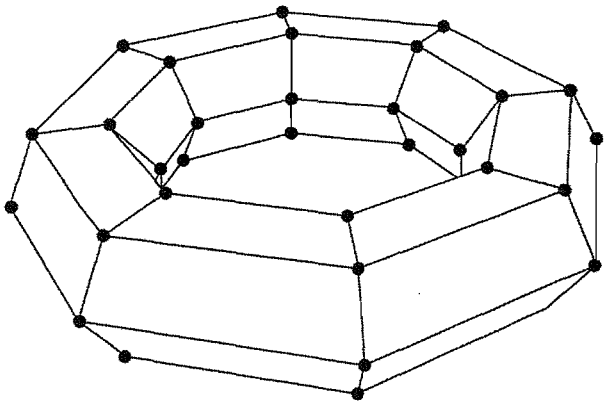


FIGURE 5.14: The proposed bus structure in a torus form.

Since the MW^{2}P-Bus structures are polygonal prism lattice buses which require 3-D wiring, in general, the interconnection complexity increases with the bus dimension and the number of nodes. This bus structure can be reduced by the synthesiser to simpler forms by pruning unused segments or trading off the speed with trimming the less-used edges during the optimisation process.

Figure (5.15) illustrates the simplification idea for a planar lattice structure. In this figure, a MW^{2}P-Bus with $N = 2$ and $M = 4$ is depicted, where Figure (5.15-a) shows a fully implemented bus on which all the required combinational paths are shown by dashed arrows. Consequently, there are some bus segments and bus switches that can be pruned from the bus without any reduction in communication capability of the bus, as depicted in Figure (5.15-b). In comparison with the full implementation, which needs

12 bus segments and 14 bus switches, the pruned form only needs 5 bus segments and 7 bus switches, which is a considerable improvement.

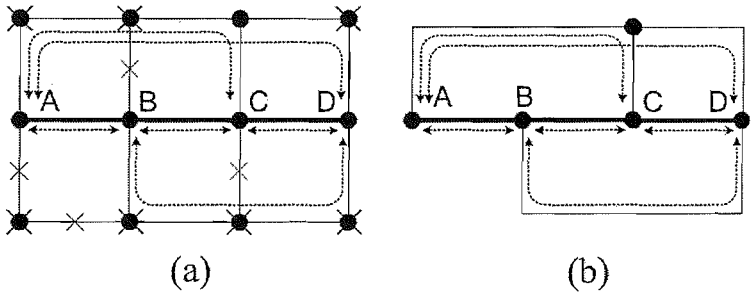


FIGURE 5.15: Graph model for the three line bus as in Figure (5.13-b) a)General structure b)Same bus after pruning unused segments.

This example suggests that (regarding maximum required path), to connect all the possible combinations of the segments, the simplified form of the MW^{2}P-Bus can directly be derived from M.. Consider a MW^{2}P-Bus with $M = 5$, which is depicted in Figure (5.16), to illustrate the issue. Figure (5.16-a) shows the fully implemented bus with the required paths shown with dashed arrows. Similar to Figure (5.15-b), the pruned bus is extracted as shown in Figure (5.16-b). In comparison with the full implementation which needs 20 bus segments and 31 bus switches, the pruned form needs 8 bus segments and 13 bus switches. Even this reduced structure might be pruned by the synthesiser after allocation and scheduling and when it is figured out that some of the bus segments and/or bus switches are not used in the design.

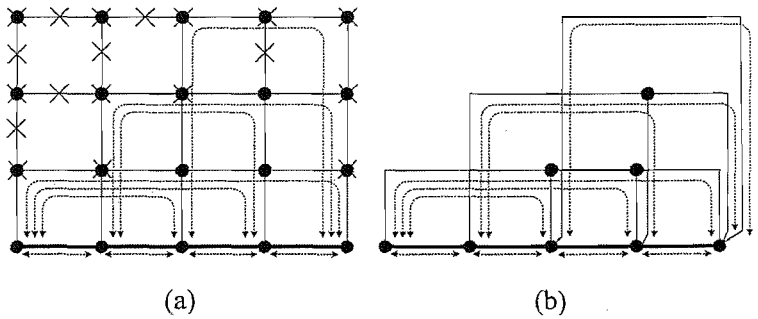


FIGURE 5.16: Graph model for a MW^{2}P-Bus with $M = 5$ a)Full implementation b)Same bus after pruning unused segments.

To generalise this reduced structure, consider Figure (5.17). Let assume that all the functional units are connected to the central bus segments in Figures (5.13), which we call base-segments, so all other parallel segments are only used to provide the required connections between the base-segments. We allocate paths for all possible combinations of two segments, starting with the left-side base-segments. It can be observed from Figure (5.17-a) that this allocation occupies the first level of parallel segments. In consequence, all other base-segments are similarly allocated to the paths which are required to connect them to other base-segments as depicted in Figure (5.17). After elimination of

unused parallel segments, the reduced structure of the bus is achieved in Figure (5.17-b), which has a considerably lower complexity and implementation cost. Even this reduced structure is only needed in designs in which all the base-segments have communications with each other at the same time, otherwise after allocation and scheduling, see section 2.2, this reduced structure might be reduced to a simpler structure.

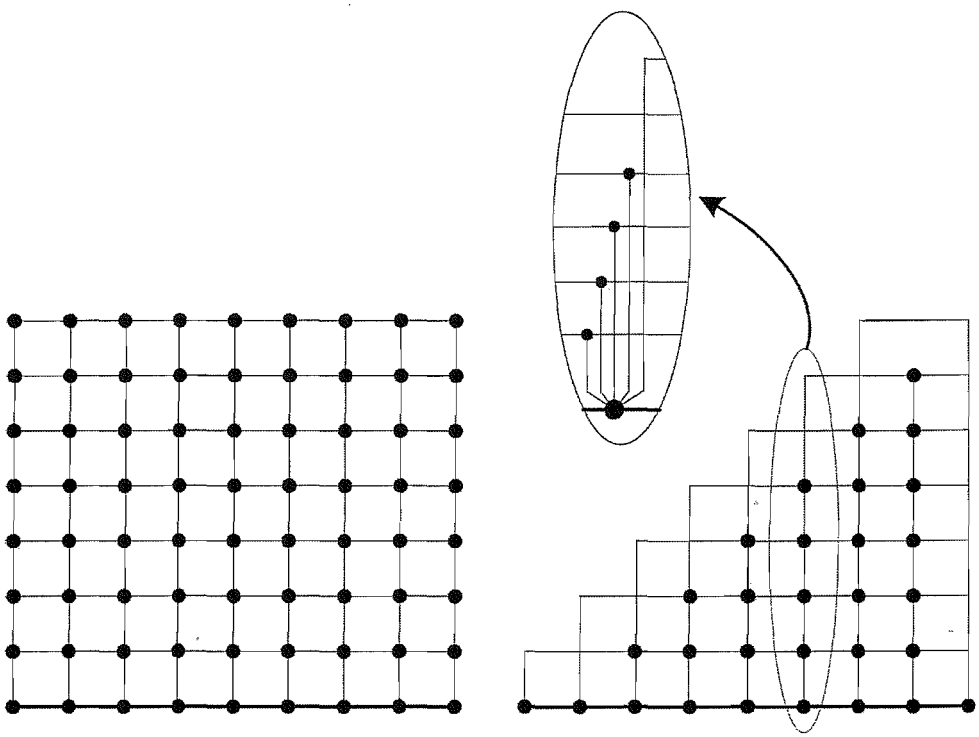


FIGURE 5.17: Graph model for a MW^{2}P-Bus mapped into a Manhattan grid a)Full implementation b)Same bus after pruning unused segments.

To estimate the required bus segments and bus switches in the reduced form of the MW^{2}P-Bus, let us count the number of bus segments, N_{BS}, and the number of bus switches, N_{SW}. According to Figure (5.17-b) N_{BS} can be calculated as

$$\begin{aligned} N_{BS} &= (1+2+\cdots(M-3))+M, \\ &= \frac{(M-3)^2+(M-3)}{2}+M, \\ &= \frac{M^2-3M+6}{2}. \end{aligned} \tag{5.4}$$

N_{SW} can be calculated as:

$$\begin{aligned} N_{SW} &= \overbrace{(1+2+\cdots(M-1))}^{\text{horizontal lines}}+\overbrace{(1+2+\cdots(M-3))}^{\text{vertical lines}} \\ &= \frac{(M-1)^2+(M-1)}{2}+\frac{(M-3)^2+(M-3)}{2}, \\ &= M^2-3M+3, \end{aligned} \tag{5.5}$$

where in both it is assumed that $M > 3$.

One parameter which is also useful in the cost estimation of this bus structure is the number of grid segments where the bus is mapped to the Manhattan Grid [124]. Due to the three-dimensional structure of the bus, it cannot be mapped to a two-dimensional grid such as a Manhattan Grid without overlaps of some edges in the grid, as is shown in Figure (5.16-b). After this mapping we count the number of the covered grid segments to approximate the wiring cost of the bus. Accordingly, for a bus with M serial base-segments, the reduced form in Figure (5.17-b) has N_{GS} grid segments which consists of horizontal ($N_{H,GS}$) and vertical ($N_{V,GS}$) lines. N_{GS} can therefore be written in the form of Equation (5.6).

$$\begin{aligned} N_{GS} &= N_{H,GS} + N_{V,GS}, \\ &= M - 1 + ((M-1) + (M-2) + \cdots + 3 + 2) + N_{V,GS}, \\ &= \frac{M^2 + M - 4}{2} + N_{V,GS}, \end{aligned} \tag{5.6}$$

where according to Figure (5.17) $N_{V,GS}$ can be written as:

$$\begin{aligned} N_{V,GS} &= (1) + (1+2) + (1+2+3) + \cdots + (1+2+\cdots+M-3), \\ &= \sum_{j=1}^{M-3} \sum_{i=1}^{j} i = \frac{1}{2} \sum_{j=1}^{M-3} (j + j^2), \\ &= \frac{1}{2} \left(\frac{(M-3)^2 + (M-3)}{2} + \frac{2(M-3)^3 + 3(M-3)^2 + (M-3)}{6} \right), \\ &= \frac{1}{2} \left(\frac{2M^3 - 3M^2 + M - 30}{6} + \frac{M^2 - 5M + 6}{2} \right), \\ &= \frac{M^3 - 7M - 6}{6}. \end{aligned} \tag{5.7}$$

Then the number of the grid segments in the reduced form of MW^{2}P-Bus (N_{GS}), with M serial segments, can be derived as in Equation (5.8).

$$\begin{aligned} N_{GS} &= \frac{M^2 + M - 4}{2} + \frac{M^3 - 7M - 6}{6}, \\ &= \frac{M^3 + 3M^2 - 4M - 18}{6}, \qquad \text{where } M > 3, \end{aligned} \tag{5.8}$$

This equation for $M = 4$ and $M = 5$ results in $N_{GS} = 13$ and $N_{GS} = 27$ respectively which can be confirmed by inspecting Figures (5.15-b) and (5.16-b).

Since this bus structure replaces multiplexers in datapath synthesis, another interesting problem compares the circuit level implementation of the bus switches and multiplexers. Bus switches which are a key element in this bus structure can be implemented in the form of gated inverting-buffers as depicted in Figure (5.18-b), where the simplest CMOS

implementation of a 2-input multiplexer using pass-transistors is as in Figure (5.18-a), more details can be found in [108]. Comparing the two circuits shows that a bus switch needs more transistors than a 2-input multiplexer (6 against 12), however, the number of required transistors will increase geometrically with the number of inputs of the multiplexer (18 for MUX-4 and 42 for MUX-8 for instance).

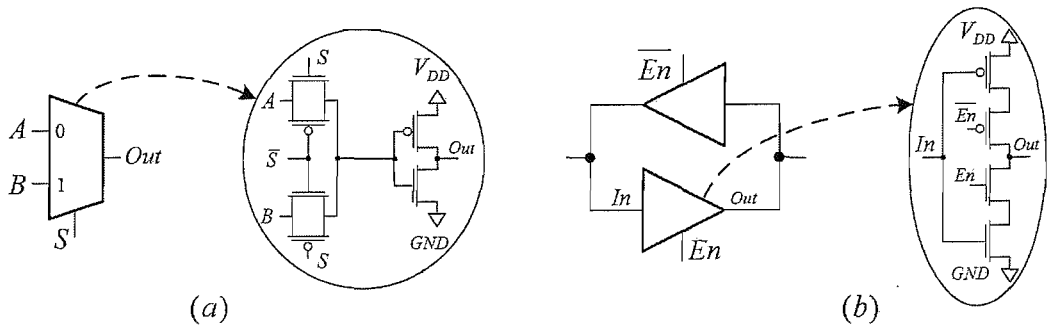


FIGURE 5.18: Comparing Bus Switches and Multiplexers a) Single-bit MUX-2 circuit b) Single-bit bus switch circuit.

5.4 Wire Length Estimation

Rent's Rule, [124], gives a complexity measure of the interconnection topology and the quality of the placement using circuit partitioning. This is the relationship between the average number of terminals (or pins), T, of a part of a circuit (a bin) and the average number of cells/macros (Basic Logic Blocks, B) inside the bin and t, which is the average number of terminals per cell/macro. The relationship between these parameters is therefore given by Rent's law as in Equation (5.9).

$$T = t \cdot B^p, \tag{5.9}$$

where p is called the Rent exponent.

There are wire length estimations based on Rent's rule [124], but since wire-length estimation is out of scope of this work and also we only need an approximation which can be used to compare MW^{2}P-Bus structures during the optimisation process, here by flattening a general form of the MW^{2}P-Bus to a modified Manhattan grid, as depicted in Figure (5.17), the MW^{2}P-Bus wire-length is approximated as a function of the node count of the grid. Knowing that every node must be connected to at least one edge, because of the lattice connectivity, MW^{2}P-Bus wire length is a function of N_{SW} (see Equation (5.5)).

On the other hand, every segment in MW^{2}P-Bus has a distinct bit-width which multiplies the number of its wires. Accordingly the wire length for a MW^{2}P-Bus bus is

$$L_B = F(\overrightarrow{W}, M), \tag{5.10}$$

where $\overrightarrow{W}$ is the array of word-lengths of the bus segments as in Equation (5.11).

$$\overrightarrow{W} = \begin{bmatrix} w_1 & w_2 & \cdots & w_{N_B} \end{bmatrix}, \tag{5.11}$$

where w_i refers to bit-width of bus segment i.

In general, $F(\overrightarrow{W}, M)$ in Equation (5.10) is not only a nonlinear function of $\overrightarrow{W}$ and M but is also a function of floor-planning/routing algorithm as well as the target technology. Then, assuming a feasible place and route of the functional units, Equation (5.8) can be used to derive an upper limit for wire-length of the MW^{2}P-Bus, as in Equation (5.12).

$$L_B \leq N_{GS} \cdot \overline{w}, \tag{5.12}$$

where the $\overline{w}$ is the average bit-width of the bus segments. Since the complexity of the function in Equation (5.8) affects the optimisation performance, a linear form of the equation is applied in our cost model as in Equation (5.13).

$$L_B \approx A \cdot \frac{M^3 + 3M^2 - 4M - 18}{6} \cdot \overline{w}, \tag{5.13}$$

where A represents an empirical constant coefficient and $\overline{w}$ is the average bit-width of the bus segments. This model provides a linear relationship with the average word-length and a polynomial (third degree) nonlinear relationship with the number of MW^{2}P-Bus base-segments (M).

It should be noted that this expression represents the average wire length for the MW^{2}P-Bus ($\overline{L_B} = \frac{L_B}{N_B}$) but not the interconnection cost, which should be specified in a hierarchically partitioned circuit placement optimisation. While the model of Equation (5.13) is a plausible model, we utilise it within the context of the given modelling problem in the optimisation algorithm in chapter 7.

5.5 Implementation Algorithm

The proposed structure for data communication in the datapath, in combination with word-length optimisation forms a high level synthesis method. For the sake of clarifying the method, here, we only consider the bus structure and functional units allocated to the bus segments, which means that the HLS sub-tasks (see section 2.2) are solved and the number of the required functional units is known and all the bus segments are also

considered to have a uniform bit-width. The implementation algorithm consists of the following steps:

1. Choose the number of the base-segments, M, of the MW^{2}P-Bus,
2. Divide the functional units into M groups,
3. Find data transfers between groups in each C-step,
4. Allocate the functional units to the base-segments to provide them with the required path for every data transfer in the DFG with minimum intra-group data transfers,
5. For all the C-steps:
   (a) For all the base-segments: start from the far-left segment to the far-right segment,
      i. allocate available segments to the required paths in the DFG,
6. If there are any unallocated paths (congestion), go to step (1) and increase M,
7. Prune the unused bus segments,

Each step of this algorithm contains sub-steps which are implemented and discussed in the optimisation algorithm.

The path allocation algorithm for inter-group data exchange, in its general form, is a routing problem or dynamic shortest path problem. To simplify and make it practical for the cost evaluation algorithm, we restate the problem as follow: "Find the shortest path between two nodes when the most left-hand side node has the highest priority and the priority of other nodes decreases from left to right". If a node requires an inter-group data transfer and it has priority, the algorithm allocates the path to it at the corresponding C-step. In the case that no path is available, this data transfer will be postponed by one clock cycle when new paths will be available. This method is not the optimum way to route the paths but it is practical. Another issue which might be considered during optimisation is the possibility of trading MW^{2}P-Bus segments with latency. It is reasonable to restrict MW^{2}P-Bus segments to a certain number to reduce the costs at the price of speed.

From an optimisation point of view, MW^{2}P-Bus is combination of two problems: bus segmentation and word-length optimisation. It is proved in [115] that the bus segmentation problem is an NP-hard optimisation problem. In [29] word-length optimisation is also demonstrated to be a NP-hard problem. So a combination of these two problems should be more difficult to solve by exact methods of optimisation in practical a time. Accordingly, stochastic methods of optimisation are more viable in practice. It means

that the step (1) and (2) in the algorithm can be implemented in the form of a random walking search to find an optimal solution for the problem. We introduce a genetic algorithm search for this problem in chapter 7.

Example 5.3. *MW²P-Bus implementation of Example (5.1).*

Figure (5.19) demonstrates the operations scheduling with one multiplier and one ALU. In this figure, numbers indicate the control steps on which control signals are applied to the functional units to start the operations or transfer the data. Table (5.4) shows data transfers in each C-step of Example (5.1) after scheduling and allocation for implementation with one multiplier (MUL) and one ALU with the assumption of a SISO (Single Input Single Output) structure for this circuit. In this table, "→" represents data transfer and the index shows the input number of the functional units (for instance, MUL_1 is the first input of the multiplier). Since the design has only one input, it is impossible to read two different values from the input in a single C-step, which is resulted in $2T_D$ (data transfer delay) extra time compared with Table (5.1). It is also assumed that the operation delay of the adder and multiplier are the same. The MW²P-Bus implementation is depicted in Figure (5.20). To show a different style of design which can be employed in this approach, we consider input and output terminals of the functional units separately and deal with them as independent data exchange points in the circuit.

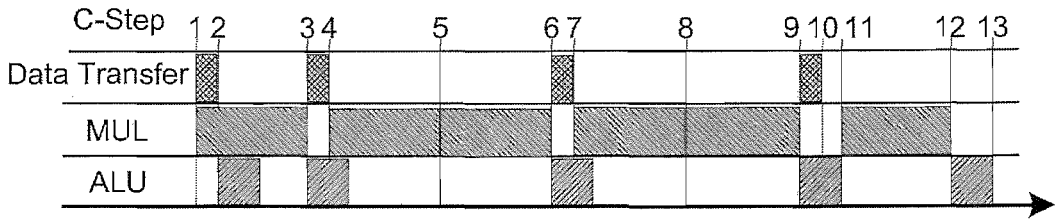


FIGURE 5.19: Data transfers in each C-step for Example (5.1) implemented with one multiplier and one ALU.

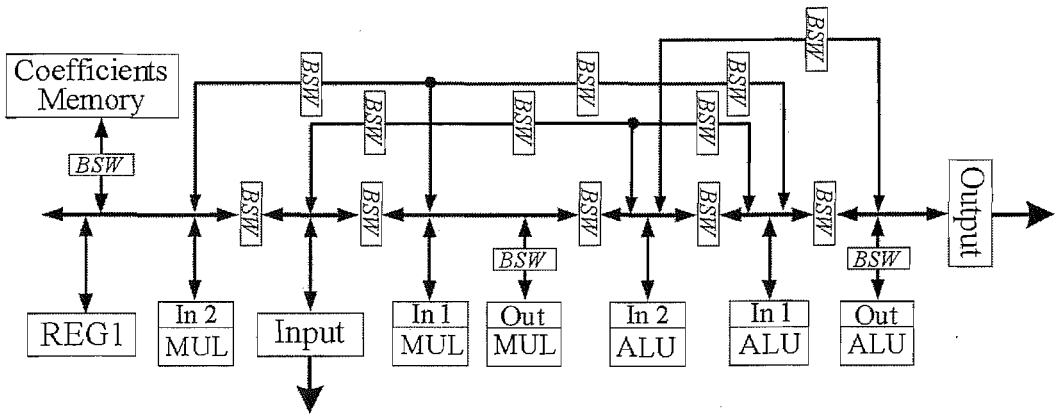


FIGURE 5.20: MWP²P-Bus implementation of the Example (5.1) circuit.

One of the advantages of this method is the flexible structure which provides the option for the designer to trade off latency against bus interconnection complexity. Accordingly,

TABLE 5.4: Data transfers in each C-step for Example (5.1) implemented with one multiplier and one ALU.

C-step	Data Transfers:		Delay
1	$x \rightarrow \text{MUL}_1$ $x \rightarrow \text{ALU}_1$	$a_1 \rightarrow \text{MUL}_2$	
2	$dx \rightarrow \text{ALU}_2$	$\text{MUL} \rightarrow \text{REG1}$	T_{MUL}
3	$a_3 \rightarrow \text{ALU}_1$ $dx \rightarrow \text{MUL}_1$	$\text{ALU} \rightarrow \text{ALU}_2$	
4	$u \rightarrow \text{MUL}_2$		
5	$\text{MUL} \rightarrow \text{MUL}_1$ $\text{ALU} \rightarrow c$	$\text{REG1} \rightarrow \text{MUL}_2$	$T_{MUL} + T_D$
6	$u \rightarrow \text{ALU}_1$	$\text{MUL} \rightarrow \text{ALU}_2$	T_{MUL}
7	$y \rightarrow \text{MUL}_1$	$a_2 \rightarrow \text{MUL}_2$	
8	$\text{MUL} \rightarrow \text{MUL}_1$	$dx \rightarrow \text{MUL}_2$	$T_{MUL} + T_D$
9	$\text{ALU} \rightarrow \text{ALU}_1$	$\text{MUL} \rightarrow \text{ALU}_2$	T_{MUL}
10	$u \rightarrow \text{MUL}_1$		
11	$dx \rightarrow \text{MUL}_2$	$\text{ALU} \rightarrow y_f$	
12	$dx \rightarrow \text{ALU}_1$	$\text{MUL} \rightarrow \text{ALU}_2$	$T_{MUL} + T_{ALU}$
13	$\text{ALU} \rightarrow u_f$		T_{ALU}
Total Delay:		$6T_{MUL} + 2T_{ALU} + 2T_D$	

the circuit of Figure (5.20) shows the maximum required complexity to achieve the same latency as peer-to-peer data communication, in which the circuit can be reduced to lower complexity with lower speed. It is also observable that this implementation provides more flexibility in data communication in that there are several paths available for every data transfer which can be useful to reconfigure the system on fly.

Example 5.4. *MW^2P-Bus implementation of Example (5.2).*

Figure (5.21) demonstrates the operations scheduling with two multipliers and two ALUs. In this figure, numbers indicate the control steps on which control signals are applied to the functional units to start the operations or transfer the data. Data transfers in each C-step of Example (5.2) after scheduling and allocation for implementation with two multipliers (MUL) and two adders/subtractors can be found in Table (5.5) and more details can be seen in appendix B. In Table (5.5), MUL stands for multipliers and ADS represents adder/subtractor and the indices show the inputs number. Again we assume a SISO structure for this circuit, delay time, therefore is $4T_D$ longer compared with Table (5.2) because of extra time which is required for data transfer from input or to output of the circuit. Grouping of the functional units to minimise the intra-groups transactions is shown in Table (5.6) and Table (5.7) shows data transactions between these groups in different C-steps. The result MW^2P-Bus implementation is depicted in Figure (5.22).

The second example shows that by correctly using the bus structure, the same speed as peer-to-peer implementation is achievable.

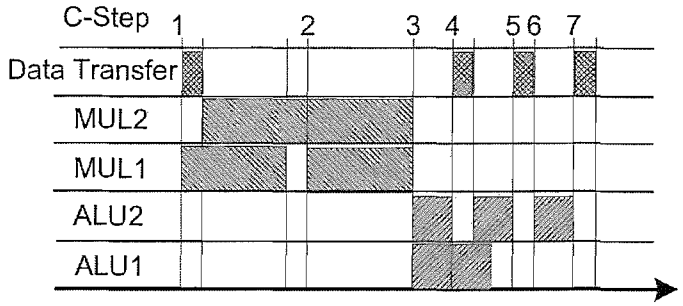


FIGURE 5.21: Operations in each C-step for Example (5.2) implemented with two multiplier and two ALU.

TABLE 5.5: Data transfers in each C-step for Example (5.2) implemented with two multiplier and two ALU.

C-step	Data Transfers:		Delay
1	$Q_r^k \rightarrow \text{MUL1}_1$	$b_0 \rightarrow \text{MUL1}_2$	
	$Q_i^k \rightarrow \text{MUL2}_1$	$b_1 \rightarrow \text{MUL2}_2$	$T_{MUL} + T_D$
2	$\text{MUL1} \rightarrow \text{ADS1}_1$	$b_3 \rightarrow \text{MUL1}_2$	
	$\text{MUL2} \rightarrow \text{ADS2}_1$	$b_4 \rightarrow \text{MUL2}_2$	T_{MUL}
3	$\text{MUL1} \rightarrow \text{ADS2}_2$	$\text{MUL2} \rightarrow \text{ADS1}_2$	T_{ALU}
4	$\text{ADS1} \rightarrow \text{REG1,ADS1}_1$	$\text{ADS2} \rightarrow \text{REG2,ADS2}_1$	
	$P_r^k \rightarrow \text{ADS1}_2$	$P_i^k \rightarrow \text{ADS2}_2$	$T_{ALU} + T_D$
5	$\text{ADS1} \rightarrow P_r^{k+1}$	$\text{ADS2} \rightarrow P_i^{k+1}$	T_D
6	$\text{REG1} \rightarrow \text{ADS2}_1$	$\text{REG2} \rightarrow \text{ADS1}_1$	T_{ALU}
7	$\text{ADS1} \rightarrow Q_r^{k+1}$	$\text{ADS2} \rightarrow Q_i^{k+1}$	T_D
Total Delay:		$2T_{MUL} + 3T_{ALU} + 4T_D$	

TABLE 5.6: Functional units grouping result.

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
IN	MUL1_{out}	MUL2_{out}	ADS1_{out}	ADS2_{out}	MEM
MUL1_1	ADS2_2	ADS1_2	ADS1_1	ADS2_1	MUL1_2
MUL2_1			REG1	REG2	MUL2_2
			Output		

TABLE 5.7: Intra-groups data transfers.

C-Step	Transfers
2	$2 \rightarrow 4 \quad 3 \rightarrow 5$
4	$1 \rightarrow 3 \quad 1 \rightarrow 2$
5	$5 \rightarrow 4$
6	$4 \rightarrow 5$
7	$5 \rightarrow 4$

It is worth to be reminded that this method is in RTL level of datapath implementation, which means that some problems of architecture level bus design are not applicable here. In other words, place and route methods will be applied to the output of our design method exactly the same way as MUX-based design. Therefore, wire-length

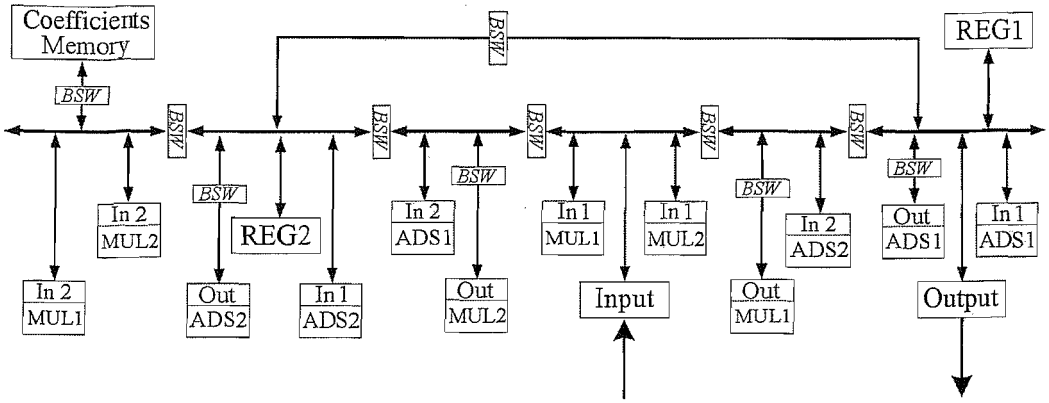


FIGURE 5.22: MWP^{2}P-Bus implementation of the Example (5.4) circuit.

of the interconnections (in average) are not expected to be longer than MUX-based method. Similarly, driving buffers and transistor sizing of the switches are not like bus drivers in architecture level of the systems. They only need to drive funtional units as much as fanout of the gates in the cell-library. However, if functional units grouping and allocation result in big number of functional units are connected to bus segments, length of the bus can be a problem regarding data transfer speed. In this case data communication from one end of the bus to the other end might need more than one clock cycle which is a limitation of the method and must be considered in designs with the big number of functional units.

5.6 Summary

In this chapter a new method for bus-oriented communication synthesis is presented. This method is based on partitioned bus and parallel segments to provide data communication for functional units whenever it is required. In the method every functional unit is connected to a bus segment which is connected to other segments by a bus-switch. These bus-switches are tri-state switches which isolate or connect segments by control signals.

In the general form, these bus segments are connected to each other in the form of a prism or torus to build up a bus structure. This bus structure is a three dimensional bus topology in which the number of serial and parallel segments state the number of available paths for every data exchange between functional units. This method is compared with peer-to-peer (multiplexer-based) method and its capability is discussed. Furthermore, in combination with multiple word-length methods this structure is extended to MW^{2}P-Bus in which (this freedom is added to the structure that) each segment can have a different width from others. This structure provides a very flexible data communication platform for computationally intensive datapath with higher flexibility which can be used in more complicated cases such as reconfigurable hardware. This structure

is applied to the implemented method in the future chapters as a predefined architecture in the proposed high level synthesis method.

Chapter 6

Synthesis Using MW^{2}P-Bus

6.1 Introduction

In this study, a design method is proposed that starts from a hierarchical behavioural specification of the target system and produces a synthesisable Register Transfer Level (RTL) representation. The resulting design is constructed hierarchically using a MW^{2}P-Bus with a flexible structure to match a variety of applications. Besides, it is very modular and manageable for the synthesiser and optimiser. Accordingly, an implementation structure is introduced in detail in this chapter which explains how this structure can be integrated into the synthesis method in the form of a predefined-architecture. This architecture is capable of containing different kinds of sub-blocks which are required to perform an algorithm. Supplementary design examples are provide in appendix B to show the capability of this architecture.

The majority of HLS methods split the target design into two parts: controller and datapath. The controller is a state machine which manages the sequence of operations and controls the datapath blocks and the datapath does the computation. On this basis, the HLS methods translate a high level specification of the design into a synthesisable specification. This translation is not a simple one-to-one mapping between these two specifications, but there exist many different and correct mappings that can be applied to a given description which satisfy the functional correctness and design constraints. Choosing the best, or at least a good, solution among all possible cases is considered as an optimisation problem, which must be performed by a "Synthesiser". In most cases, the synthesiser works in a similar manner to a software compiler in which a high level specification is translated to a low-level implementation based on rules and restrictions using intermediate specifications.

Hardware implementations of the algorithms, can be considered as a class of designs which have some specific characteristics, see section 2.4. Design and implementation

of specific application hardware not only requires information about the field of the application but also, sometimes, demands a set of specific methodologies or tools. Computationally intensive algorithms have characteristics which distinguish them from other digital systems. Due to these common properties, the target design for them can be restricted to improve the synthesis process. In our method, a pre-defined target architecture for the design is used to reduce the feasible space of optimisation and consequently improve the synthesis method.

From a synthesis point of view, on the other hand, this target architecture is a restriction in that it forces the synthesiser to map every design to a pre-defined structure which dominates the feasible solution space in favour of the optimisation performance. Accordingly, the datapath structure is constrained to a MW²P-Bus and a set of pre-defined functional units with configurable arithmetic characteristics. However this structure is a restriction for the synthesiser, but has sufficient flexibility to be used with a wide range of computational algorithms. This chapter represents the proposed method for high level synthesis of computational algorithms which, from an abstract high level synthesis standpoint, is a restricted HLS.

This chapter is organised as follow: first, an overview of the synthesis method is presented; the second section provides details regarding pre-defined architecture and the last section gives details of the synthesis tool.

6.2 Synthesis Method

There are a limited number of functional units which are used to construct most of the hardware implementations of the computational applications. Algorithms which are based on linear operations, for instance, can intrinsically be implemented by adders, multipliers plus memories and the required glue logic. Nonlinear operations, such as elementary functions, also can be evaluated by combinations of the elementary arithmetic operations in the form of mathematical approximations with sufficient accuracy, see [92]. It means that a library of the elementary arithmetic units can be adequate to implement most computational applications. Taking into consideration the discussions regarding target architectures and restricted HLS in sections 2.3 and 2.4, a design method is proposed which is specified for computational algorithm synthesis starting from a high level description of the goal system and is based on three parts: Functional Blocks Data Base, Soft-Architecture and Synthesiser, as depicted in Figure (6.1).

The Functional Blocks Data Base (FBDB) is a library of functions and sub-systems which may be used in computational algorithms. In addition to sub-systems implementation information, this data base must have a full set of optimisation parameters for optimiser cost functions like: area, speed, power consumption and accuracy. It means that each functional block and its constituents must be designed and optimised, then

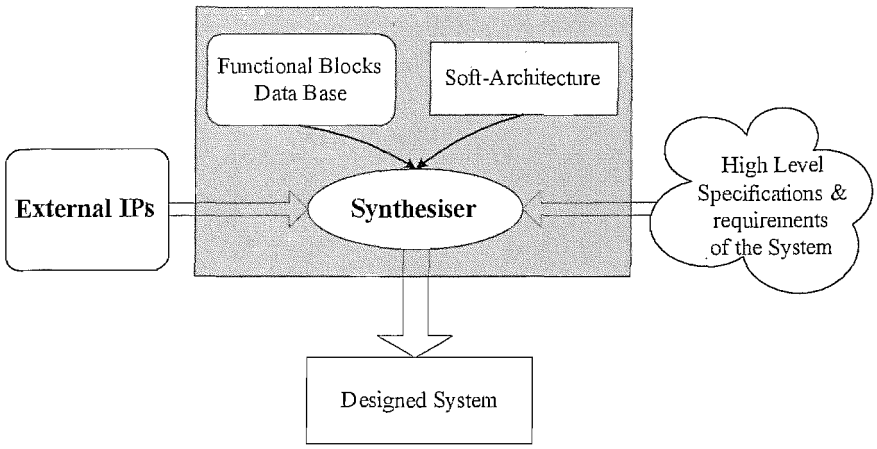


FIGURE 6.1: General description of proposed system design method.

their required parameters (area, speed, power consumption and accuracy) be modelled for different arithmetic configurations to store in this data base. This library can also be enriched by external IPs for further applications or higher performance.

One can imagine the target architecture as the general plan of a system which indicates the type and the connection of the sub-systems. This architecture, which should have the flexibility to cope with a vast variety of possible applications, consists of four subdivisions: algorithm executers, interfaces, memories and controllers, as depicted in Figure (6.2). Furthermore, it gives details of sub-systems grouping, interconnection structure, communication protocol and general aspects of the system operations. Details of the sub-systems or their functional blocks will be produced by the synthesiser according to the system specification, FBDB and external IP characteristics. Since this method works in several levels of abstraction and in each level it must choose the best selections, according to the system specifications and requirements. Design building blocks can be chose either from FBDB or external IPs to implement the target system.

The proposed design platform has been graphically explained in Figure (6.2). As can be seen, there is a database of the preliminary available subsystems which can be used in the design by fitting them into the soft-architecture. In the following sections, different parts of this platform are explained.

6.3 Target Architecture

The synthesis method proposed for computationally intensive hardware in this study is based on a combination of word-length and bus structure optimisation as discussed in chapters 4 and 5. These two problems are considered to be NP-hard, [29] and [115], which means that exact methods of optimisation are impractical. We have chosen a genetic algorithm to search the feasible space for optimum designs, but the number of

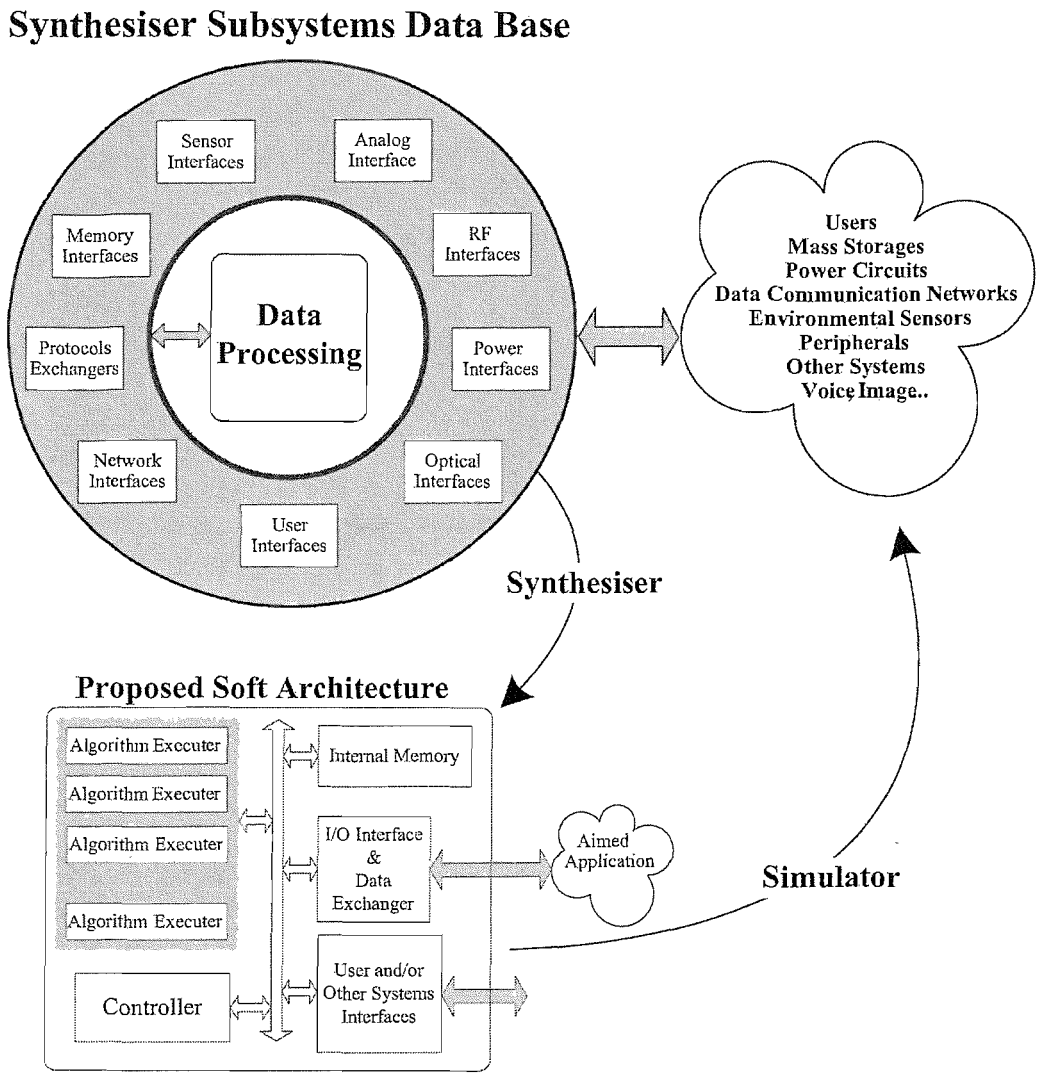


FIGURE 6.2: Structure of target hardware in the proposed design method.

all possible solutions can be very large which makes the search very time consuming. One way to reduce the number of solutions is restriction of the structure. In other words, by limiting the design to a pre-defined architecture we have reduced the size of the feasible space to a practical dimension.

The pre-defined architecture has a hierarchical structure consisting of four divisions: Algorithm Executer, Memory, Interfaces and Controller, as depicted in Figure (6.2). The algorithm executers are hardware implementation of the algorithms; the memory contains permanent information, such as coefficients and also temporary data; the interface is responsible for connecting the core hardware to the environment and controller(s) can be programmed to keep the sequence(s) of operations. This division is obviously based on the functionality of the sub-blocks to allow the synthesiser to optimise each part independently in a top-down design hierarchy.

The Algorithm Executer (AE) contains a configurable combination of Macro-Cells (MC), as depicted in Figure (6.3), in which different kinds of computational algorithms can be implemented using macro-operations provided by the macro-cells. Since these AEs should work in parallel, and possibly with interactions with each other, every AE needs to have an independent controller which is configurable, to utilise MCs, performing different sequences of macro-operations.

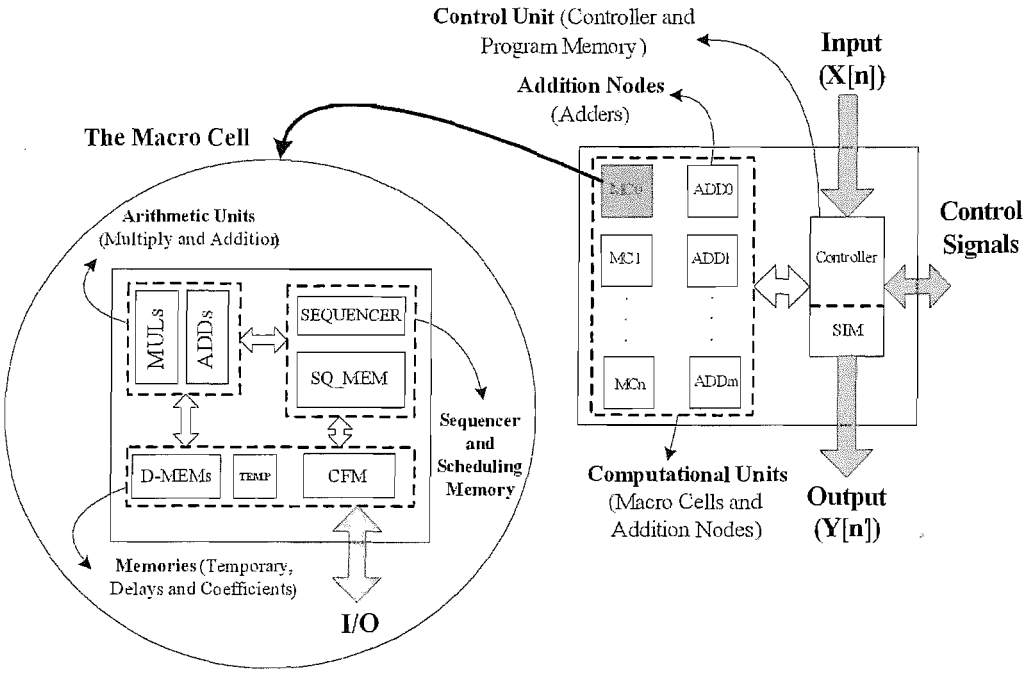


FIGURE 6.3: Algorithm Executers structure.

The MC contains the basic required Functional Units (FU) to execute the algorithms; sequencer and memories. Functional units are the finest grains in the architecture. Choosing between different arithmetic characteristics, such as the rounding method or word-length, creates different options for the synthesiser to optimise the design taking computational accuracy into account as a synthesis parameter. As a result of this flexibility, the MC can have different structures to perform an operation with different costs. In addition to the number and structure of the functional units, it is also possible to configure the sequencers unit of the MCs to perform different operations.

Another part of the architecture is the interface blocks. Every system should be connected to other information sources and destinations. In general, these systems might not use the same format of data and it means that a data interface is required for connection. To deal with these cases, interface blocks are considered to be built up from three smaller parts: External Side, Internal Side and Exchange Controller, as depicted in the Figure (6.4). The external part is dependent on the system external interconnections, thus it could cover every kind of communication circuit according to usage. This diversity of circuits must be provided by either the FBDB or external IPs. The synthesiser chooses this block according to system specifications and requirements. The internal

side transfers data to/from the system. It needs to work with the other sub-systems' connection protocol and is under the control of the exchange controller. The exchange controller manages data format which should be swapped between the internal side and external sides with coordination with the main controller.

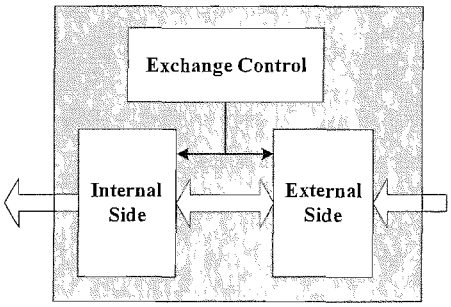


FIGURE 6.4: Interfaces basic block diagram.

Regarding the memory map of the system, two scenarios are possible: a central shared memory and distributed memory. However, bearing in mind that the AEs and MCs structure, there are enough memories in each AE to execute the algorithms, to make the architecture more flexible, a set of memories has been considered in the top level of the architecture. Furthermore, there are some kinds of applications, such as adaptive systems, for which the system needs to be configured during operation so a connection is considered between top level memory and the MC coefficients memory (CFM). Loading from memory to the CFMs can be serial and independent of other operations in the system.

In this architecture, at each level of abstraction there are controllers and sequencers to control the operations of the units and sub-blocks in that level. These controllers manage the operation of the units in that level and also exchange information with the upper level as well as sub-blocks. Nevertheless all the controllers have similar structures. These controllers construct a hierarchal controlling network in which each controller controls its sub-block controllers in master slave configurations.

There are two possible implementations for the controllers: firmware and programmable. The first one has a read-only structure, such as ROM or FSM, which saves the controlling signals for each step of operation and in which no further changes or reconfigurations are possible. The second one, which has more hardware overhead, uses instructions and rewritable memories, thus it can be reconfigured after first configuration. The first option can be implemented with lower cost, whereas the latter seems more suitable for systems with adaptive or reconfigurable operations. A simple description of both types is shown in Figure (6.5). In addition to the datapath, controllers have relationships with each other in a hierarchical master-slave structure.

From the bus structure point of view, the proposed architecture basically uses a shared bus structure, as in Figure (6.6), which has been expanded in a hierarchical structure. In

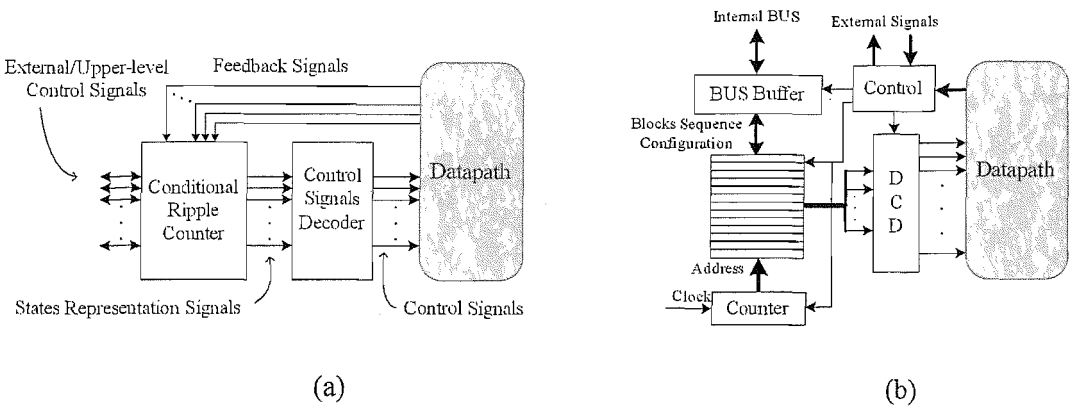


FIGURE 6.5: Controller structure a) Firmware type b) Reprogrammable type.

this structure, every level of the system has its own bus which is managed by a controller in the same block. As shown in Figure (6.6), this structure is a tree-shape expansion of the bus where each branch of the bus is a shared bus. However this structure provides an organised interconnection for data communication in the system, communicating using the shared parts of the bus becomes a serious bottleneck in the applications with high speed and/or intensive data exchanges. With due attention to this fact, in a more general form, this structure has been improved by splitting the shared parts of the single buses into several disjoint sections with controllable break points.

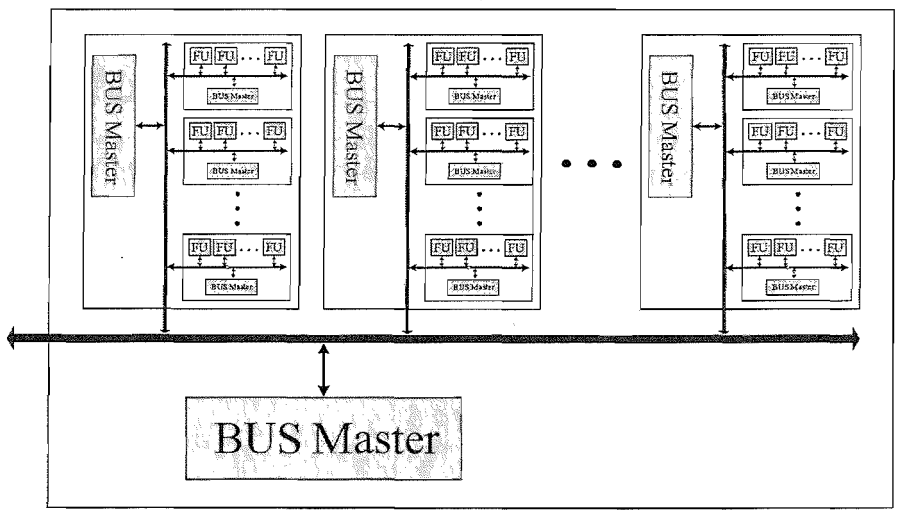


FIGURE 6.6: Bus structure of the proposed architecture from a physical connection point of view.

From a hierarchical structure viewpoint, all buffers between hierarchical levels are breaking points which are controlled by the controllers at higher levels of the abstraction. Accordingly, this idea is extended to the internal buses in each level, in which shared buses are split up and sub-blocks are categorised into groups with the most communications together. A controller at the same level manages these bus break points to make interconnections dynamic and flexible for general data transfer.

Comparing this communication structure with the MW²P-Bus, see chapter 5, it can be observed that the MW²P-Bus has the structure which is required in this architecture. Deploying the MW²P-Bus into the architecture makes it also compatible with the data transfer aware synthesis as discussed in chapter 5 and SNA method of bit-width optimisation proposed in chapter 4.

6.4 Synthesiser

In Figure (6.1) the synthesiser is a synthesis-optimisation tool whose input is a high level specification of the algorithm in a C-like difference equation format, and whose output is synthesisable RTL VHDL [141]. From a HLS point of view, this synthesiser is a restricted method in the sense that there is a pre-defined hierarchical architecture to which the system must be mapped. In addition, a set of library files are used to synthesis and optimise the design where the libraries contain basic blocks of the system and their synthesis and cost parameters (computational noise, area, power, and delay). These cost parameters can be used in a cost evaluation program after scheduling, allocation and binding to optimise the design. Figure (6.7) shows a basic block diagram of the synthesising process in the proposed method.

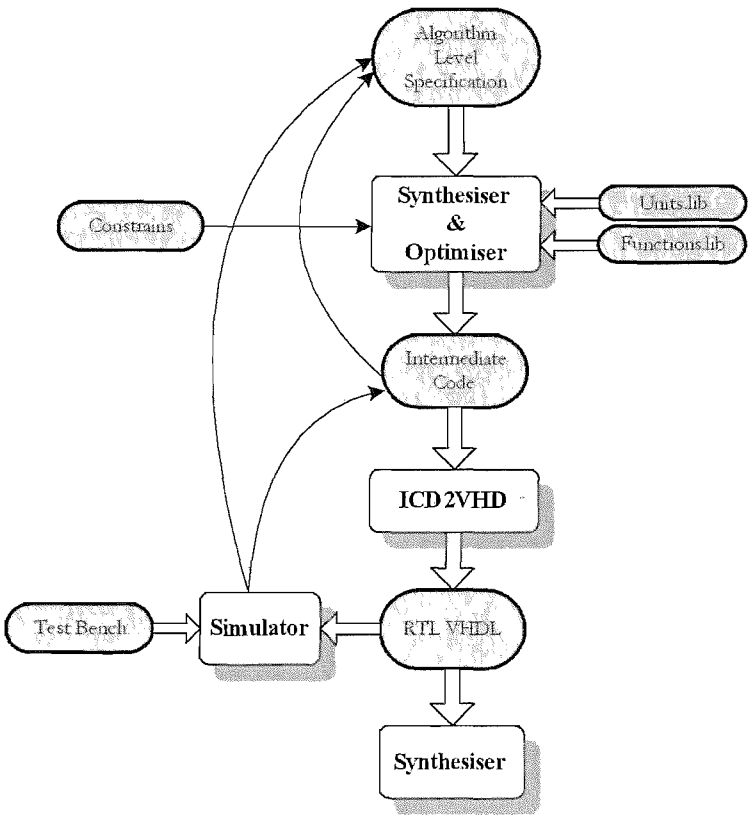


FIGURE 6.7: Synthesiser flow chart.

In this method, RTL specifications of the system are produced in two parts: first is synthesis/optimisation, whose input is the high level specification of the system which creates an intermediate specification of the system. Technically this part does the synthesis and optimisation task of the design and its output is a set of files in the format of intermediate codes (ICD). ICD files present the synthesised system in a datapath-controller structure, full details of this intermediate specification regarding the grammar and mnemonics of the ICD files are provided in appendix A. The second part of the synthesis method is the ICD2VHD program which translates the ICD files to synthesisable VHDL.

To make the operations of the synthesiser clearer, the following sub-sections present more information about the synthesiser. The optimisation task will be explained in chapter (7) and design examples are included in chapter (8).

6.4.1 Internal Representations

There are several internal data structures to represent the different stages of the system synthesis in this tool. First of all, a parser program catches the high level specification of the algorithm and passes it to other programs to extract the algorithm structure. This algorithm is translated to a directed graph (Digraph) based data structure which is implemented using the STLPLUS library [112]. Moreover, since the design must be mapped into the target architecture, the target architecture also has been converted to another data structure which is based on digraphs as well. In this way, information exchange between input specifications and the resulting design will be easier. Figure (6.8) shows a general form of the input and the output data structures of the synthesiser.

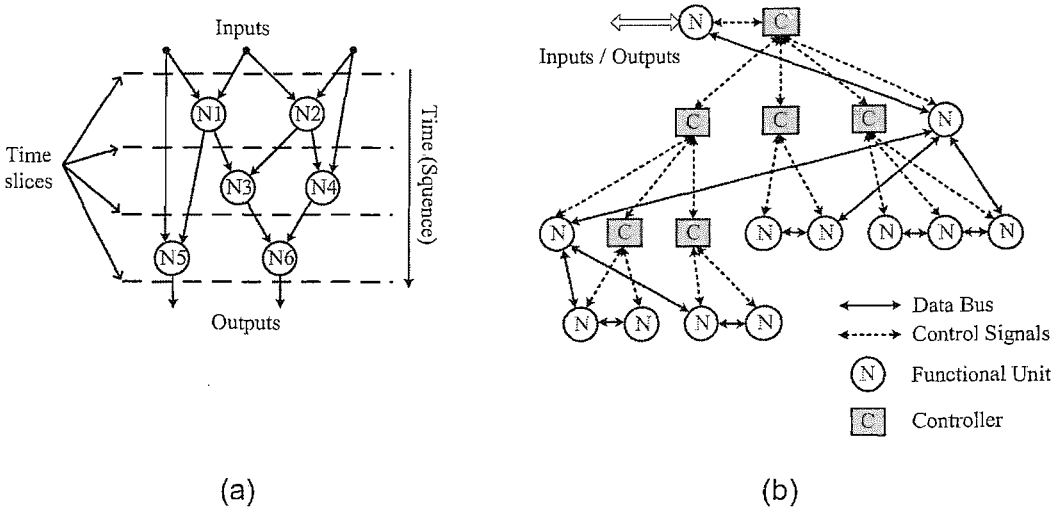


FIGURE 6.8: Synthesiser data structures a)Input digraph b)Implementation architecture digraph.

In Figure (6.8-a) a simple scheduling diagram of the input algorithm has been depicted in the form of a digraph where numbered nodes are arithmetic operations in the algorithm

and arrows represent data transfers between functional units. Figure (6.8-b) gives an example of an implemented algorithm with controllers in the form of a digraph. N-type nodes in this digraph are hardware implementations of functional units each of which could be a nested digraph and represent a sub-system. B-type nodes represent bus switches; they control the data communication between levels of architecture or functional units. C-type nodes are controllers which control the bus switches and "N" nodes. Apart from controlling datapaths, as discussed in section 6.3, controllers have a hierarchical relationship with each other which makes them a single controller in distributed form. From this point of view, this data structure is divided into two parts: a network of controllers and a network of datapaths.

The system datapath is formed from three major parts: sub-blocks, arithmetic units and glue logic. Each sub-block can be another system at the lower level of the hierarchy and contains other datapaths and controllers. But from a higher level of hierarchy, they are nodes, just like simple arithmetic units. Arithmetic units are the basic building blocks of the system and perform the required arithmetic operation. Arithmetic units originate either from the FBDB data base of the synthesiser or from the user-defined external IP data base.

Since datapaths are fully involved in the system optimisation, their representation and manipulation has a crucial accountability in the final result especially in the word-length optimisation process. Internally, datapaths are represented as a set of digraphs connected to each other and controllers.

As mentioned in section 6.3 there are two possible structures for controllers: firmware and programmable controllers. In view of the fact that many of the computational applications can be implemented by the first one, only firmware type of the controller is employed in the synthesiser, however by some modifications implementation of the programmable controller is possible as well.

There are two subjects of interest regarding controllers in this design platform: controller structure and controller interaction with each other and sub-systems. Essentially, controllers are finite state machines (FSM) which continuously send out activation signals to manage the datapath behaviour. Each unit, depending on its structure, receives at least one activation signal. Some of the units or sub-systems are more complicated and apart from their incoming control signals, they might have outgoing signals for a handshaking connection. The controller gets these handshaking signals as the internal controlling signals.

Consistent with the target architecture, controllers build a tree network in which a master slave relationship between them is established. This relationship is created by a set of handshaking signals between controllers in different levels of the hierarchy, see Figure (6.8-b). These signals are called external controlling feedback signals. The implementation of the controllers is based on the ICD file format which is very similar

to register transfer signals tables (similar to Tables in appendix B). ICD codes provide for controllers two different configurations which we call Structured by External Control Signals and Unstructured. Figure (6.9) shows these two configurations.

Generally, controllers are state machines which are under control of the both internal and external controlling signals. Internal control signals are feedback or handshaking signals from sub-blocks and external control signals are handshaking signals from higher level of abstraction in the architecture; see Figure (6.5). According to how these internal and external controlling signals affect the state machine sequence, these two configurations are defined in Figure (6.9-a) and (6.9-b).

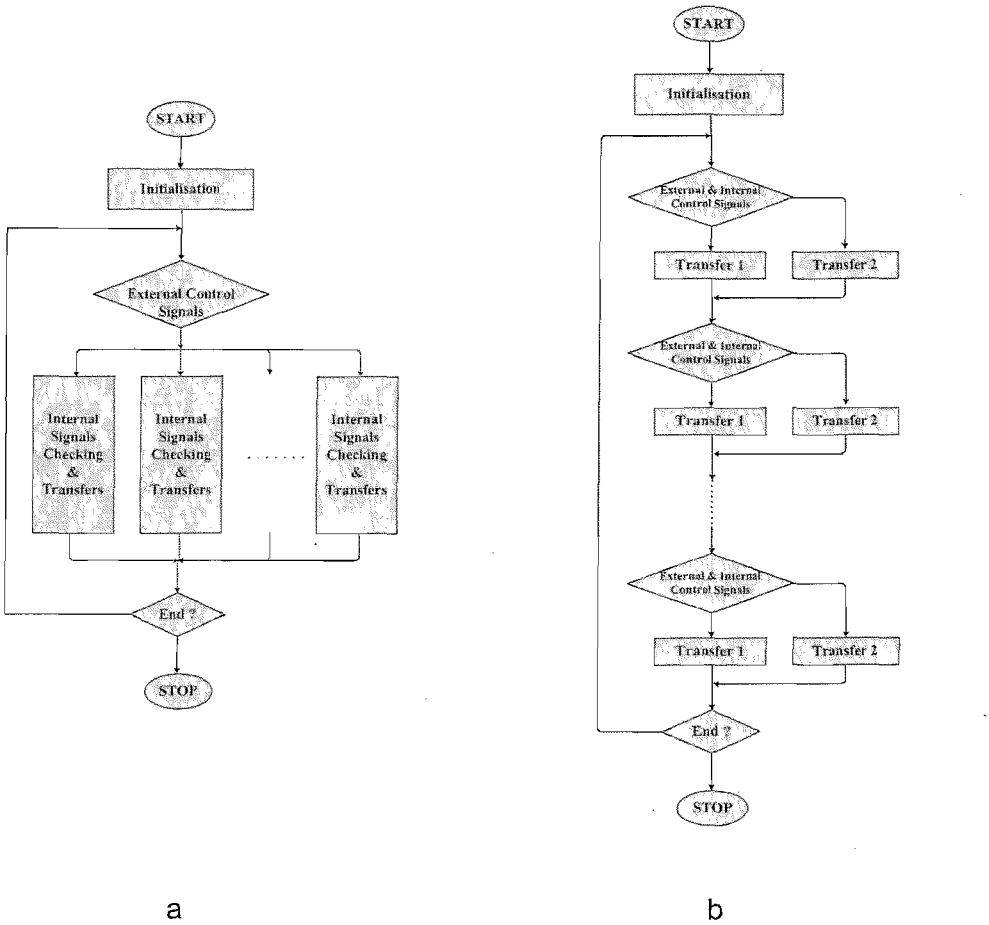


FIGURE 6.9: Controller Structures a)Structured based on external control signals b)Unstructured.

In Figure (6.9-a) before starting the main loop, external control signals are checked and according to their values a set of operations will execute. This structure is suitable for applications or sub-systems which must give full authority to the higher level of the hierarchy to limit their operations in some circumstances. Alternatively, the second structure, Figure (6.9-b), is more flexible in that in each state decision depends on the all internal and external controlling signals.

6.4.2 Functional Blocks Data Base

The synthesiser and optimiser use library files for design implementation and Optimisation. Two library files are created and used in this study: FUNCTIONS and UNITS. From a synthesis point of view, FUNCTIONS gives the functionality information of the basic blocks like: unit type, operation and signalling requirements. UNITS, on the other hand, gives information about the implementation and optimisation of the functional units. List (6-2) and list (6-3) give samples of these libraries.

List 6.2: FUNCTION.lib library sample file.

```
// Basic Blocks FUNCTION library: (sample)

OPERATIONS:
{
BUS  =BSW2 IN:[0;d;BUS] OUT:[1;d;BUS]
BB   =BSW1 SP:[0;d;d]
*    = MUL  IN1:[1;d;BUS] IN2:[0;d;BUS] OUT:[2;0;BUS]
+    = ADS  IN1:[1;d;BUS] IN2:[0;d;BUS] OUT:[2;d;BUS]
-    = ADS  IN1:[1;d;BUS] IN2:[0,3;d;BUS] OUT:[2;d;BUS]
REG= reg  IN:[0;d;BUS] OUT:[1;d;BUS]
}
```

List 6.3: UNITS.lib library sample file.

```
// Basic Blocks UNIT library:
(sample)

UNITS:
{
BSW1 01 00 A= 1 D= 1 N= 1 P= 1
BSW2 02 00 A= 2 D= 1 N= 1 P= 1
MUL 03 01 A= 4 D= 1 N= 1 P= 1
ADS 04 00 A= 4 D= 1 N= 1 P= 1
reg 02 00 A= 1 D= 1 N= 1 P= 1
rom 10 00 A= 1 D= 1 N= 1 P= 1
}
A: area     P: Power dissipation
N: noise    D: delay
```

6.5 Summary

This chapter provided a comprehensive explanation of the architectural implementation of the design. A bus-oriented structure is proposed which utilises the concept of the MW²P-Bus in the form of a soft-architecture as target structure for the synthesiser. This architecture consists of several hierarchical levels in which macro-cells are connected to a central MW²P-Bus to implement the specified algorithm. This set of macro-cells and MW²P-Bus is controlled by a controller which can be implemented in the form of a single controller or a distributed network of controllers.

This architecture is used as the target structure for optimisation to reduce the size of the feasible space and consequently increase the optimisation speed. The next chapter provides details of the optimisation method for this synthesiser and the corresponding algorithms.

Chapter 7

Optimisation Algorithm

7.1 Introduction

In mathematics, optimisation is the discipline which is concerned with finding the maxima and minima of functions, possibly subject to constraints. In most realistic optimisation problems, particularly those in design, a simultaneous optimisation of more than one objective function is required. It is unlikely that the different objectives would be optimised by the same choice of alternative parameters. Hence, some trade-off between the criteria is needed to ensure a satisfactory design optimisation. A large number of problems in engineering need to be formulated as a global optimisation problem and in general, global optimisation problems can be very difficult to solve [33].

Many different ways have been proposed and investigated in a variety of applications to solve a general or special case of optimisation problems. From one viewpoint, these approaches can be separated into two categories: exact methods and stochastic searches. In the case of problems with NP complexity, exact methods soon become computational intractable and are only available for a subclass of optimisation problems [46]. In contrast, there are possibilities for a particular class of problems to employ heuristic methods with practical optimisation times.

High level synthesis is concerned with the automatic generation and allocation of functional units, registers, buses, controllers and glue logic from an abstract specification of the system, see chapter 2. Always, there are many different "correct" options for a synthesiser to map a given specification, so choosing the best, or at least a good, solution among all possible cases is considered as an optimisation problem. High level synthesis, in general, involves at least two NP-complete optimisation problems. The first is scheduling in which the operations given by the high level specification are assigned a control step. The second is allocation which assigns functional units to operations given in the high level description [30]. In the worst case, a synthesis task may require solving

an exponential sub-problem at each step of the main problem. Therefore, there has been a trend toward developing efficient heuristic algorithms which avoid certain locally optimal solutions.

Such an optimisation should take into account several design quality constraints. Area, delay and power consumption are most often considered as quality measures in this regard. One usual optimisation method is to search inside the space of all acceptable designs, which is called the feasible solutions space, to find the optimal solution. From one point of view, the design space can be defined as multidimensional space spanned by different characteristics of design. Every point in this space represents a possible implementation.

In this study four basic objectives are considered for optimisation: area, power consumption, latency and output digital noise, as the accuracy of the system is considered as the quality measure of the design. In accordance with this assumption, the cost functions are investigated to provide suitable models and cost functions for these objectives which relate them to the word-length of the design units as a controlling parameter. A multiple objective optimisation is employed to combine these cost functions in a Genetic method.

7.2 Method Overview

Principally, optimisation can be considered as a process of selecting a set of "parameters" for the system to make "some aspects", called objectives, of it work more "efficiently". In other words, there are a set of parameters which control the objectives and optimisation searches for the available values of the controlling parameters which give the best (or sometimes better) efficiency. The definition of the controlling parameters and the inspecting objectives are dependent on the nature of the under investigation problem. In high level synthesis there are a set of well known parameters by which optimisers find the optimum solutions for problems [33]. In this study, these parameters are the HLS parameters, as explained in chapter 2, word-length (see chapter 4) of the functional units and the MW^{2}P-BUS structure (as explained in chapter 5).

Since the HLS objectives are closely dependent on the new proposed parameters (MW^{2}P-bus structure and word-length) design and optimisation with the combination of them requires a new optimisation approach. In this study a genetic algorithm is chosen to find an optimal solution, then binding and allocation procedures are merged into the bus partitioning and word-length allocation in the form of the gene codes to integrate all the objectives in one problem.

It is shown in chapters 2 and 5 that bus partitioning is a very effective method to improve the communication speed in bus-oriented synthesis methods. The proposed method in this work is an extension of segmented bus structures into more complex structures as

well as multiple-width concept. This flexibility and extendibility provides a wide range of opportunities for trade-offs between system costs and performances. Consequently, considering bus structure as an optimisation parameter, in terms of the number of bus segments, bus switches and wire-length, is practicable.

The majority of the high level synthesis methods consider arithmetic units as atomic sub-blocks which perform operations with certain accuracy and a pre-known implementation costs. Regarding recent studies and our discussion in chapters 3 and 4 it is generally accepted that this assumption is not necessarily valid. Accordingly, word-length can be considered as an optimisation parameter which has a considerable impact on the implementation costs as well as computation accuracy. Particularly, in the systems with computationally intensive datapaths, the word-length as an optimisation parameter has a more significant role.

The first step to take these new parameters (word-length and bus structure) into account in the optimisation is: finding the relationships between design objectives and the new parameters. In this study, circuit area, power consumption, latency and accuracy are considered as design objectives, which need to be investigated in terms of their relationship with word-length of the functional units and the bus structure.

To combine all the objectives with the word-length and MW^{2}P-BUS structure, a stochastic search method is proposed in the form of vector evaluating genetic algorithm [34]. Controlling parameters are coded in a gene which consist all the required information for evaluation of the effect of the different parameters on the optimisation objectives.

In the following subsection more details of the optimisation method, extracted cost functions and the corresponding algorithms are discussed.

7.3 Cost Functions

The costs of the design can be divided into three parts: those of datapaths; controllers; and interconnections. Since the design space is extended by word-length and bus partitioning here, the effect of these two new costs must be evaluated on each part individually. The controller part is not dependent on the word-length or system bus partitioning and so it should be considered as a constant value in the cost function but the effect on the two other parts must be investigated. Having focused on word-length, it is shown in [16] and [28] that accuracy, area and power consumption costs are dramatically dependent on the word-length and execution delay is a function of the word-length in the case of functional units whose operation is based on sequential bit operations (for example sequential multipliers). Bus partitioning, on the other hand, influences costs by adding bus switches and their control wires. The cost model is as:

$$F_{Total}(\vec{X}) = F_{Controller} + F_{Interconnect}(\vec{X}) + F_{Datapath}(\vec{X}), \tag{7.1}$$

where F is the cost function and $\overrightarrow{X}$ is the set of MW^{2}P-Bus parameters. An important point, which needs to be reiterated here, is that the proposed cost models are functions to evaluate different designs during optimisation, which means their ability to map feasible design space individuals into a set of distinct cost values, are more important than their precision. In the following subsections, descriptions of the cost models are presented. All the relations and values are based on basic cells in the ST 0.12 μm technology using the Synopsys tools.

7.3.1 Delay Cost Function

The execution delay of the design is evaluated from the number of C-steps in the schedule. Basically, the delay is a nonlinear function of parameters such as: the types of available functional units in the synthesiser library; functional unit grouping; and the word-length assigned to each group. From these, an evaluation function is implemented in the optimiser which estimates the delay value (for each produced design, genome, in the genetic algorithm) using a Resource Constrained (RC) List Scheduling algorithm [33]. The basic delay for registers, bus switches and combinatorial functional units is considered to be one clock cycle, whereas the delay for a sequential functional unit (a Booth multiplier, for instance [100, 35]) is dependent on the word-length of the functional unit. In the case of the Booth multiplier, which is one of the functional units considered, latency is linearly dependent on word-length.

Another impact of word-length manipulation which requires more attention here is its effect on the maximum delay of the functional unit units which might affect the working frequency of the system. In basic scheduling without chaining or multicycling (see [33]), the slowest functional unit dictates the maximum execution delay in a clock cycle, in addition, the latency of the functional units are dependent on their word-length, thus there is a relationship between word-length and maximum frequency (however, selection of system frequency is dependent on many different parameters ranging from implementation technology, device design, system dependency on its peripherals and so on, but word-length should be considered as a parameter too).

7.3.2 Area Cost Function

Design area and its dependency on the word-length of the functional units is estimated for all the functional units, cells and glue logic and is approximately linear with respect to word-length, except for combinatorial multipliers which exhibit a second order dependency. In addition, the impact of bus partitioning on the area can be approximated by the bus switch area, and thus the area cost function can be derived as Equation (7.2),

$$F_A(\overrightarrow{X}) = A_C + A_{FU}(\overrightarrow{X}) + A_B + A_{BS}(\overrightarrow{X}), \tag{7.2}$$

where $F_A(\vec{X})$ is the total area cost, A_C is the controller area, $A_{FU}(\vec{X})$ is the datapath area, A_B is the interconnection area and $A_{BS}(\vec{X})$ is the bus switch area.

The area of building blocks such as sequential multipliers, adders, registers, buffers and switches can be assumed to have a proportional relationship to word length while the area of a combinatorial multiplier can be modelled by a second order relationship with its word length. Design implementation results confirm this assumption as depicted in Figure (7.1).

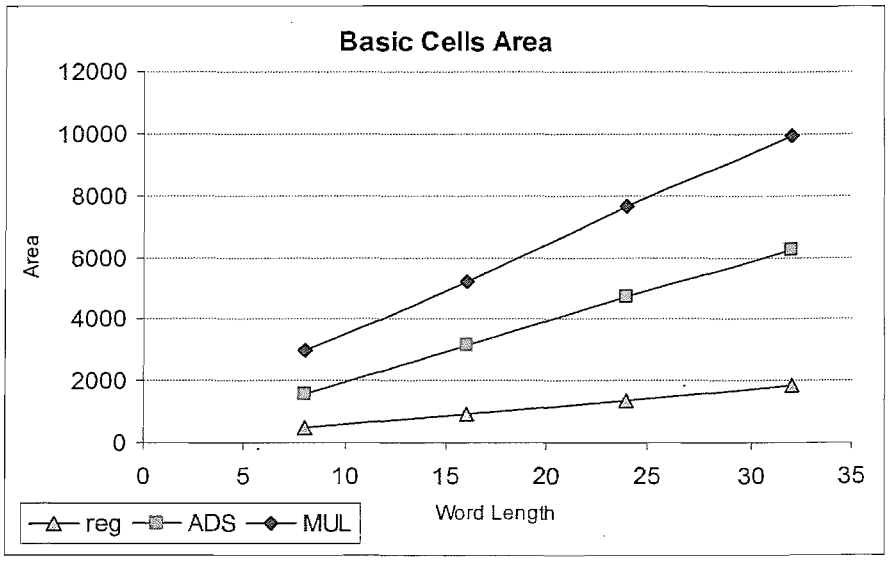


FIGURE 7.1: Dependency of area on word length for basic cells (Registers, Adder and Sequential Multiplier).

7.3.3 Power Consumption Cost Function

To evaluate the system power consumption, a model is required that includes dynamic and static power consumption for all the sub-blocks, interconnections, I/Os, drivers and controllers [108]. Knowing that changing the word-length of the functional units does not affect the controller activity and structure, the power consumption of controllers (P_C) is a fixed term in the estimated power consumption. In addition, because of using MW^{2}P-Bus approach in this study, which is closely related to the word-length of the functional units and the number of wires, interconnection power consumption (P_B) depends on the word-length of the shared bus segments; therefore, ignoring the P_B dependency on w is not acceptable at this level of abstraction. On the other hand, including approximation models of interconnection power consumption is too complex to be combined with the model of functional units' power consumption in a practical optimisation method. To cope with this difficulty, we assume that the wiring complexity of the system bus can be separated from the rest of the interconnections, which means that the cost model of the MW^{2}P-Bus can be isolated from the rest of the system. Having this assumption,

P_B only refers to the interconnections power consumption without data buses, thus the power consumption model is approximated as in Equation (7.3). The MW^{2}P-Bus power consumption model is also discussed in section 7.3.5.

$$F_P(\overrightarrow{X}) \approx P_{FU}(\overrightarrow{X}) + P_B + P_C, \tag{7.3}$$

A set of designs is used to evaluate the functional unit dependency on word-length and the results are presented in Figure (7.2). In this figure, the average power consumptions for the basic cells in the library (Register, Adders/Subtractor and Multiplier), with random input data, is shown with respect to word-length. In these simulations, the "Nominal Low Leakage" ST 0.12μm technology file is used. From this, we can see that power consumption has approximately a linear dependency to the word-length. On the other hand, power consumption is a combination of static and dynamic parts; accordingly, in each functional unit it is a sum of static and dynamic parts as in Equation (7.4).

$$P_k = P_{k,Dynamic} + P_{k,Static}, \tag{7.4}$$

Here P_k is the power consumption of the k$^{\text{th}}$ functional unit.

In general, dynamic and static power consumption are data dependent [78] but in this study, to estimate power consumption in the optimisation procedure, static power consumption is considered proportional to the total power, Equation (7.5).

$$P_{k,Static} = \lambda_k \cdot P_k, \tag{7.5}$$

where λ_k is the leakage power factor. Simulations verify this assumption for basic blocks for different word lengths.

Another assumption used to reduce the evaluation complexity is a time slot approximation [138]. In this approximation the total power consumption of a functional unit is calculated in two parts: "activation" time slots and "standby" time slots. During functional operation, power consumption is the sum of dynamic and static power whereas in standby, only the leakage power is taken in account. Based on this approximation, the total power consumption for each functional unit is given in Equation (7.6).

$$\begin{aligned} F_P(\overrightarrow{X}) &= \frac{1}{T}\sum_{k=1}^{N_F+N_B} w_k \cdot (t_k p_k + (T - t_k) \cdot p_k \cdot \lambda_k), \\ &= \frac{1}{T}\sum_{k=1}^{N_F+N_B} w_k \cdot p_k \cdot (t_k + \lambda_k \cdot (T - t_k)), \end{aligned} \tag{7.6}$$

where $F_P(\overrightarrow{X})$ is the average power consumption of the system, p_k is the average power per bit of the k$^{\text{th}}$ functional unit or bus switch, w_i is the word-length assigned to the group which contains the k$^{\text{th}}$ functional unit or the bus segment which is connected to

the bus switch, t_k is the activation period of the kth functional unit or bus switch, λ_k is the leakage power factor, T is the total system operation time (latency), N_F is the number of functional units and N_B is the number of bus sections.

To verify this equation, a set of simulations are performed using cell based implementation of the functional units with ST0.12μm technology. First, functional units are synthesised using Mentor Graphics tool, Leonardo, then the result Verilog specification is simulated over a relatively long period of time (10 second) with random number inputs by ModelSim simulator, to extract the switching activity. Afterwards, these information with ST technology files for nominal design are used with the Synopsys Primepower tool to estimate the average power consumption over the simulation period.

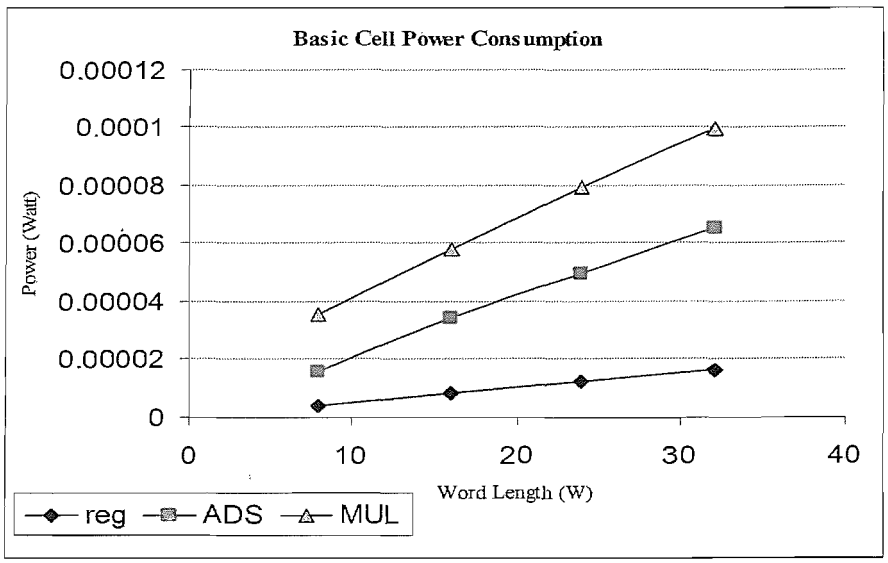


FIGURE 7.2: Dependency of area on word length for basic cells (Registers, Adder and Multiplier).

7.3.4 Noise Cost Function

In SNA method, as explained in chapter 4, variable $\hat{x}$ is initially represented as in Equation(7.7).

$$\hat{x} = F_x(\vec{E}), \tag{7.7}$$

where $F_x(\cdot)$ is a fraction of polynomials with M known coefficients $(x_1, x_2, \cdots, x_M)$; and $\vec{E}$ is an array as in Equation (7.8).

$$\vec{E} = [\epsilon_1, \epsilon_2, \cdots, \epsilon_m], \tag{7.8}$$

where ϵ_i are symbolic representations of random values.

Initially, noise symbols are considered to have a uniform PDF model that is presented in [99] and then improved in [28] which is the commonly accepted model in the multiple

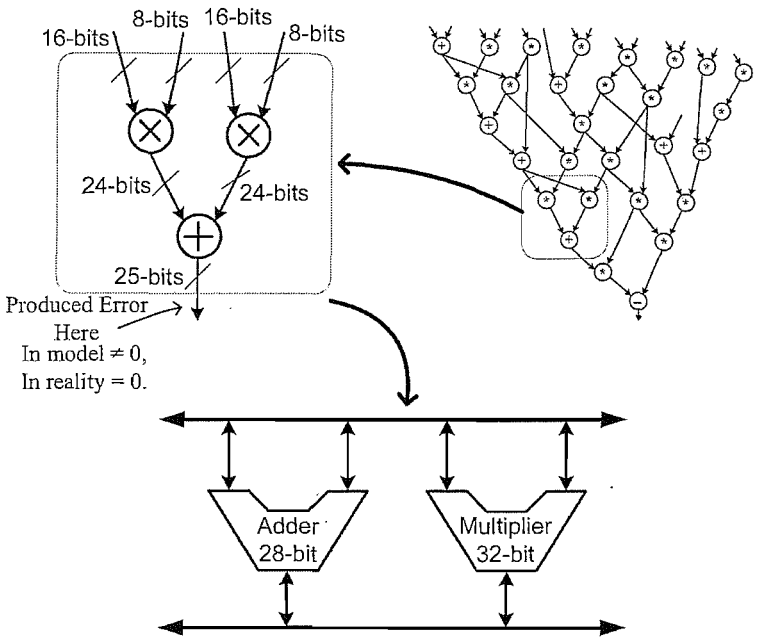


FIGURE 7.3: Mapping a multiple-WL DFG to hardware which shares resource.

word-length paradigm. Noise cost can be considered to be error range and/or variance. With this initial assumption, since the PDF is uniform over $[-1, +1]$, the variance (σ_k) of the error is:

$$\sigma_k^2 = \frac{2^{2p}}{12}(2^{2n_2} - 2^{2n_1}), \tag{7.9}$$

where p represents the decimal point position, n_2 represents the word-length of the previous node and n_1 represents the word-length of the next node. According to this model, the values of noise sources are specified by the word-length of the current FU and its preceding (parent) node(s). Equation (7.9) offers a simple model which is mostly correct; however it can be misleading in some cases, especially in stochastic search methods.

Consider an example, as depicted in Figure (7.3), which shows the general maximum required word-length in a small section of a DFG. In the figure, the annotated word-lengths on the inputs of the nodes in the DFG, are assumed that has been inherited from the parent nodes in DFG and intermediate word-length are calculated based on the input word-lengths and the operation types. Let us assume that this DFG is implemented on resources shared hardware, as shown in the figure, which only can provide a 24-bit adder and a 32-bit multiplier for this section of the DFG. Now, since multiplications, which need only 24-bits on the DFG, are implemented on the 32-bit multipliers in the adder node if 32-bit is considered as the input word-length, the adder output should have error, which is not. Accordingly, noise sources in the model in [28] must be considered to be dependent on all the prior nodes in the DFG, instead of only previous nodes, or the word-length of the implemented hardware.

This simple example suggests that in the noise source evaluation in Equation (7.9), the word-length for every node in the DFG must be calculated from data range propagation through all its preceding parent nodes. Especially in stochastic search methods or HLS integrated methods this data range analysis must be repeated at every iteration. To deal with this problem, for the GA-based method we suggest an ordered production and evaluation of the word-lengths in the W-vector of the genome which considers dependency between allocated word-lengths in the genome vectors.

In our optimisation method, the basic cost function of computational noise is as in Equation (7.9) but n_1 and n_2 are calculated with aforementioned considerations. In more accurate design evaluations, error range and error PDF of the computational circuit are calculated using SNA method.

7.3.5 MW^{2}P-Bus Cost Function

The proposed bus structure has a fairly complicated interconnection which increases the interconnection cost. To evaluate the design efficiency, costs of the bus are considered independently alongside the other costs of the datapath.

Since the general structure of the MWP-BUS, which is similar to a single shared bus, is not dependent on the splitting point, except for added bus switches, thus its area can be considered as a constant value. But MW^{2}P-Bus is more complicated and to simplify the problem here, the area of the MW^{2}P-BUS is considered linearly related to its wire-length. According to the results of section 5.4 an approximation for the wire length of the MW^{2}P-Bus is in Equation (7.10).

$$L_B \approx A \cdot \frac{M^3 + 3M^2 - 4M - 18}{6} \cdot \overline{w}, \tag{7.10}$$

where A, as explained in section 5.4, is a constant value and does not affect our comparative results during optimisation.

By employing the proposed method in [56] and [55] the average energy consumption of the split-bus architecture per clock cycle is calculated as in Equation (7.11).

$$\begin{aligned} E_{B,P} \;=\; & \frac{1}{2} \cdot \alpha_{SW} \cdot V_{dd}^2 \cdot \left\{ \sum_{k=1}^{N_B} \left[C_{Bk} \sum_{i \in BUS_k} \sum_{j \in BUS_k, i \neq j} xfer(M_i, M_j) \right] \right. \\ & \left. + \sum_{k=1}^{N_B} \sum_{l=1}^{N_B} \left[C_{P(k,l)} \sum_{i \in BUS_k} \sum_{j \in BUS_k} xfer(M_i, M_j) \right] \right\}, \end{aligned} \tag{7.11}$$

where α_{SW} is the average *Switching Activity*, N_B is number of buses, C_{Bk} is equivalent capacitance of the k$^{\text{th}}$ bus, $C_{P(k,l)}$ is equivalent capacitance of the path from bus k to l, M_i is functional unit i and $xfer(M_i, M_j)$ is the probability of data transfer from module

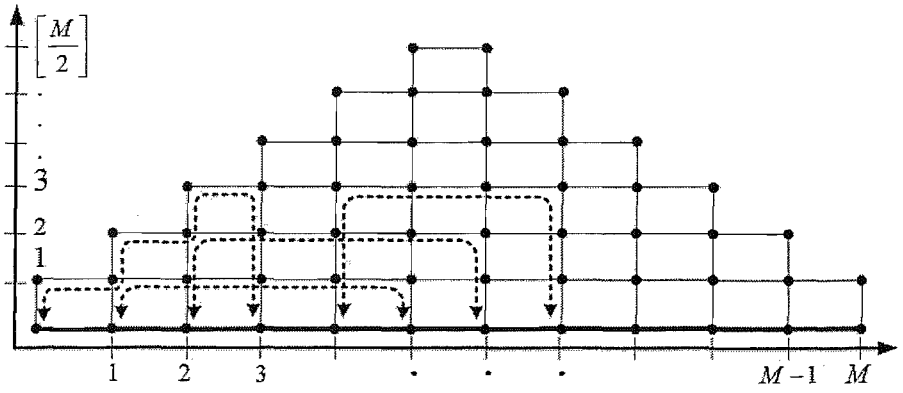


FIGURE 7.4: MW^{2}P-Bus provides a dynamic connection path.

M_i to module M_j in any clock cycle. This method is more suitable for implementations which consist of submodules which have a stochastic communication scheme.

Assume that the receiver gate for each module is at its minimum size and its input capacitance is C_g. Furthermore, the output capacitance of the driver for each module M_i is $C_{o,i}$, C_{Bk}, calculated as in Equation (7.12).

$$\begin{aligned} C_{Bk} &= w_k \cdot L_{Bk} \cdot (C_u + C_c) + \sum_{i=1}^{N_k} w_k \cdot (C_{o,i} + C_g), \\ &= w_k \cdot (L_{Bk} \cdot (C_u + C_c) + N_k \cdot (C_o + C_g)), \\ &= w_k \cdot (L_{Bk} \cdot C_{1k} + N_k \cdot C_{2,k}), \end{aligned} \tag{7.12}$$

where L_{Bk} is the physical length of the k$^{\text{th}}$ bus, C_u denotes the capacitance per unit length of the bus wires, C_c denotes the coupling capacitance per unit length of the bus due to the parallel running bus wires as well as other nearby wires on adjacent metal layers, N_k is the number of FUs connected to the kth bus and w_k is their WL. In Equation (7.12), these parameters are abstracted in $C_{1,k}$and $C_{2,k}$ for simplicity.

In contrast, by revising the data communication structure which is proposed here, the datapath has a dynamic data communication architecture in that there are no fixed wired connections between functional units during their operation , as depicted in Figure (7.4). Thus, unlike Equation (7.11) the characteristics of the interconnection change dynamically during datapath operation. Basically, these characteristics are dependent on the source and destination nodes and the available path(s) in the MW^{2}P-Bus for connection at that time. In general these parameters are functions of datapath HLS objectives, constraints and also the structure of the MW^{2}P-Bus and number of its switches and segments. Each connection path is made of bus segments in the selected path from source to destination. Accordingly, the i$^{\text{th}}$ path capacitance ($C_{path,i}$) can be calculated

by adding up its bus segments as in Equation (7.13).

$$C_{path,i} = \sum_{k=1}^{N_i} C_{Bk}, \tag{7.13}$$

where C_{Bk} is the capacitance of the kth bus segment in the ith path. By employing the proposed method and Equations (7.12) and (7.13), the average energy consumption of the MW^{2}P-Bus per clock cycle ($\overline{E}_B$) is calculated as in Equation (7.14).

$$\begin{aligned} \overline{E}_B &= \frac{1}{2}\alpha_{SW} V_{dd}^2 \sum_{i=1}^{N_P} C_{path,i}, \\ &= \frac{1}{2}\alpha_{SW} V_{dd}^2 \sum_{i=1}^{N_P} \sum_{k=1}^{N_i} w_k (L_{Bk} C_{1,k} + N_k C_{2,k}) \end{aligned} \tag{7.14}$$

where α_{SW} is the average switching activity, N_P is the number of paths and V_{dd} is the power supply voltage. In this study, this power consumption is related to the length of bus segments as well as their width (WL).

7.4 Optimisation Algorithm

The implemented synthesiser employs an optimisation method which is based on a Genetic Algorithm (GA). This method tries to match functional units to the target architecture with optimum word-lengths and MW^{2}P-Bus structure. In general, a genetic algorithm is a heuristic stochastic search algorithm for the solution of optimisation problems. Starting from a random initial guess solution (called the first generation), better descendants are tried in an attempt to find one that is the best under some criteria and conditions. It is based on the idea of evolutionary theory, that individuals having a high value of quality will survive to the next generation with greater probability [49].

The utilised genetic operators (including weighted roulette wheel, crossovers and mutation [34]) are extracted from a standard GA procedure for variable length, integer array genomes. The resultant genes represent the number and word-length of each functional unit in the datapath. It must be noted that the gene's length might be different for individuals in every generation. The synthesiser tool employs an Elite-Preserving, Vector Evaluated Genetic Algorithm (VEGA) optimisation with a fitness function of a Weighted Chebyshev combination of the basic datapath costs (area, delay, energy and noise) and bus costs (area and power consumption) to find the optimal points in the constrained feasible space [34]. After assignment of the nodes of the DFG to word-lengths of functional units, resource constrained allocation and scheduling [33] is applied to evaluate each individual's fitness.

In the case of a single shared bus without segmentation, the genome of the individuals in the optimisation procedure is as depicted in Figure (7.5). This figure represents an array of integers in which the first part of the array gives the number of the functional units that are used in the design (binding result for functional units). Afterwards, word-lengths which are allocated to each functional unit are represented. It can be observe that the length of the genes are variable, for example in the case which is shown in Figure (7.5) the array length is $N_{FU} = M + A$. The proposed optimisation algorithm for the single shared bus without segmentation is shown in Listing (7.1).

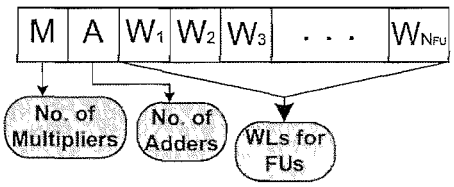


FIGURE 7.5: Genome structure for GA optimiser.

Listing 7.1: The synthesiser algorithm for non-partitioned bus

```
INPUT: Data Flow Graph;
OUTPUT: Synthesisable RTL-VHDL;
Objectives: Minimising the area, power consumption;
        and delay and maximising the accuracy;
BEGIN
  Initialise the DFG using ASAP;
  Find the maximum required FUs for each type
  Find the Bound solutions for each cost;
  Find the the uniform WL for all FUs which satisfies
     the accuracy constrain;
  First Generation;
  FOR N iterations;
     FOR all the population;
        Find Area();
        Find Static Energy Consumption;
        Do List-Scheduling();
        Find the C-Steps
        Find Delay();
        Evaluate Dynamic Energy Consumption;
        Evaluate Digital Noise in the output;
        Find Power();
        IF (a cost is out of its constraint range) THEN
            modify its corresponding weight coefficient
            in the combined cost function;
```

```
        Find the overall combination cost;
      END FOR;
      Produce the next generation;
   END FOR;
END;
```

In algorithm of Listing (7.1), the input is assumed to be represented in the form of data flow graph (DFG), where the output is in the form of intermediate representation (ICD) as explained in section 6.4, which can be translated to synthesisable RTL-VHDL. The objectives of the optimisation algorithm are considered to be area, power consumption, latency and accuracy of the design with the cost functions as explained in section 7.3. In the first step, the algorithm finds the maximum required number of functional units for implementation of the input DFG with applying ASAP scheduling algorithm. See section 2.2 for more details regarding this scheduling algorithm. Then maximum and minimum bounds of the each cost function are calculated to be used in the normalisation of the costs in the fitness function of Weighted Chebyshev combinated cost as shown in Equation (7.15) [22].

$$F(\vec{W}) = \frac{K_A \cdot F_A(\vec{W}) + K_P \cdot F_P(\vec{W}) + K_N \cdot F_N(\vec{W}) + K_D \cdot F_D(\vec{W})}{K_A + K_P + K_N + K_D}, \tag{7.15}$$

where $\bar{F}_A$, $\bar{F}_P$, $\bar{F}_N$ and $\bar{F}_D$ are normalised cost functions for area, power consumption, digital noise and delay respectively and K_A, K_P, K_N and K_D are constant weighting factors for costs such that increasing one of them against the others increases the importance of the corresponding cost in the optimisation process.

The genetic algorithm starts with a first generation which is produced randomly, based on the genome structure of the Figure (7.5). Afterward, in each iteration of the algorithm new genomes are produced based on choosing the best individuals to produce more offspring [34]. After assignment of the nodes of the DFG to functional units, a resource constrained (RC) allocation and scheduling [33] is applied to evaluate each individual's fitness. Population size, number of iterations, percentage of crossover and mutation and their probabilities are chosen regarding the maximum required functional units in the design.

From this experience, a genetic search does not converge in a reasonable time for complicated designs (more than 50 nodes in the DFG) because of the size of the feasible space ($> 10^{54}$ assuming that the maximum allowed word-length is 32). Thus a biased generation of the individuals is used to speed up the GA optimisation. Accordingly, before the optimisation search, a design with a uniformly chosen word-length is found that has the closest costs to the constraint values, since the optimum results are most likely to be found close to this point, the GA search for optimal points is performed

around this preliminary (bias) point. However this guided search might lose some optimal points in separated islands in the feasible space. The results show that it satisfies the implementation requirements.

In practice, a non-constrained optimisation is a powerful tool in those cases where a trade off is possible between all design costs, but in most practical implementations, there are previously known constraints which must be satisfied. Comparison of our results as well as inspection on the cost functions, suggests that by freezing one of the costs and taking it as a design constraint during synthesis, it is possible to achieve the same required objective with minimum costs for the others. To illustrate this issue, a set of constrained optimisations were performed in which a static penalty method [23] is employed to find the optimal points. In the algorithm of Listing (7.1), the weight coefficients of the combined cost function of Equation (7.15) are adaptively checked in the case of user restricted optimisation (indicated as the **IF** () instruction) to keep the restricted costs in the range defined by the user. It means that if a cost falls out of the restricted zone, the algorithm will increase the corresponding weight factor for the combinational cost evaluation until it is dragged back into range.

This method can be modified to be used for the MW^{2}P-Bus structure. In Figure (7.6) the general format of the genes in this case is depicted. Every gene has sections which are (from left to right in Figure (7.6): DFG assignment to the groups; number of the groups; number of FU0 (multiplier for instance) in each group; number of FU1 (ALU for instance) in each group and so on for other functional unit types and the last section is the word-length of each bus segment. Similar to the non partitioned bus, in this genome values are also integers and the gene length is variable as are the minimum and maximum numbers of the functional units. In Figure (7.6) the gene-length is $= N_{DFG} + 1 + 3N_G$ where N_{DFG} is number of nodes in the DFG and N_G is number of the bus base-segments, see chapter 5.

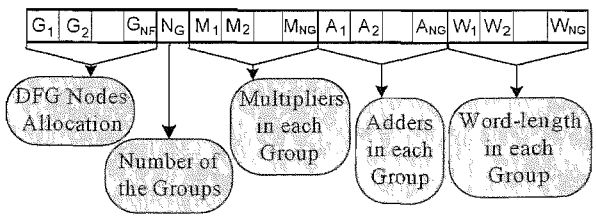


FIGURE 7.6: Genomes structure in the applied GA

The optimisation algorithm is very similar to the Listing (7.1) with some modifications, as shown in Listing (7.2). After assignment of the nodes of the DFG to groups (bus segments) by genes, RC allocation and scheduling [33] is applied to evaluate each individual's fitness.

Listing 7.2: The synthesiser algorithm for MW^{2}P-Bus

```
INPUT: Data Flow Graph;
OUTPUT: Synthesisable RTL-VHDL;
Objectives: Minimising the area, power consumption;
        and delay and maximising the accuracy;
BEGIN
  Initialise the DFG using ASAP;
  Find the maximum required FUs for each type
  Find the maximum number of the bus segments;
  Find the Bound solutions for each cost;
  First Generation;
  FOR N-iterations;
    FOR all the population;
      Find Area();
      Find Static Energy Consumption;
      Do List-Scheduling();
      Find the C-Steps for Non-Partitioned Bus;
      Modify the Bus Partitioning Effects;
      Find Delay();
      Evaluate Dynamic Energy Consumption;
      Evaluate Digital Noise in the output;
      Find Power();
      IF (a cost is out of its constraint range) THEN
          modify its corresponding weight coefficient
          in the combined cost function;
      Find the overall combination cost;
    END FOR;
    Produce the next generation;
  END FOR;
END;
```

7.5 Summary

In this chapter, an optimisation method for the synthesiser of the proposed method is presented. First, the model of the problem is presented in which a relationship between the design parameters and the cost functions is derived. The dimensions of the feasible space are vital parameters in every kind of optimisation. Furthermore, the relationship between cost functions and controlling parameters has a great impact on the optimisation method chosen. In the presented problem, which is based on a target architecture, the feasible space is limited to a smaller range of parameters which results in an increase

in optimisation speed. However, regarding the dimensions of the feasible space and nonlinearity of the relationships between the cost functions and the parameters, there is still a big number of possible solutions for the problem.

In addition to these difficulties, the optimisation problem consists of several objectives namely, design area, power consumption, latency and accuracy. Thus a multi-objective optimisation is required to deal with the NP-hard proposed problem and the nonlinear cost functions. A genetic algorithm is proposed for optimisation of the problem, which is based on vector evaluation to cope with the multi-objective problem. The required gene codes and cost functions are provided for the problem and algorithm is explained in detail. The next chapter presents results for some case studies.

Chapter 8

Case Studies

8.1 Introduction

Chapters 4, 5 and 6 have described a number of new techniques for the analysis, design and optimisation of computationally intensive circuits. In this chapter, several case studies are presented to evaluate the improvements which our proposed method can make in design implementation costs. To show the ability of the SNA method in error analysis of the computational systems with nonlinear units and also with data inputs which cannot be assumed to have uniformly distribution PDFs, design procedure is applied comprehensively to a Monte-Carlo simulation of an option pricing equation.

In the following, several other case studies are investigated using different implementation structures, which were presented in chapter 5. These implementations are divided into three cases, which are word-length optimisation with single shared bus structure, MWP-Bus structure and MW^{2}P-Bus structure.

8.2 Black-Scholes Option Pricing Equation

Fischer Black and Myron Scholes [10, 11] published a paper which redefined finance and derivatives and which have had considerable effect on the stock exchange market. The piece is arguably one of the most important papers within finance theory to date, allowing pricing of various derivatives, including options on commodities, financial assets and even pricing of employee stock options [70].

From a mathematical point of view, this model can be expressed in the form of a Partial Differential Equation (PDE) [95]. There are several approaches to solve this problem numerically, which are briefly explained in [53]. In this study, a Monte-Carlo approach is chosen for solution implementation and its corresponding algorithm appearers in the MATLAB coding list of Listing (8.1) (quoted from [53]).

Listing 8.1: Matlab codes for Monte Carlo simulation of the Black-Scholes PDE

```
%=========================================
%        Monte Carlo valuation for a European call
%=========================================
%        Problem and method parameters
% S = 2; E = 1; r = 0.05; sigma = 0.25; T = 3; M = 1e6; randn(state, 100);
%=========================================
    Svals = S · exp((r − sigma^2/2) · T + sigma · √T · randn(M, 1));
    Pvals = exp(−r · T) · max(Svals − E, 0);
    Pmean = mean(Pvals);
    width = 1.96·std(Pvals)/√M;
    conf = [Pmean − width, Pmean + width];
%=========================================
```

By inspecting the algorithm of Listing (8.1), it can be observed that this algorithm can be divided into two parts: initial calculations and iterations. The first part refers to the values which need to be calculated only once at the beginning of the program and can be saved in the memory for repetitive iterations, which includes random number generation and its combination with the initially calculated values. Since initial calculations only happen once for each simulation, they are not considered in our optimisation and assumed to be saved in the local memory. The iterative part of the algorithm , as depicted in Figure (8.1) and (8.2-a), can also be divided into two parts: Gaussian Random Number Generator (GRNG) and the output calculation part, see Figure (8.2-b). Several methods have been introduced in the literature for specific hardware implementations of random number generators. Two famous and widely used algorithms are Box-Muller [13] and Ziggurat [82] methods; their hardware implementations can be found in [74] and [140] respectively. Accordingly we focus on the output calculation part of the algorithm as depicted in Figure (8.2-b).

As mentioned, there are a set of pre-calculated values which are nonlinear functions of the initial inputs. These nonlinear functions in the algorithm can be evaluated by different methods; a full explanation of many methods can be found in [92]. It is discussed in [73] and [87] that piecewise polynomial is an efficient method which can provide the required accuracy with lower hardware cost. So we assume that a piecewise polynomial approximation is employed for nonlinear function evaluation as depicted in Figure (8.3). In this structure a multivariate function $f(x_1, x_2, \cdots, x_m)$ is approximated by $\hat{f}$ in the form of an order-n polynomial $P_n(x_1, x_2, \cdots, x_m)$ in which its coefficients can be dependent on the input variable(s) $(x_1, x_2, \cdots, x_m)$. After calculation of the coefficients, $\hat{f}$ is

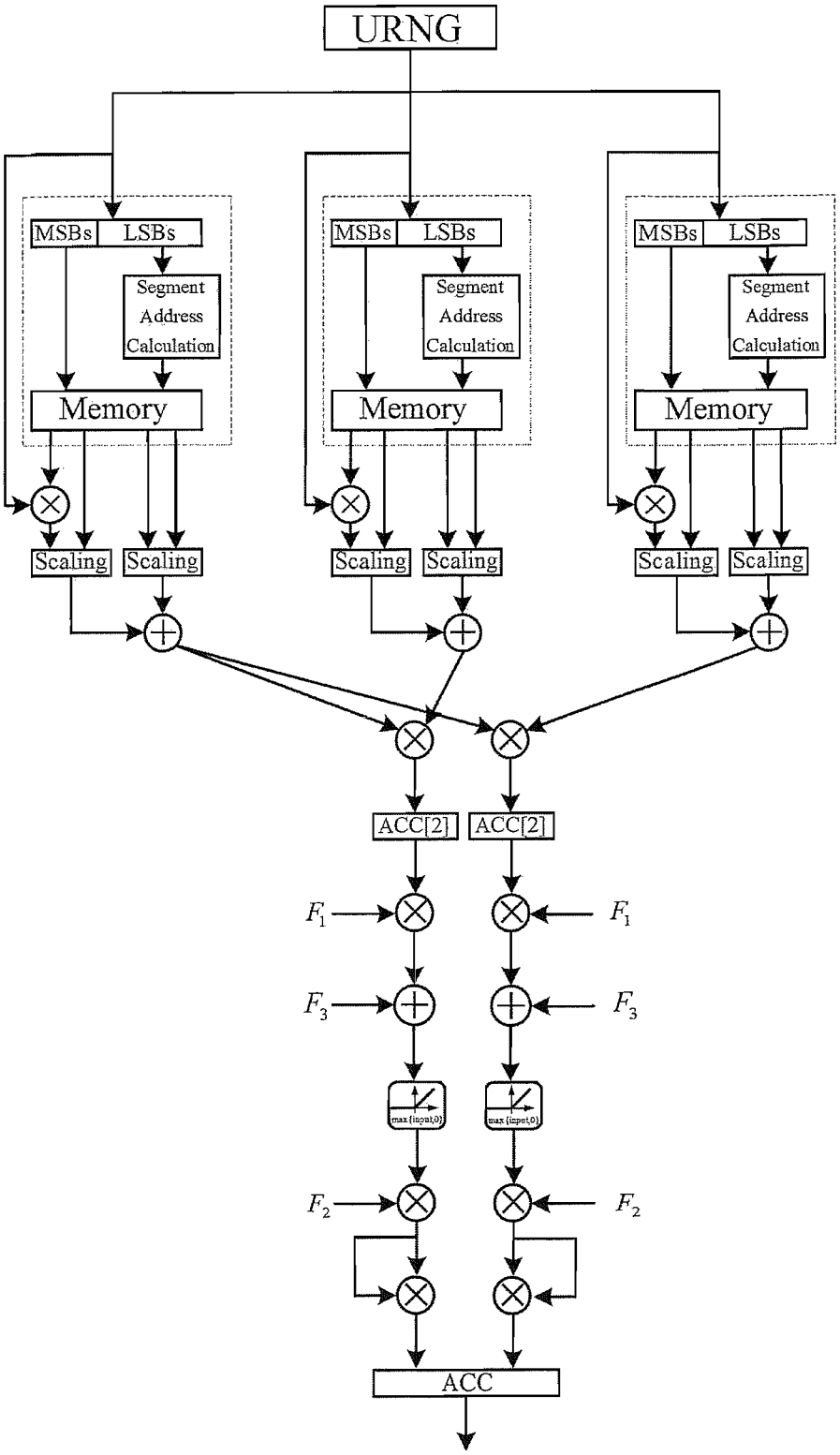


FIGURE 8.1: Iteration part of the Black-Scholes Monte-Carlo algorithm.

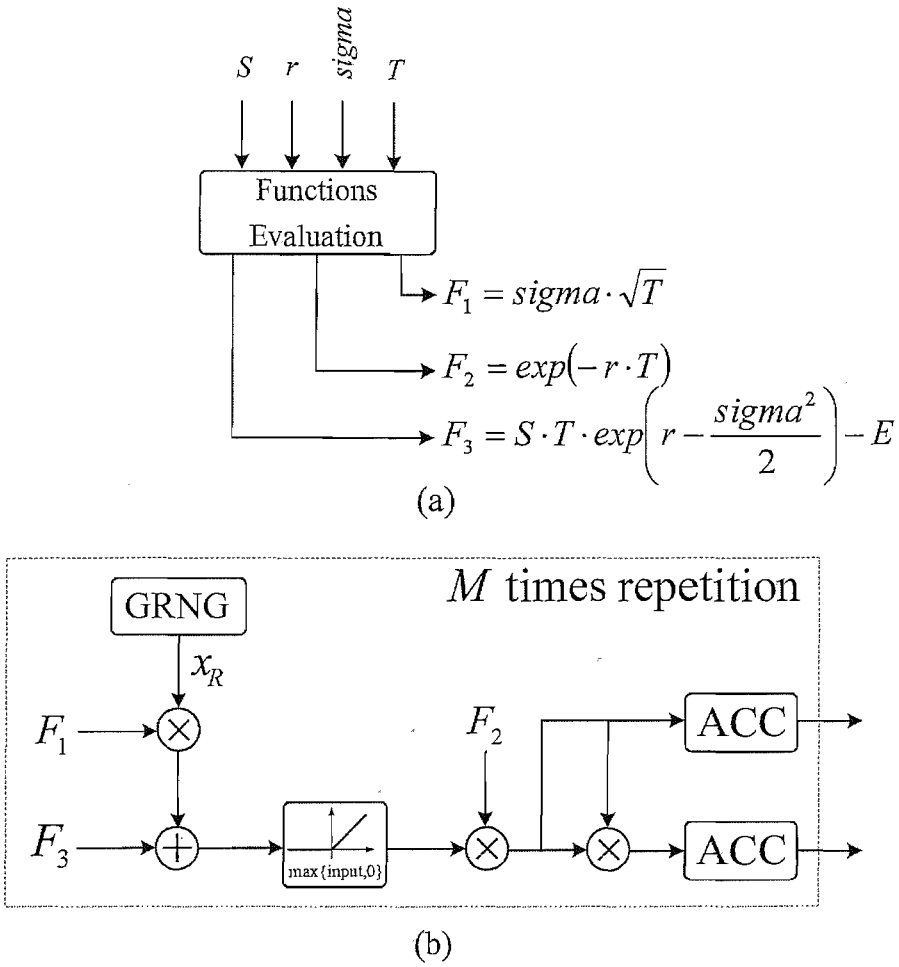


FIGURE 8.2: General structure of Monte-Carlo evaluation of the Black-Scholes equation a)initial calculations b)iterative part.

constructed by calculating $P_n(x_1, x_2, \cdots, x_m)$. This method, with $n = 1$, is utilised in [73] and [36] with $n = 2$ and with $n = 3$ in [87].

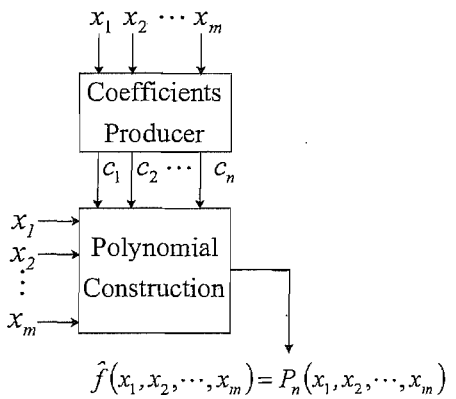


FIGURE 8.3: General structure for multivariate function evaluation.

We recall from chapter 4 that the computational error consists of two parts: inherited error from function evaluation units and errors which are produced by arithmetic units

(multiplier and adder/subtractor). Accordingly, for error analysis of the iteration part of the algorithm in Listing (8-1), the first step should be estimation of the errors which the input variables carry in. The major error sources are those resulting from the piecewise polynomial function evaluation units. Then accuracy analysis can be performed based on the errors which are produced by the function evaluation units in the initial calculations or in the random number generator unit. According to [87], the output error of the function evaluation consists of three parts: E_{poly}, E_{coef} and E_{round}, as shown in Equation (8.1) .

$$\hat{f}(\vec{X}) = P_n(\vec{X}) = f(\vec{X}) + E_{poly} + E_{coef} + E_{round}, \tag{8.1}$$

where E_{poly} represents the error which arises from the polynomial approximation of the function. This error depends on the polynomial order and the approximation method. E_{coef} is the quantisation error of the polynomial coefficients, which needs to be fitted into the lookup table. E_{round}, on the other hand, represents the rounding error which is produced by arithmetic operations during polynomial reconstruction. Accordingly, input values of the repetitive part of the algorithm of Figure (8.2-b) can be written as follow:

$$\begin{aligned}
\hat{x}_R &= x_{R0} + x_{R1}\epsilon_R, \\
\hat{F}_1 &= F_{10} + F_{11}\epsilon_{F1}, \\
\hat{F}_2 &= F_{20} + F_{21}\epsilon_{F2}, \\
\hat{F}_3 &= F_{30} + F_{31}\epsilon_{F3},
\end{aligned}$$

where x_R is the Gaussian random number that is produced by the GRNG unit and ϵ represents noise symbols. We assume that these errors are uniformly distributed over the range, however, other types of distributions can also be used in our analysis method. Applying these assumptions, the error at the output of the circuit in Figure (8.2-b) can be calculated using SNA method, as explained in chapter 4.

$$\begin{aligned}
&= \overbrace{(x_{R0}F_{10} + F_{30})}^{A_I} + \\
&\quad (x_{R1}F_{10}\epsilon_R + x_{R0}F_{11}\epsilon_{F1} + x_{R1}F_{11}\epsilon_R\epsilon_{F1} + F_{31}\epsilon_{F3}) + \\
&\quad (a_{M1}\epsilon_{M1} + a_{A1}\epsilon_{A1}), \\
&= A_I + A_E,
\end{aligned} \tag{8.2}$$

where A_I represents the ideal calculation result and A_E is the computational error, $a_M\epsilon_M$ and $a_A\epsilon_A$ represent error coefficients and symbols for multipliers and adders respectively.

The output value of the Equation (8.2) is the input of the nonlinear unit in Figure (8.2-b) which results in Equation (8.3).

$$\max\{(A_I + A_E), 0\} = \begin{cases} A_I & A_I > -A_E \\ 0 & \text{otherwise} \end{cases} \tag{8.3}$$

Thus the error after this nonlinear function in comparison with the ideal computation can be depicted as in Figure (8.4), which shows that the output error of this unit can be written as in Equation (8.4).

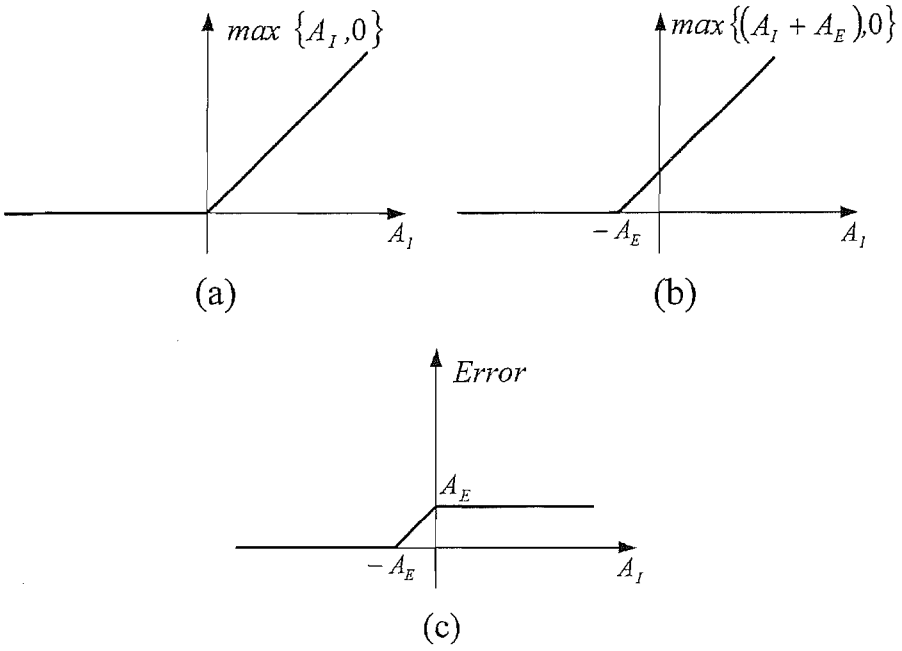


FIGURE 8.4: Input-Output error dependency of the nonlinear unit in the circuit of Figure (8.2) (assuming $A_E > 0$) a)in ideal case b)with input error c)difference between (a) and (b).

$$\text{Error after nonlinear unit} = \begin{cases} 0 & A_I < \min\{-A_E, 0\} \\ A_I + A_E & \min\{-A_E, 0\} \leq A_I < \max\{-A_E, 0\} \\ A_E & \max\{-A_E, 0\} \leq A_I \end{cases} \quad (8.4)$$

where $A_I = x_{R0} \cdot F_{10} + F_{30}$ and according to the algorithm, x_{R0} is a standard Gaussian random variable ($x_{R0} \sim N(0,1)$), then A_I is also a Gaussian random variable with mean F_{30} and variance F_{10}^2 ($A_I \sim N\left((F_{30}), (F_{10}^2)\right)$). Therefore, the result error has a PDF ($f_{E_o}(x)$) which can be stated (8.5).

$$f_{E_o}(x) = \begin{cases} 0 & x < \min\{A_E, 0\} \\ P(A_E = 0) + P(A_I < \min\{-A_E, 0\}) & x = 0 \\ P(\min\{-A_E, 0\} \leq A_I < \max\{-A_E, 0\}) & \min\{A_E, 0\} < x < \max\{A_E, 0\} \\ P(\max\{-A_E, 0\} \leq A_I) & x = A_E \\ 0 & x > \max\{A_E, 0\} \end{cases} \quad (8.5)$$

Since PDF of the A_I and A_E are known, $f_{E_o}(x)$ can be derived and used to extract the output error of the algorithm. It can be observed that since the computational error affects both data and decision in this nonlinear unit, it is impossible to analyse with the other error analysis methods mentioned in chapter 3, which shows the strength of over method comparing previous works in the field.

Considering Listing (8.1), let us assume that all the input values including x_R are represented in 32-bit fixed-point numbers and all the arithmetic operations use truncation on their output rounding. We also assume that input errors which come from other units have a uniform distribution with $10^{-8}\%$ deviation from the central value, however our method will be the same for other kind of distributions, see chapter 4 for more details. It is also assumed that x_R and ϵ_R are independent. Figure (8.5) shows the output error without the effect of the rounding.

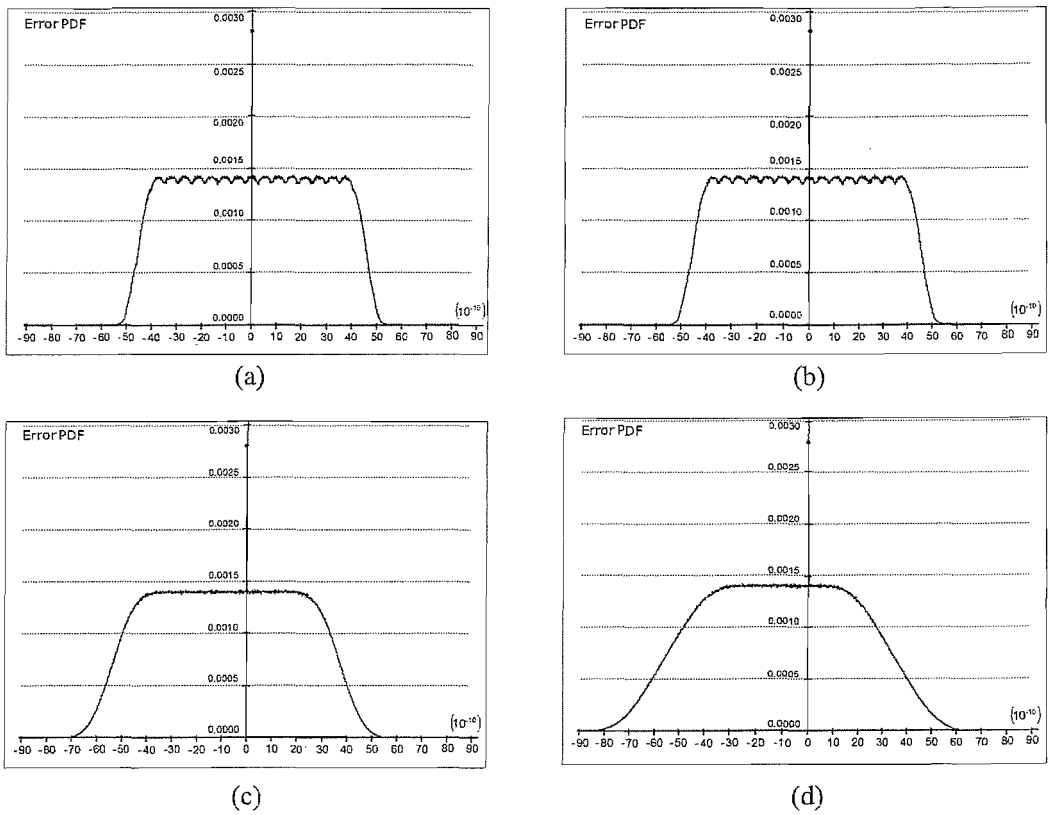


FIGURE 8.5: Error PDF of the algorithm a)Without considering rounding effect and the nonlinear unit b)With effect of the nonlinear unit c)With effect of the nonlinear unit and 10^{-10} rounding error d)With effect of the nonlinear unit and uniform 32-bit-width for all the arithmetic units.

In practice, it can be observed that multipliers are the most time consuming operations in this algorithm. As shown in the literature [65, 100, 35], multipliers can be implemented in the form of serial or parallel (pipelining) structures. The implementation cost of parallel multipliers is higher than serial multipliers, whereas its execution time can be as low as one clock cycle. Furthermore, our inspection shows that area, delay and power consumption of the serial multiplier is almost linearly dependent on the word-length whereas parallel multipliers show quadratic dependency on word-length with minimum operation latency. The iterative part of the Monte-Carlo simulation, therefore, can be speeded up either by utilising pipelining multipliers or by using several serial multipliers working in parallel for every loop of the iteration of the algorithm. Either way, word-length optimisation has a great impact on the implementation cost of the hardware. To

show this impact on design costs and compare the effects of different word-lengths on the output error of the computation, we perform an exhaustive comparison between design area on Xilinx XC2VP2 FPGA (in terms of usage percentage) and output error range and variance. The results are presented in Table (8.1). Synthesis results of Table (8.1) are extracted using Mentor Graphics synthesiser tool (Leonardo 2005 a.76). Here we consider that the error range of the function evaluation units is the same as the output calculation part in Figure (8.2).

TABLE 8.1: FPGA LUT usage for different uniform word-lengths.

Uniform WL	Total LUTs	LUT usage	Error range $x_h - x_l$	Error σ^2
32	3957	140%	2.318×10^{-8}	3.608×10^{-18}
31	3709	131%	5.031×10^{-8}	1.447×10^{-17}
30	3470	123%	1.017×10^{-7}	5.790×10^{-17}
29	3238	114%	2.031×10^{-7}	2.310×10^{-16}
28	3015	107%	4.318×10^{-7}	9.249×10^{-16}
27	2799	99%	7.778×10^{-7}	3.705×10^{-15}
26	2592	92%	1.584×10^{-6}	1.476×10^{-14}
25	2392	84%	3.219×10^{-6}	5.917×10^{-14}
24	2201	78%	6.260×10^{-6}	2.370×10^{-13}
23	2017	71%	1.215×10^{-5}	9.429×10^{-13}
22	1842	65%	2.541×10^{-5}	3.784×10^{-12}
21	1674	59%	4.759×10^{-5}	1.516×10^{-11}
20	1515	53%	9.826×10^{-5}	6.071×10^{-11}
19	1363	48%	1.988×10^{-4}	2.420×10^{-10}
18	1220	43%	2.931×10^{-4}	9.703×10^{-10}
17	1084	38%	7.985×10^{-4}	3.885×10^{-9}
16	957	33%	1.570×10^{-3}	1.548×10^{-8}

On the other hand, after optimisation the word-length of the arithmetic units will be: $W_1 = 25$, $W_2 = 26$ and $W_3 = 23$ (for FUs in Figure (8.2-b) from left to right starting with the upper multiplier). The output error range is 4.978×10^{-6} and the output error variance is 9.240×10^{-14}. The output error PDF of the optimised design is also compared with the output error PDF of uniform word-length design with $W = 24$, $W = 27$ and $W = 25$ in Figure (8.6). These figures show that optimised design has a better error PDF in comparison with $W = 24$ uniform word-length design however it is worse comparing with the case of uniform $W = 25$ and $W = 27$. Implementation of this design needs 2243 **LUT**s (79%) on the same FPGA, which is 24% lower than uniform word-length ($W = 27$) implementation and 6.64% less than uniform word-length $W = 25$.

It can be seen that with higher error range it is possible to save considerably resources and generally this method can be used to fit the design to the target FPGA by trading word-length without loosing accuracy. Furthermore, it must be considered that in this example the number of arithmetic units in the design is very low (one adder and two

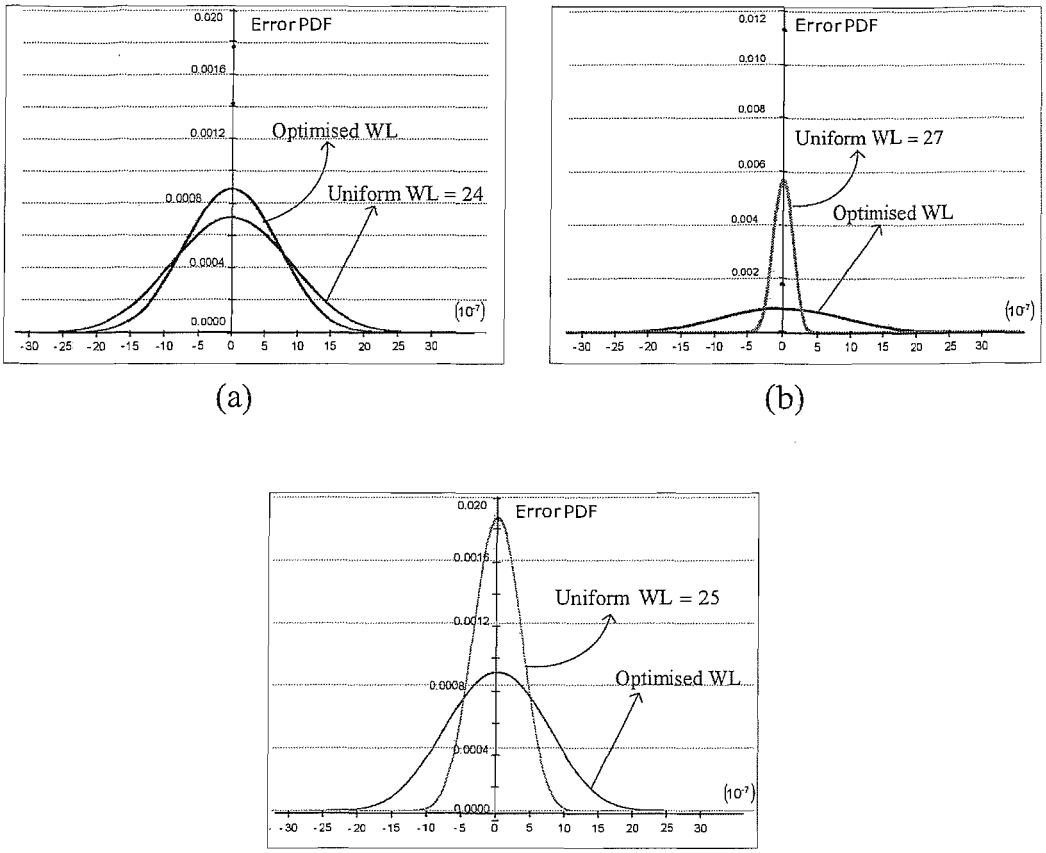


FIGURE 8.6: Error PDF of the iteration part of the Black-Scholes equation after WL optimisation a)comparing with $W = 24$ b)comparing with $W = 27$ c)comparing with $W = 25$.

multipliers), which means the optimiser has not had too much freedom to chose the different word-lengths for functional units.

The following sections present more complicated case studies with different implementation structure.

8.3 Single Shared Bus Implementation

In this section four case studies are implemented in ST 0.12μm technology using the proposed method with a single shared bus structure. *Design I* is an order-18 difference equation, *Design II* is a Filter (FIR-25), *Design III* is an 8-point FFT and *Design IV* is a DCT 4×4, more details regarding these circuits can be found in appendix B. Multipliers in all these designs in this section are sequential (Booth) multipliers.

In the first step, to examine the effect of the word-length on the implementation costs, circuits are implemented with different uniform word-lengths, which means all the arithmetic units have the same word-length and the results of these implementations are provided in Table (8.2) using our sample predesigned cell library of arithmetic units including sequential multiplier. In this table, basic design costs: area, power consumption, latency and digital noise in the output node are presented for different values of word-lengths (W=8, 16, 24 and 32). It must be remembered that power consumption is a function of the latency, thus to reduce the complexity of the results, we measure energy per clock cycle to show the average power consumption in all the results in the rest of this chapter.

The results in Table (8.2) show a considerable dependency of the design costs on the word-length. As depicted in Figure (8.7), area and power consumption decrease almost linearly by word-length reduction. Latency can be decreased providing that sequential multipliers or other arithmetic units are used in the design. Digital noise, on the other hand, shows an exponentially increase when word-length reduces, which means accuracy is more sensitive to the word-length than other costs.

Apart from the initial investigation, this set of information can be used as the basic evaluation scales and the constraint values for other optimisation results, as explained in section 7.4. In other words, these uniform word-length designs can be used as the reference point for the genetic search to reduce the optimisation time. Practically, in constrained design optimisations, the optimal points are expected to be found around the uniform word-length design which satisfies the constraint. Therefore, instead of starting from a random initial point, our search algorithm starts from the uniform word-length design which satisfies the constraint. These constraints are referred to as C_1, C_2, C_3 and C_4 in Tables (8.3) to (8.6).

In the second step, constrained optimisations are applied for designs considering word-length as a synthesis parameter. Since there are four different costs in this study, four different cases of constraints are considered. Table (8.3) shows the synthesis results for the same systems where design area is constrained. The constraint values for the area cost function in Table (8.3) are the area cost results in Table (8.2).

Similarly Tables (8.4), (8.5) and (8.6) show synthesis results with optimisation constraints for energy consumption, output noise and latency respectively. Again the constraint values for each column and row of these tables can be found in the corresponding column and row in the Table (8.2). In these tables A, E, N and D stand for: area cost (in μm^2), energy consumption cost (in μWatt), digital noise variance in the output and delay cost (measured by the number of clock cycles) of the design respectively. Although choosing word length of the FUs can can affect the maximum delay of the operations to select the clock frequency of the design, in this work, it is assumed that all designs have the same clock rate for to perform a basic comparison between different designs.

TABLE 8.2: Different fixed-uniform WL for designs.

Designs	Cost	Different cases of Uniform WL for all FUs			
		$C_1(W=8)$	$C_2(W=16)$	$C_3(W=24)$	$C_4(W=32)$
Design I	Area	4152	8304	12456	16608
	Energy	5672.73	20779.2	45753.7	80602.9
	Noise	1.03E-2	4.04E-5	1.58E-7	6.16E-10
	Delay	168	311	454	598
Design II	Area	22184	44368	66552	88736
	Energy	7143.11	26929.2	59584.1	105061
	Noise	1.28E-2	5.09E-5	1.99E-7	7.62E-10
	Delay	58	82	106	130
Design III	Area	14456	44368	89736	119648
	Energy	9631.03	35813.4	78909	138029
	Noise	2.95E-2	1.15E-4	4.50E-7	1.76E-9
	Delay	100	110	121	145
Design IV	Area	29912	111344	174744	222688
	Energy	18256.5	71076.6	156085	273138
	Noise	3.26E-2	1.27E-4	4.97E-7	1.94E-9
	Delay	121	130	152	178

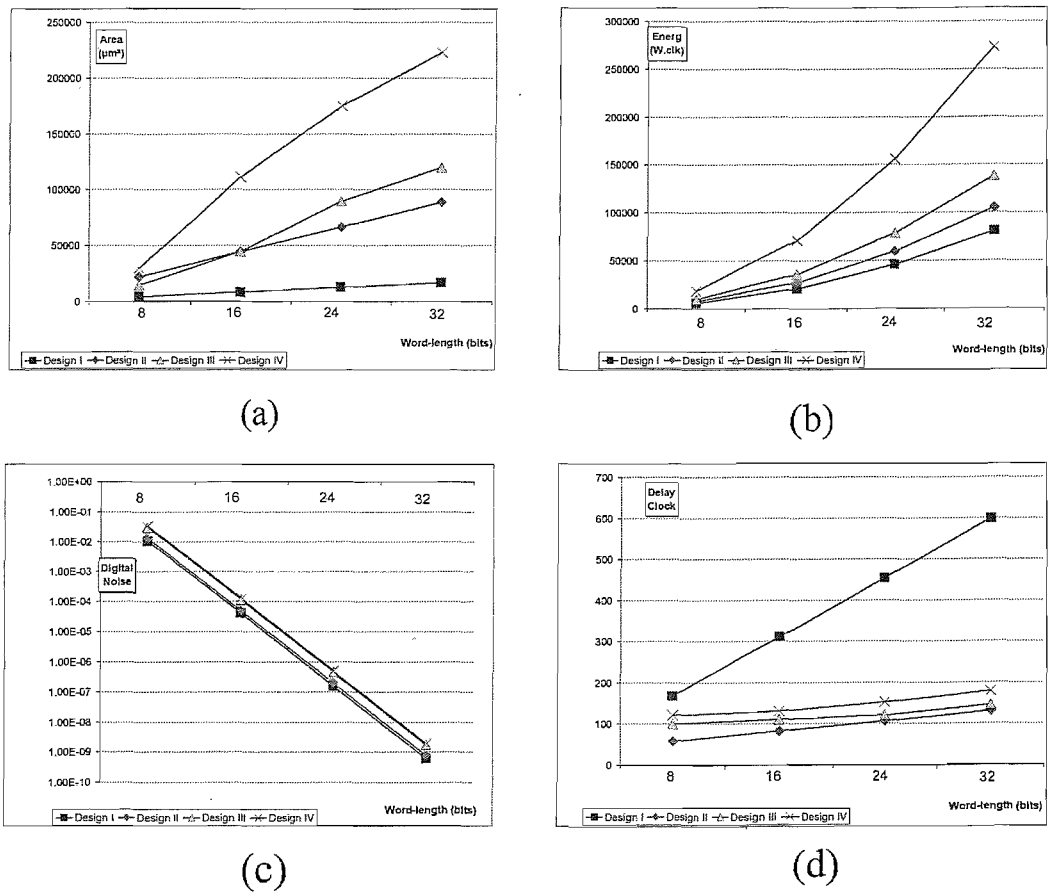


FIGURE 8.7: Basic costs dependency on the WL for different designs a)Area b)Power consumption c)Digital noise d)Delay.

TABLE 8.3: Optimisation results with area constrained synthesis based on the four uniform WL cases in cases Table(8.2)).

Designs	Cost	Area is constrained to:			
		C_1	C_2	C_3	C_4
Design I	Energy	4478.08	18350.9	42218.3	75706.9
	Noise	1.03E-2	4.04E-5	1.09E-7	6.16E-10
	Delay	150	293	436	580
Design II	Energy	6295.61	25751.8	58066.5	101421
	Noise	1.15E-2	4.80E-5	1.77E-7	6.82E-10
	Delay	53	79	102	126
Design III	Energy	9095.21	34298.4	77773.5	136270
	Noise	2.01E-2	5.68E-5	3.14E-7	1.05E-9
	Delay	100	107	113	137
Design IV	Energy	17341.5	70962.8	152568	271351
	Noise	2.62E-2	1.22E-4	4.71E-7	1.63e-9
	Delay	119	126	143	168

TABLE 8.4: Optimisation results with energy constrained synthesis based on the four uniform WL cases in cases Table(8.2)).

Designs	Cost	Energy is constrained to:			
		C_1	C_2	C_3	C_4
Design I	Area	3633	7785	11937	16483
	Noise	1.03E-2	4.04E-5	1.58E-7	4.27E-10
	Delay	150	293	436	580
Design II	Area	21987	44243	66105	87967
	Noise	1.14E-2	4.29E-5	1.66E-7	6.92E-10
	Delay	55	78	102	127
Design III	Area	14456	44118	82724	118754
	Noise	2.73E-2	6.31E-5	2.46E-7	1.40E-9
	Delay	100	106	119	135
Design IV	Area	27086	100468	164315	222438
	Noise	2.79E-2	1.15E-4	4.69E-7	1.86E-9
	Delay	119	126	144	169

From these results it can be concluded that in almost all cases, word-length optimisation saves hardware costs, however this improvement is dependent on the circuit complexity and its constraints. In summary, the results show up to 16.5%, 21%, 50%, 10% improvements in area, energy consumption, digital noise and latency respectively. Table (8.7) shows improvements for different costs resulting from constrained optimisation in comparison with the basic cases of the uniform word-lengths in Table (8.2).

TABLE 8.5: Optimisation results with noise constrained synthesis based on the four uniform WL cases in cases Table(8.2)).

Designs	Cost	Noise is constrained to:			
		C_1	C_2	C_3	C_4
Design I	Area	3633	7785	11937	16089
	Energy	4478.08	18350.9	42091.6	75706.9
	Delay	150	293	436	580
Design II	Area	20646	43349	64495	87323
	Energy	6074.11	24996	55273.2	100628
	Delay	53	79	101	126
Design III	Area	12649	41595	88645	116178
	Energy	7572.32	31676.3	76467.4	128521
	Delay	99	107	114	135
Design IV	Area	26889	100915	163027	219987
	Energy	16856.9	69308.4	147610	265048
	Delay	119	126	145	168

TABLE 8.6: Optimisation results with delay constrained synthesis based on the four uniform WL cases in cases Table(8.2)).

Designs	Cost	Delay is constrained to:			
		C_1	C_2	C_3	C_4
Design I	Area	3633	7785	11937	16089
	Energy	4478.08	18350.9	42091.6	75706.9
	Noise	1.03E-2	4.04E-5	1.58E-7	6.16E-10
Design II	Area	21665	44243	65783	88736
	Energy	6936.68	26541.3	57530	104750
	Noise	1.20E-2	4.46E-5	1.87E-7	6.96E-10
Design III	Area	14331	36909	79951	119201
	Energy	8879.05	30696	73861.1	136681
	Noise	2.81E-2	9.87E-5	3.29E-7	1.32E-9
Design IV	Area	26245	106908	165353	221919
	Energy	16355.9	70709.7	151477	269987
	Noise	2.06E-2	1.10E-4	4.58E-7	1.84E-9

TABLE 8.7: Cost reduction resulted for different designs from constrained optimisation based on the four uniform WL cases in cases Table(8.2)).

Constrains	Costs	Cost reduction (Min%-Max%)			
		Design I	Design II	Design III	Design IV
Area	Energy	06.0 - 21.0	02.5 - 11.9	01.3 - 05.6	00.2 - 05.0
	Noise	00.0 - 31.0	05.7 - 10.2	30.0 - 50.6	04.0 - 19.6
	Delay	03.0 - 10.7	03.1 - 08.6	00.0 - 06.6	01.7 - 05.9
Energy	Area	00.8 - 12.3	00.3 - 00.9	07.5 - 45.3	00.1 - 09.8
	Noise	00.0 - 30.6	09.2 - 16.6	00.0 - 06.9	04.1 - 14.4
	Delay	03.0 - 10.7	02.3 - 05.2	00.1 - 09.8	01.7 - 05.3
Noise	Area	03.1 - 12.3	01.6 - 06.9	01.2 - 12.5	01.2 - 10.1
	Energy	06.0 - 21.0	04.2 - 15.0	03.1 - 21.4	02.5 - 07.7
	Delay	03.0 - 10.7	03.0 - 08.6	01.0 - 06.9	01.7 - 05.6
Delay	Area	03.1 - 12.3	00.0 - 02.3	00.4 - 16.8	00.3 - 12.3
	Energy	06.0 - 21.0	00.3 - 03.4	01.0 - 14.3	00.5 - 10.4
	Noise	00.0 - 00.0	06.0 - 12.4	04.7 - 26.9	05.2 - 37.0

8.4 Partitioned Bus Implementation

The segmented bus structures (MWP-BUS and MW^{2}P-BUS) can improve the design efficiency, as explained in chapter 5. To investigate the effect of the structures in design optimisation, a number of case studies have been implemented in ST 0.12μm technology using this method. 8-point Digital Cosine Transform (DCT) as, *Design I*, 5-order Elliptic Filter, as *Design II*, and an RGB to YCbCr converter, as *Design III*, are used to illustrate this method of datapath optimisation in comparison with other implementation techniques. The results of design syntheses are presented in Tables (8.8) to (8.11). Since the effectiveness of the method depends on the number of functional units and the data traffic between them, every case study is implemented using four different binding assumptions in which the number of functional unit is restricted (with 2, 3, 4 and 5 Add/Subtract and Multiplier FUs) for each benchmark.

Similar to the previous section, at the first step, for each binding, every design is implemented with the assumption of a fixed and uniform word-length (W=16) for all the functional units and then their design costs (area, power consumption, delay and variance of digital noise) are calculated as the reference values for optimisations. These results are placed in the first column of the results in the tables, which is tagged as **Single Bus Uniform W=16**.

In the second step, a general word-length optimisation method, as introduced in chapter 7.4, is applied to each design. These optimisations are performed based on a single shared bus assumption. It is observable that the results of this word-length optimised synthesis suggest reductions in the costs. As it obtainable from the table, the computational accuracy most often is traded with improvements in other costs. As discussed in chapter 3, however, improvements are possible without losing accuracy.

In the third step, the partitioned bus method (MWBP) is used to synthesise the benchmarks. The major achievement of this method in comparison with the non-partitioned bus is latency reduction of the circuit. Since this method speeds up data communications between functional units, its resultant improvement varies with the number of functional units and the data traffic between them. On the other hand, since MWP-BUS requires bus switches to control data communications in the bus, the area and energy consumption of the circuit are expected to increase as a result of applying this method. This side effect can be observed in the results. Digital noise is not expected to change considerably in comparison with the single bus method.

In completion of Table (8.8) to (8.11), Table (8.14) presents the MWBP bus partitioning and functional units grouping results with their word-lengths for each bus segment. In comparison of the word-length optimisation with a shared bus structure, in terms of wire-length of the bus, MWBP trades average bus width with bus switches. In other word, MWBP uses bus switches to reduce the average wire length in the final implementation

of the systems. Table (8.14) shows that different word-lengths are allocated to the bus segments when the number of functional units are more than 6. In all the groupings, IO-port of the system is in the first group.

In conclusion, the simulation results show that there is a considerable improvement in delay because of the bus partitioning method, which is a valuable achievement, as shown in Table (8.12) and Table (8.13). Moreover, according to Table (8.8) to (8.11), MWBP is more effective in the case of more complicated designs (in terms of the number of functional units and design complexity), which means by increasing the number of functional units in binding or complexity of the implemented benchmark, the effect of the bus switches reduces the overall cost. Table (8.14) also supports this supposition that a bigger number of functional units consequences partitioned buses with different word-length as an optimal point.

TABLE 8.8: MWP-Bus and MW^{2}P-Bus optimisation results for different number of FUs, Binding I.

Designs	Costs	Binding I			
		2 ALU , 2 Multiplier			
		Single Bus Uniform W=16	Single Bus Optimised W	MWP-Bus Optimised W	MW^{2}P-Bus Optimised W
Design I	Area	16608	14532	15344	15344
	Delay	185	169	159	159
	Noise	1.14E-7	4.57E-7	4.14E-7	4.14E-7
	Energy	18829.4	14652.3	15466.2	15466.2
Design II	Area	16608	14532	15344	15344
	Delay	115	107	94	94
	Noise	7.95E-8	3.18E-7	3.95E-7	3.95E-7
	Energy	9102.68	7161.25	8198.25	8198.25
Design III	Area	16608	14532	15344	15344
	Delay	88	80	72	72
	Noise	6.71E-8	2.68E-7	6.71E-8	6.71E-8
	Energy	9924.84	7664.48	8150.20	8150.20

In the forth step, the MW^{2}P-BUS is used to synthesis the benchmarks. As discussed in chapter 5, when the number of functional units increases, data traffic in the bus segments, which are placed in the central part MWP-BUS, can increases to very high rates. In these cases, data communication speed and consequently system latency are decreased considerably. MW^{2}P-BUS structure, alternatively, adds parallel segments to the bus to provide supplementary paths to avoid path congestions in the central segments of the bus. In other words, in comparison with MWP-BUS, MW^{2}P-BUS trades hardware resources (bus switches and bus segments) with latency. Last columns in Tables (8.8) to (8.11) give the optimisation results with MW^{2}P-BUS method and Table (8.15) presents the bus partitioning and FU grouping results with their word-lengths for each bus segment.

TABLE 8.9: MWP-Bus and MW^{2}P-Bus optimisation results for different number of FUs, Binding II.

Designs	Costs	Binding II 3 ALU , 3 Multiplier			
		Single Bus Uniform W=16	Single Bus Optimised W	MWP-Bus Optimised W	MW^{2}P-Bus Optimised W
Design I	Area 5	24912	21798	22610	22610
	Delay	148	136	83	83
	Noise	1.14E-7	1.14E-7	1.14E-7	1.14E-7
	Energy	19078	14858.1	15479	15479
Design II	Area	24912	21798	22610	23480
	Delay	107	101	74	63
	Noise	7.95E-8	3.18E-7	7.95E-8	7.95E-8
	Energy	9408.44	7422.91	8782.98	9120.94
Design III	Area	24912	21798	22610	22610
	Delay	71	65	33	33
	Noise	6.71E-8	2.68E-7	6.71E-8	6.71E-8
	Energy	10049.2	7767.38	8528.61	8528.61

TABLE 8.10: MWP-Bus and MW^{2}P-Bus optimisation results for different number of FUs, Binding III.

Designs	Costs	Binding III 4 ALU , 4 Multiplier			
		Single Bus Uniform W=16	Single Bus Optimised W	MWP-Bus Optimised W	MW^{2}P-Bus Optimised W
Design I	Area	33216	29064	31265	32154.33
	Delay	138	130	65	46
	Noise	1.14E-7	4.57E-7	3.19E-7	1.19E-7
	Energy	19440.9	15187.4	15769.6	17218.17
Design II	Area	33216	29774	31265	35687.5
	Delay	101	97	74	60
	Noise	7.95E-8	3.18E-7	1.20E-7	7.20E-8
	Energy	9875.48	7587.42	8947.87	10145.74
Design III	Area	33216	29064	31265	32135
	Delay	59	55	32	20
	Noise	6.71E-8	2.68E-7	3.71E-7	1.49E-7
	Energy	10126.4	7840.88	8924.59	9686.86

To interpret the results of MWP and MW^{2}P methods in Tables (8.8) to (8.11), we need to compare Tables (8.14) and (8.15). It is observable from Tables (8.14) and (8.15) that both methods resulted in the same grouping and word-lengths for some designs, which have a fewer number of functional units (Binding I & II). Recalling from chapter 5, MWP and MW^{2}P have the same structure when functional units are divided into two groups, thus implementation costs for both methods are the same in Tables (8.8)

TABLE 8.11: MWP-Bus and MW^{2}P-Bus optimisation results for different number of FUs, Binding IV.

Designs	Costs	Binding IV, 5 ALU , 5 Multiplier			
		Single Bus Uniform W=16	**Single Bus Optimised W**	**MWP-Bus Optimised W**	**MW^{2}P-Bus Optimised W**
Design I	Area	41520	36330	37961	41083.5
	Delay	132	124	71	35
	Noise	1.14E-7	4.57E-7	1.14E-6	1.14E-7
	Energy	19803.8	15481.4	16227.6	17562.41
Design II	Area	41520	36282	37675	45940
	Delay	101	90	72	59
	Noise	7.95E-8	3.18E-7	3.39E-7	7.39E-8
	Energy	10278.7	7734.30	9207.75	10925.8
Design III	Area	41520	20790	38589	43084
	Delay	59	52	29	19
	Noise	6.71E-8	2.68E-7	1.95E-7	6.15E-8
	Energy	10324.7	7946.15	9317.45	10416.26

TABLE 8.12: Delay improvements of the MWP-Bus compared with single shared bus and optimised single shared in Tables (8.8) to Table (8.11).

Compared with	Improvement (Min% - Max%)
Single Bus with uniform WL	14.05 - 53.52
Single Bus with optimised WL	5.92 - 50

TABLE 8.13: Delay improvements of the MW^{2}P-Bus compared with other synthesis methods in Tables (8.8) to Table (8.11).

Compared with	Improvement (Min% - Max%)
Single Bus with uniform WL	14.05 - 73.48
Single Bus with optimised WL	5.92 - 71.77
Single Bus with MWP-Bus	0 - 50.70

to (8.11) designs with two groups (and two bus segments). For designs with more number of groups MWP and MW^{2}P result in different implementation costs. Overall, the simulation results show that there is a considerable improvement in system latency (see Tables (8.12) and (8.13)) because of the MW^{2}P-BUS method, which is a valuable achievement. It must be reminded that MW^{2}P-BUS delay improvement is result of added parallel bus segments by which occupied buses can be bypassed (see chapter 5), therefore, we do not expect noise improvement by applying MW^{2}P-BUS compared with MWP-BUS. According to Table (8.8) to (8.11), MW^{2}P-BUS is more effective in the case of more complicated designs (in terms of the number of functional units and design complexity), which means that by increasing the number of functional units in

TABLE 8.14: Design configurations after MWP-Bus optimisation.

Designs	Groups	Binding I			Binding II			Binding III			Binding IV		
		W	*	+	W	*	+	W	*	+	W	*	+
Design I	1	15	1	1	15	2	2	15	2	2	15	3	2
	2	15	1	1	16	1	1	15	1	1	14	1	2
	3	-	-	-	-	-	-	16	1	1	15	1	1
Design II	1	15	1	1	15	1	1	14	2	2	14	2	2
	2	15	1	1	14	1	1	15	1	1	16	2	1
	3	-	-	-	14	1	1	14	1	1	14	1	2
Design III	1	15	1	1	15	2	2	15	2	2	15	3	2
	2	15	1	1	15	1	1	14	1	1	16	1	1
	3	-	-	-	-	-	-	15	1	1	15	1	2

the binding or complexity of the implemented benchmark, the effect of the bus switch costs reduces the total cost.

In the tables moving from MWP to MW^{2}P there is an improvement in speed because of parallel segments, which are used in MW^{2}P structure but error stays same which is expected regarding this fact that parallel segments should not affect computational noise in the circuit. There are couple of cases in the in which noise improves when MW^{2}P is applied, reason is the optimisation method. Since we have employed a genetic algorithm, which is based on random search for optimum results, it is possible to find slightly better results in some cases. Clearly, using exact methods of optimisation such as linear programming can solve this problem, however optimisation speed of such a methods is a severe problem.

To increase the readability of the results, Figures (8.9) to (8.10) graphically show the cost improvements comparing different methods for each design. It is attainable from graphs that using a single shared bus with optimised word-length gives cheapest designs, in terms of area and power consumption, but the slowest, where the MWP and MW^{2}P can decrease the latency with price of more area and power consumption.

In general, the optimisation time for this synthesis method can be impractically high due to the very big feasible space of the optimisation, greater than 10^{80} for a 50-node DFG for instance. As discussed in chapter 7 and also section 8.3, by choosing the uniform word-length design, which accomplishes the design constraint(s), as first generation in the genetic research; this search time decreases significantly. Optimisation times on a AMD Opteron (Dual) Processor 246 with 2.01 GHz clock frequency are measured for different designs in this chapter and the average values are presented in Table (8.16). According to the table, these times are dependent on the design, which is to be expected.

TABLE 8.15: Design configurations after MW^{2}P-Bus optimisation.

Designs	Groups	Binding I			Binding II			Binding III			Binding IV		
		W	*	+	W	*	+	W	*	+	W	*	+
Design I	1	15	1	1	15	2	2	15	2	2	15	2	2
	2	15	1	1	16	1	1	15	1	1	15	1	1
	3	-	-	-	-	-	-	16	1	1	16	1	1
	4	-	-	-	-	-	-	-	-	-	15	1	1
Design II	1	15	1	1	15	1	1	15	1	1	15	1	1
	2	15	1	1	15	1	1	16	1	1	15	1	1
	3	-	-	-	15	1	1	15	1	1	15	1	1
	4	-	-	-	-	-	-	15	1	1	15	1	1
	5	-	-	-	-	-	-	-	-	-	17	1	1
Design III	1	15	1	1	15	2	2	15	2	2	15	2	2
	2	15	1	1	15	1	1	15	1	1	16	1	1
	3	-	-	-	-	-	-	15	1	1	16	1	1
	4	-	-	-	-	-	-	-	-	-	15	1	1

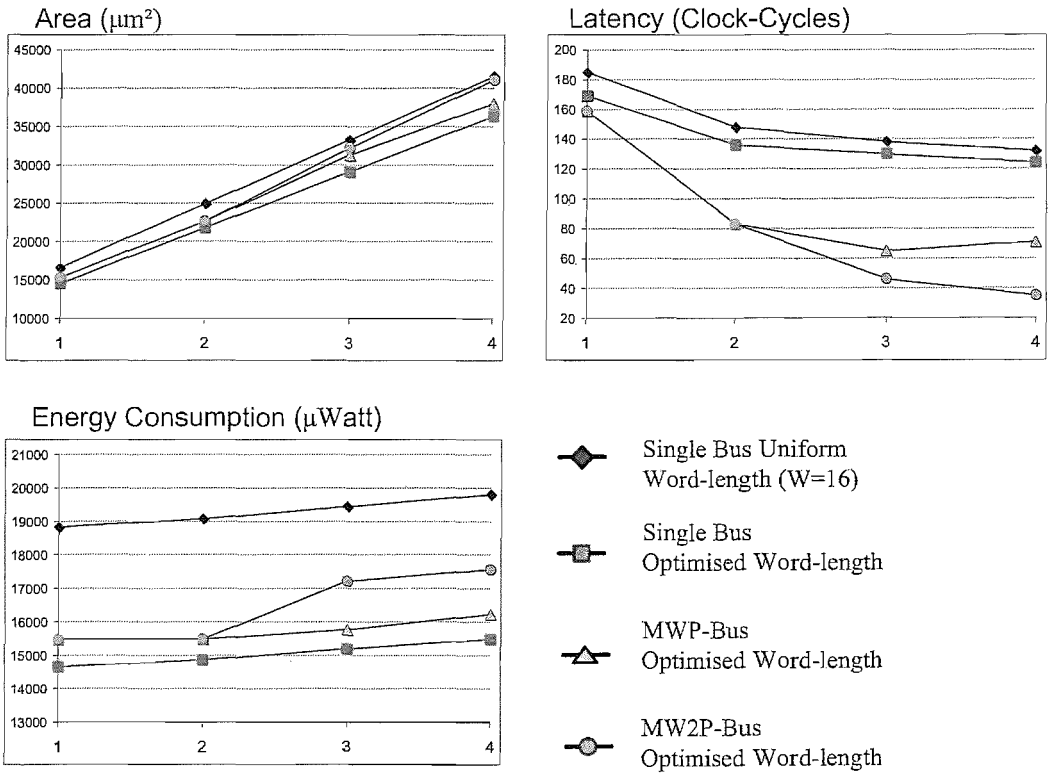


FIGURE 8.8: Optimisation results compare versus Bindings (1,2,3,4) for Design I.

8.5 Summary

Several case studies have been presented in this chapter to evaluate the proposed synthesis method. These implementations are separated into four: a design with more detail

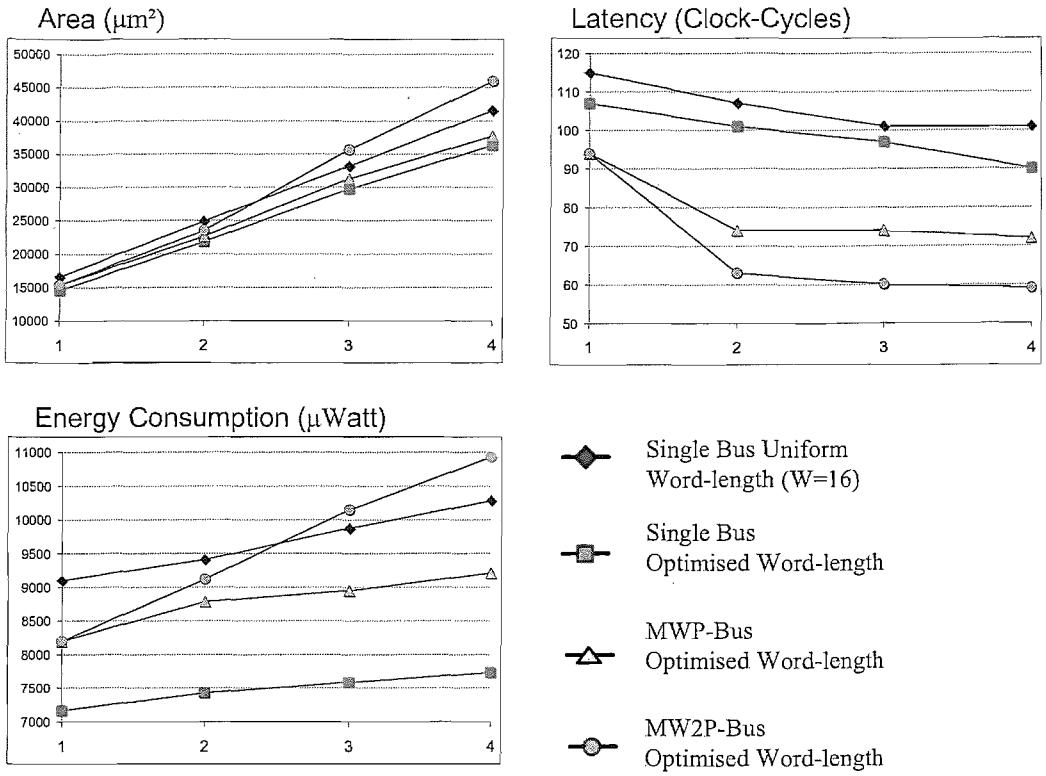


FIGURE 8.9: Optimisation results compare versus Bindings (1,2,3,4) for Design II.

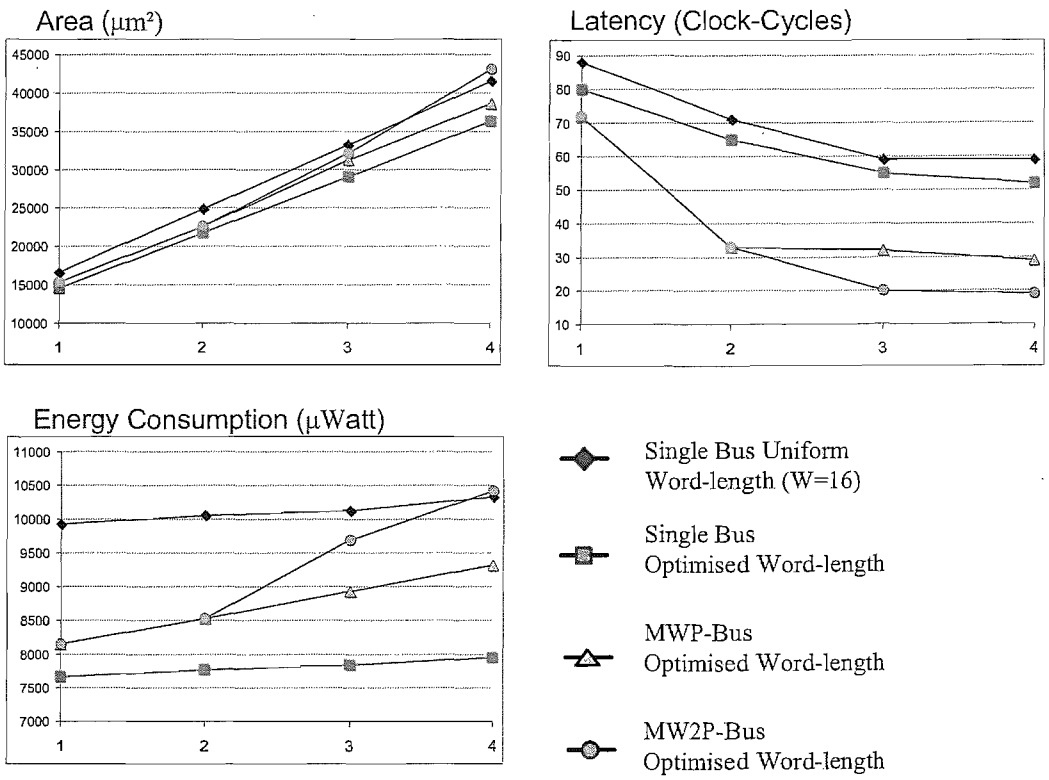


FIGURE 8.10: Optimisation results compare versus Bindings (1,2,3,4) for Design II.

TABLE 8.16: Optimisation time for different designs

Design	Optimisation time (Sec)
DCT	1467
Elliptic Filter	621
RGB to YCbCr	876

to show the effectiveness of the symbolic noise analysis method and three set of designs with different implementation structures.

It can be concluded from results that word-length optimisation has a considerable effect on design costs and specifically is more useful in the case of designs with more functional units. Furthermore, SNA as a computational error analysis method can provide comprehensive information about the output error. The iteration part of the Monte-Carlo simulation of the Black-Scholes equation is chose to be implemented by this method which results show lower error in the output with fewer resources usage. The single shared bus, partitioned bus (MWP-BUS) and MW^{2}P-BUS methods are utilised and compared for the other case studies. Our results show that latency of the circuits can be reduced considerably by using MWP-BUS and MW^{2}P-BUS methods, however these methods require more hardware resources because of their bus switches.

Chapter 9

Conclusions and Future Works

9.1 Conclusion

Application specific high level synthesis methods can produce more efficient designs because they are created with more focus on the specific features of the target designs. Hardware implementations of the computational tasks, regarding their special requirements such as high number of arithmetic operations can be considered as a distinct category with certain specifications. Two lines of applications can be envisaged for the computational hardware synthesis methods: computational circuits which need to be embedded in low-cost systems and hardware accelerators which can be attached to the systems with intensive computing tasks.

The work described in this thesis has proposed a high level synthesis method for computationally intensive hardware. To accomplish this goal, two issues have been addressed: the computational error analysis and the implementation structure. A detailed investigation on computational error analysis is performed which shows that in general substantial savings in design costs (area, energy and latency) can be achieved by taking the arithmetic characteristics of the functional units into account. Furthermore, it is shown that choosing an appropriate data communication structure is a vital parameter in the systems with high data exchange rates, accordingly, extended bus structures can improve datapath synthesis. As a result of exploring these subjects, novel approaches are introduced. The proposed method is restricted to a predefined architecture and takes the arithmetic characteristics of the functional units into account as an optimisation parameter.

In chapter 4 a new method for modelling computational error was presented which takes into consideration the intermediate error dependencies as well as the probability distribution function of the errors. In comparison with the current methods, which are either based on error range analysis or noise analysis, this method is more comprehensive in

terms of the information it can provide for the designer to select the arithmetic characteristics of the functional units and also in terms of its ability to deal with nonlinear systems. In addition the noise spectra at the output can be evaluated with controllable accurately to see the error distribution between its bounds. This method is applied to several examples and results show its power in practice.

Chapter 5 presented a new method of bus-oriented communication synthesis which is based on partitioned bus and parallel segments to provide data communications for functional units. Each bus segment, in this structure, can have a distinct width from other segments. This structure provides a very flexible data communication platform for computationally intensive datapaths with flexibility in datapath interconnections. The implemented synthesis method utilises the proposed bus structure as in the predefined architecture. Examination of the results demonstrates a considerable improvement in design costs when this structure is employed for synthesis and optimisation instead of general word-length optimisation methods.

In conclusion, the contributions of this work can be categorised into three:

From the EDA point of view, this work explored high level synthesis of computational intensive circuits considering accuracy along with the other synthesis objectives and optimising with one more extra dimension design space.

From the computational accuracy analysis viewpoint, a more precise and inclusive method was introduced which can be combined with the other synthesis cost evaluation techniques.

From the datapath synthesis viewpoint, a generalisation of the bus-oriented design was explored which provides an efficient and flexible method for communication synthesis in datapath level. This structure can be used as a hierarchical form to localising data processing in sub-groups of the system which can increase the system speed and flexibility.

9.2 Future of the Work

The present thesis could be elaborated in several directions. The following introduces some interesting and relevant areas of future research.

- High level synthesis for computational intensive systems,
- Implementation structure and the corresponding optimisation methods,
- Accuracy modelling tradeoffs,

The motivation for the first direction is the presented synthesis method in chapters 6 and 7 which combines four optimisation objectives. Concerning system specification, as

the complexity of the systems and underlying hardware continues to increase, the design cost for these have been raised drastically. Therefore, more comprehensive specification languages, such as SystemC or MATLAB, should be considered for design specification in the synthesis tool. This modification would extend compatibility of the methods with algorithm level design and simulation tools.

Another extension of the synthesis tool could be in the form of a C++ library which is attachable to C programs with heavy computational tasks. The idea is to automatically divide these kinds of program into two parts: the computational part which is mapped to external hardware (FPGAs for instance) and the rest of the program which remains in the software part. Since the computational part is specifically implemented in an optimum form on hardware, the execution speed of the application will be expected to be increased.

Furthermore, the proposed synthesis tool relies on a functional units library to provide the required basic blocks of the design, however, complicated functions such as elementary functions are application dependent and need to be designed and optimised precisely using some extra methods. I believe by combining the presented method with function evaluation methods it is possible to design more efficient computational systems with more clear information about their computational error.

Regarding the datapath synthesis method, this work is based on the MW^2P bus structure, which in the general form has a three dimensional structure. Since floorplanning method and the implementation technology has a great impact on the wiring complexity of the design, to provide a more accurate cost model for the bus structure these issues need to be considered. I suggest an investigation on place and route methods and data traffic rate between functional units (not just connection between them), because as discussed in chapter 5 data transfer between functional units needs to be taken into account in the design optimisation of the datapath dominant systems.

There is an interesting question regarding digital computation: why do general purpose processors, normally, offer higher precision computation compared with hardware implemented algorithms? General purpose processors provide a fixed number of computational resources which can be used flexibly by software. From a hardware point of view, it is a structure with a fixed datapath and a flexible controller. However, in all the synthesis and optimisation methods which utilise arithmetic characteristic as an optimisation parameter, only the datapath is taken into account and the controller has a rigid structure. In other words, it is generally assumed that the controller is independent from the computation accuracy and precision of the system, which is not true, bearing in mind the example of general purpose processors. Therefore, the next step in hardware implementation of algorithm must take the controller into consideration. To do so, in first step, a general structure for controller and its implementation cost, in terms of number of states, area and power consumption and so on is required. The

second step is decomposition of the complicated mathematical operations into sequences of simple building blocks which allows controller to have several choices to implement operations. And finally, combine the controller structures and models with datapath costs and models in to a synthesis tool. As a result, if the software complexity can be translated to more complicated controllers, higher precision computation can be provided. For example, a controller which can make decisions to reuse accurate functional units or combine several functional units to perform more precise calculations or uses different representations for data interchangeably can improve the computation accuracy. However design and implementation of such a controller is more expensive in terms of hardware resources and energy consumption. This issue can be subjected as a new research work in which controller and reconfigurability of the functional units are employed to trade off accuracy, latency and power consumption.

Appendix A

ICD Files Mnemonics

A.1 Introduction

This appendix presents more details about intermediate codes (ICD) syntax and grammar. ICD files have a very straightforward structure which is compatible with the proposed architecture in this study, see chapter 6, and can be considered as a compact representation format for Finite State Machines (FSMs).

A.2 Syntax

Table (A.1) gives a list of ICD codes. These codes simply provide the ability to present controlling signals in each state as well as structure of the controller FSM. A short explanation for each mnemonic is provided in the table.

A.3 Structure

ICD structure is as shown in Figure (A.1), which contains two parts: units declaration and sequence of control signals. In the first part, the under system is introduced which contains name, input and output pin numbers and word-length of the system and all the sub-blocks. System controller is specified in the second part, employing the basic capabilities of the ICD like: labelling, conditional and unconditional branching provides the ability of implementation of the algorithms. As it is discussed in section 6.3, controller can be implemented in two forms in the proposed architecture, structured and unstructured. Figure (A.1) represents a system with structured controller. An unstructured controller is depicted in Figure (A.2). Similar to the other structure, the heading of the file represents the name, number of inputs, number of outputs and bit-width of

TABLE A.1: ICD codes mnemonic.

	Mnemonic	Example	Description
1	U## name – ++ **	`U01 ADD 02 01 16`	System or Sub System name and No. of Inputs (-), Outputs (++) and word-length (**).
2	U##(–)	`U78(0)`	Signal activation, unit U##, signal number (–).
3	L–	`L05`	Label
4	X "-...-"	`X "01001"`	External Output Control Signals Setting, if '*d*' used it means that use the previous value.
5	WX–	`WX1`	Wait State Depending on External Control Signal Number (–).
6	W–(++)	`W06(0)`	Wait State Depending on Internal Control Signal No + of Sub-System No –.
7	GX–++	`GX0132`	Jump to ++ (L++) depending on External Control Signal No –.
8	Gddd++	`Gddd32`	Jump unconditionally to ++ (L++).
9	G–++**	`G120132`	Jump to ** (L**) depending on signal No ++ of sub system No –.
10	RLX	`RLX`	Relaxation state.
11	Xd =	`Xd=`	Unstructured scheduling description Branch.
12	X= –	`X= 00`	Structured scheduling description Branch - according to input external control signals.
13	END	`END .`	End of the specification.
14	d	`Gddd00`	Don't care or consider the previous value if needed.
15	,	`U00(0) , U05(1) ;`	End of the operation.
16	;	`; RLX ;`	End of the sate.
17	.	`END.`	End of the branch
18		`WX1 X "00" ;`	Conditional control signals set, IF part.
19		`WX1 X "01" ;`	Conditional control signals set, ELSE part.

IO of the system. The second line introduces the controller name and number of input and outputs of it. Third part gives the datapath information which contains the name, functionality, number of inputs, number of outputs and bit-width of the functional units. The last part of the file shows the controlling signals specification, which indicates the activated control signals in each state of the controller. The second part of the synthesis method is the ICD2VHD program which translates the ICD files to synthesisable VHDL, see chapter 6 for more information regarding the synthesis procedure. Two more examples are provided in the Listings (A.1) and (A.2).

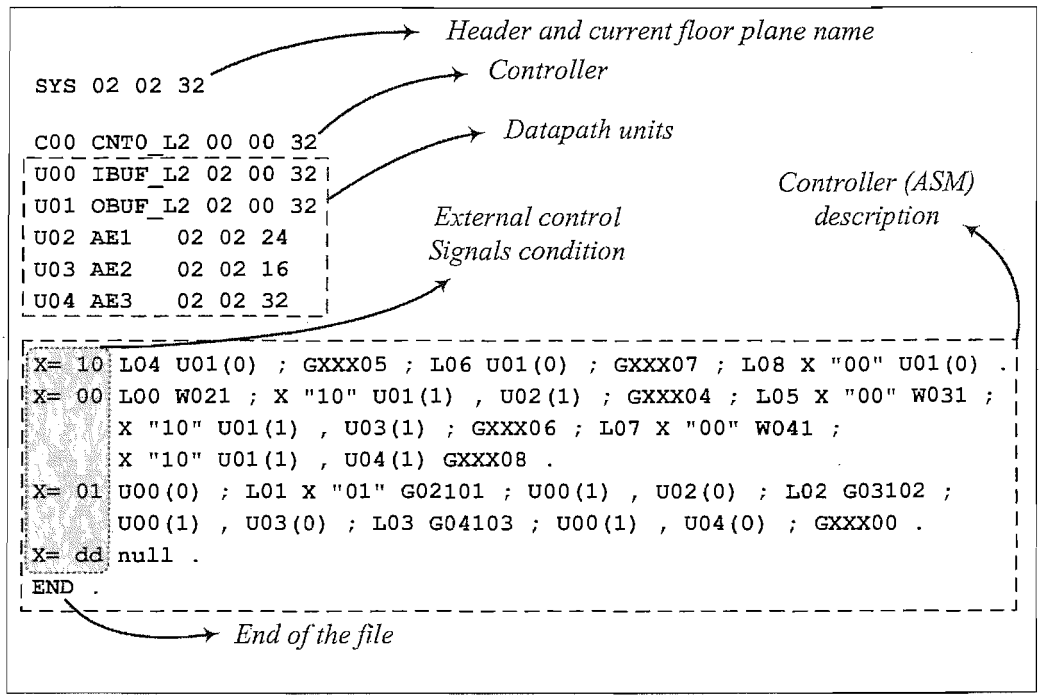


FIGURE A.1: General format of the ICD file for a design with three sub-systems, structured controller.

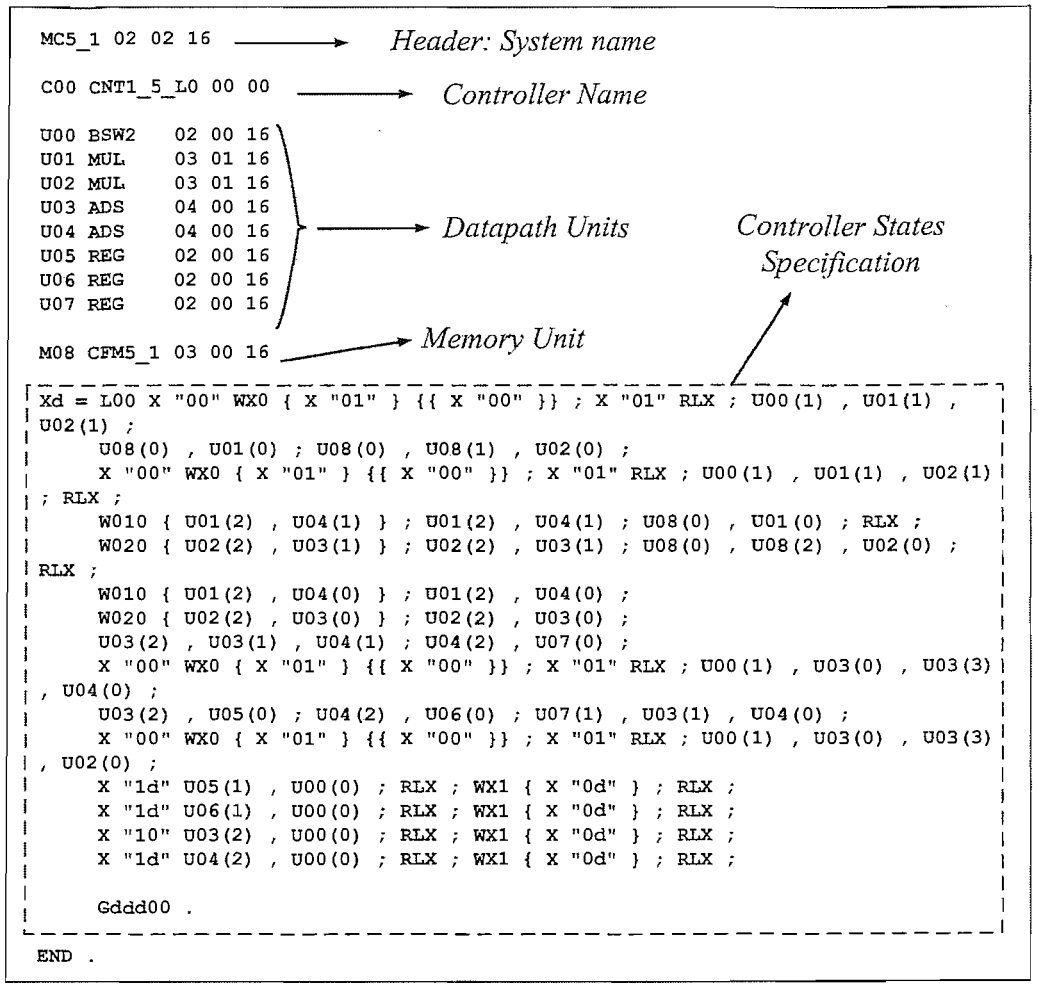


FIGURE A.2: ICD specification of the macro-cell for a matrix based transform (see appendix B), unstructured controller form.

Listing A.1: structured controller ICD file, highest level of specification.

```
AE1 02 02 16
C00 CNT0_1_L1 00 00 16
U00 IBUF_L1 01 00 16
U01 OBUF_L1 01 00 16
U02 MC1_1 02 02 16
Xd = L00 X "00" WX0 { X "01" } {{ X "00" }} ; X "01" RLX ; U00(0) , U02(0) ;
    RLX ; W021 ; X "10" U02(1) , U01(0) ; RLX ; WX1 { X "00" } ; RLX ;
    Gddd00 .
END .
```

Listing A.2: structured controller ICD file, macro-cell (MC) level of specification

```
SYS 02 02 32
C00 CNT0_L2 00 00 32
U00 IBUF_L2 02 00 32
U01 OBUF_L2 02 00 32
U02 AE1 02 02 16
U03 AE2 02 02 24
U04 AE3 02 02 32
X= 10 L04 U01(0) ; GXXX05 ; L06 U01(0) ; GXXX07 ; L08 X "00" U01(0).
X= 00 L00 W021 ; X "10" U01(1) , U02(1) ; GXXX04 ; L05 X "00" W031;
      X "10" U01(1) , U03(1) ; GXXX06 ; L07 X "00" W041 ;
      X "10" U01(1) , U04(1) GXXX08 .
X= 01 U00(0) ; L01 X "01" G02101 ; U00(1) , U02(0) ; L02 G03102 ;
      U00(1) , U03(0) ; L03 G04103 ; U00(1) , U04(0) ; GXXX00 .
X= dd null .
END .
```

Appendix B

Supplementary Examples

In general, the proposed architecture exerts a set of restrictions on the high level synthesis methodology. However these limitations have improved the synthesis process, generality and ability of the architecture might be in question. To evaluate the flexibility of the architecture and its ability to run basic DSP algorithms some of the more public and basic algorithms are mapped to the architecture in this appendix. The first part of the section is a survey on the basic algorithms and the second part presents the details of the implementation of some algorithms on the architecture.

Signal processing algorithms originate from pure mathematical methods to analyse the functions. There is a wide variety of DSP algorithms which have been divided into different categories from different points of view ([132]), but since algorithms application and mathematical properties are out of this study concern, they have been considered in two broad categories: Filtering and Transforms. Following subsections give more details in both.

B.1 Filtering

In general a system, and most signal transforms, can be described in the form of a difference equation as illustrated in Equation (B.1).

$$F\{x, f(x), f(x+h), f(x+2h), ..., f(x+nh)\} = 0, \tag{B.1}$$

where, F can be a nonlinear time variant function. In the simplest case where F is linear and time invariant, LTI system [98], the relationship between input, x and output y is:

$$\sum_{k=0}^{N} a_k \cdot y[n-k] = \sum_{k=0}^{M} b_k \cdot x[n-k], \tag{B.2}$$

And by using Z transform ($Z\{\}$, see [98] or [105] for details), we have:

$$\mathbf{Z}\{h[n]\} = H(z) = \frac{\sum_{k=0}^{N} a_k \cdot z^{-k}}{\sum_{k=0}^{M} b_k \cdot z^{-k}} \tag{B.3}$$

where $H(z)$ can be expanded in the form of multiplication of first order farctions to build up a case-cade structure as

$$H(z) = A \cdot \frac{\prod_{k=0}^{M} (z_k - z^{-1})}{\prod_{k=0}^{N} (p_k - z^{-1})} \tag{B.4}$$

or be expanded as addition of second order fractions as

$$H(z) = \sum_{k=0}^{Max\{\frac{N}{2},\frac{M}{2}\}} \frac{b_{2k} + b_{1k} \cdot z^{-1} + b_{0k} \cdot z^{-2}}{a_{2k} + a_{1k} \cdot z^{-1} + a_{0k} \cdot z^{-2}}, \tag{B.5}$$

Some very popular examples of these systems are filters.

From one point of view, filters (systems) can be divided into two categories: Finite-Duration Impulse Response (FIR) and Infinite-Duration Impulse Response (IIR) filters. From the implementation point of view, on the other hand, a FIR filter has no feedback but an IIR filter has feedback(s) and its computation operation is recursive which increases the complexity and possibility of instability in comparison with FIR system. There are a lot of related works in this regard [98], specially in design and implementation of filters as the most popular systems in practice; here we review some of the most popular.

B.1.1 Related Algorithms and Models

FIR systems have a finite duration Impulse Response, or Unit Sample Response in discrete time systems. Their general description by difference equations is:

$$y[n] = \sum_{k=0}^{M-1} b_k \cdot x[n-k], \tag{B.6}$$

Alternatively, by a transfer function in the form of:

$$\mathbf{Z}\{h[n]\} = H(z) = \sum_{k=0}^{M-1} b_k \cdot z^{-k}, \tag{B.7}$$

Furthermore, the unit sample response of the FIR system is identical to the coefficients $\{b_k\}$, that is:

$$h[n] = \begin{cases} b_n & 0 \leq n \leq M-1 \\ 0 & otherwise, \end{cases} \tag{B.8}$$

The length of the FIR system is denoted as M to conform to the established notation in the technical literature. Several methods for implementing an FIR filter have been introduced which we recite them here:

- Direct Form Structure
- Cascade Form Structures
- Frequency Sampling Structures
- Lattice Structure

IIR systems have an impulse response with infinite duration in time space. The rational system function that characterizes an IIR system is:

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k], \tag{B.9}$$

This also can be expressed as a system function like:

$$\mathbf{Z}\{h[n]\} = H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}, \tag{B.10}$$

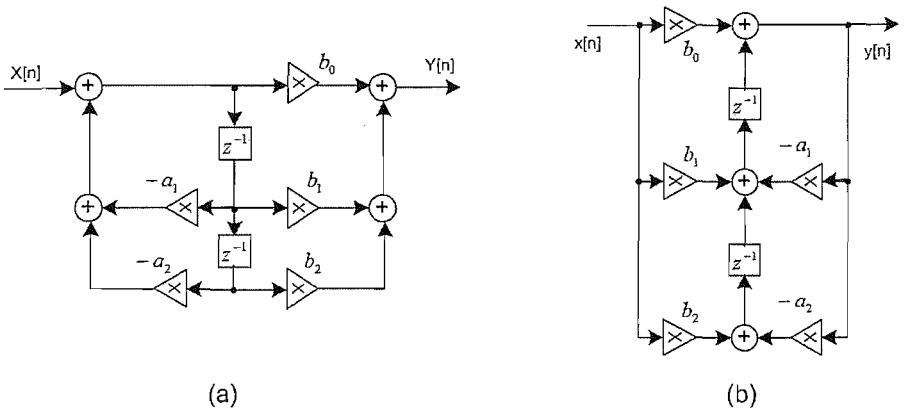
Similar to FIR systems, there are several types of realisations structures including:

- Direct Form Structures
- Cascade Form Structures
- Parallel Form Structures
- Lattice Structure
- Lattice-Ladder Structure

There are other structures for IIR systems like: wave digital structures, frequency-response masking and all-pass-based forms, etc [31, 98]. They all represent various tradeoffs; the best choice in a given context is not yet fully understood, and may never be [98]. Each method according to its implementation costs has advantages and disadvantages. For example, parameters like Complexity, Memory Requirements, Speed, Finite Word-length Effects and Flexibility [101].

Since IIR filters have a more general form, they are considered in this design example. Three basic structures of an order-2 (two pole) IIR filter have been shown in the Figure

(B.1). By manipulating the general form of an IIR system simply we can reduce the design of a high order system $(m, n > 2)$ to the design of two pole system by which ,as a building block of the larger system, we can make a more complex systems using them in cascade or parallel form or a mixed structures Figure (B.2).



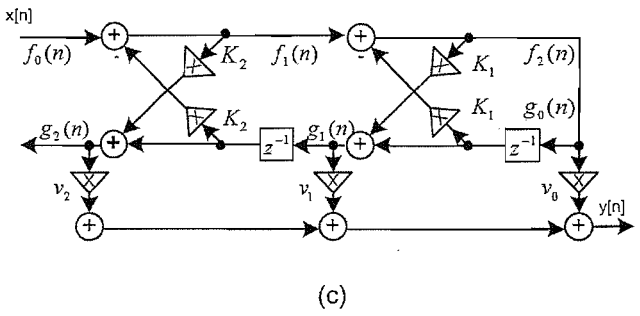


FIGURE B.1: Implementation of general tow-pole IIR system a)simple form b)transposed form c)lattice form.

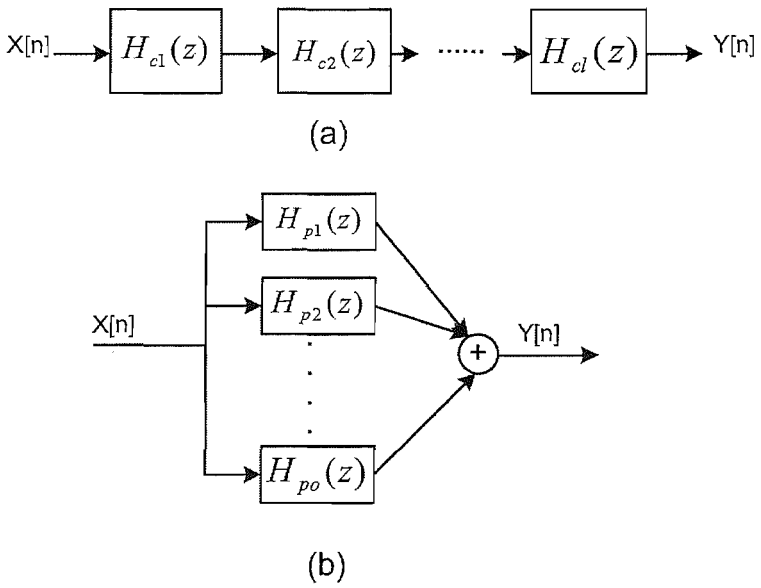


FIGURE B.2: High order filters can be decomposed in a)cascade form b)parallel form.

B.2 Transforms

Since Fourier transform of signals has a great impact on signals and systems analysis, many different kind of computation methods have been introduced and discussed for it [98]. According to complexity parameters and resources restrictions, these computation methods have different efficiencies. This section discusses the basis of some public methods for DFT realisation and afterward an implementation for each has been designed.

There are different algorithms of DFT implementations. Here, as a case study, three basic algorithms have been discussed and implemented: Goertzel Filters [31], FFT and Matrix Based method.

B.2.1 Goertzel Algorithm

A computation procedure that is somewhat more efficient than the direct method is called Goertzel Method (or filters). In this method the DFT can be viewed as the response of a filter [31]. DFT of sequences $x[n]$ is:

$$X[n] = \sum_{k=0}^{N-1} x[k] \cdot W_N^{nk}, \tag{B.11}$$

where $W_N^{nk} = e^{-j\frac{2\pi k}{N}}$. Clearly $W_N^{nN} = e^{-j\frac{2\pi nN}{N}} = 1$ so the DFT equation can be rewriten as:

$$\begin{aligned} X[n] &= W_N^{nN} \cdot \sum_{k=0}^{N-1} x[k] \cdot W_N^{nk}, \\ &= \sum_{k=0}^{N-1} x[k] \cdot W_N^{-n(N-k)}, \end{aligned} \tag{B.12}$$

For convenience, let us define a new sequence:

$$y_n[m] = \sum_{k=0}^{N-1} x[k] \cdot W_N^{-n(m-k)}, \tag{B.13}$$

where it can be seen that $X[n] = y_n[N]$. The above equation shows that $y_n[m]$ is the convolution of $x[n]$ with the sequence $W_N^{-km)}$. Consequently, $y_n[m]$ can be considered as the response of a system with an impulse (unit sample) response of $W_N^{-km)}$ and $X[n]$ is the value of the output when $m = N$. The signal flow graph of a system with unit sample response $W_N^{-km)}$ is depicted in Figure (B.3).

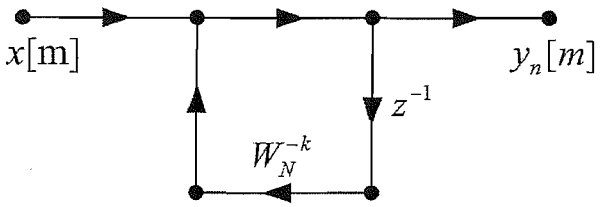


FIGURE B.3: Signal Flow Graph of first order Goertzel filter.

System function of the first order Goertzel filter is:

$$H_n(z) = \frac{1}{1 - W_N^{-n} z^{-1}}, \tag{B.14}$$

from which, by multiplying both numerator and denominator of $H_n(z)$ by the factor $(1 - W_n^N z^{-1})$, we obtain:

$$\begin{aligned} H_n(z) &= \frac{1 - W_n^N z^{-1}}{(1 - W_N^{-n} z^{-1})(1 - W_n^N z^{-1})}, \\ &= \frac{1 - W_n^N z^{-1}}{1 - 2\cos\left(\frac{2\pi}{N} n\right) z^{-1} + z^{-2}}, \end{aligned} \tag{B.15}$$

This is a second order system with signal flow graph of the Figure (B.4). The Goertzel

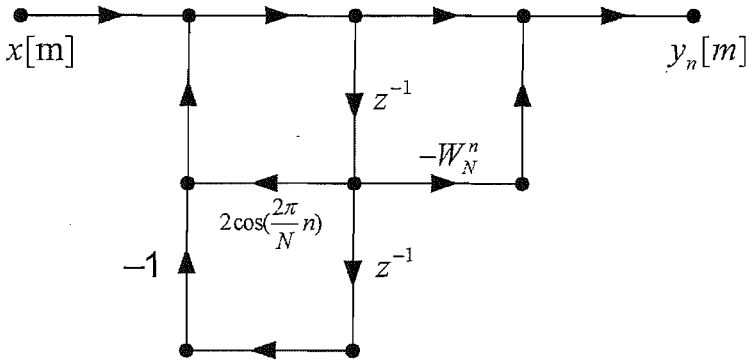


FIGURE B.4: Signal Flow Graph of second order Goertzel filter.

algorithm is more efficient than the FFT when only a few samples of DFT ($X[n]$) are required.

B.2.2 Fast Fourier Transform

The Fast Fourier Transform (FFT) is an important tool used in digital signal processing applications. The definition of the FFT is identical to the DFT, only the method of computation differs. From a matrix based view point, a DFT transformation of an N-point input ($x[n]; n = 0, 1 \cdots N-1$) takes a vector of N coefficients and returns a vector with its values evaluated at N points. Namely this transform is evaluated at the N roots of unity, which are evenly spaced on the unit circle. Since the relationship between input

and output is linear, the Fourier transform must be given by a matrix multiplication like:

$$X[n] = F_N \times x[n], \tag{B.16}$$

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ . \\ X[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & . & 1 \\ 1 & W_N^1 & W_N^2 & . & W_N^{N-1} \\ 1 & W_N^2 & W_N^3 & . & W_N^{2(N-1)} \\ . & . & . & . & . \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & . & W_N^{(N-1)^2} \end{bmatrix} \times \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ . \\ x[N-1] \end{bmatrix}$$

where the $W_N = e^{-j\frac{2\pi}{N}}$ is the i$^{\text{th}}$ participate of the N$^{\text{th}}$ root of 1, F_N is the $N \times N$ Fourier matrix and $X[n]$ is the DFT transformed of $x[n]$. FFT is based on decomposition of the matrix multiplication to smaller matrixes which can affect the computational complexity dramatically.

To achieve the efficiency of an FFT, it is important that N be a highly composite number, which is typically considered as a power of 2 ($N = 2^M$). Consequently, the whole algorithm breaks down into a repeated application of an elementary transform known in a *butterfly* structure. Figure (B.5) shows the general form of radix-2 FFT butterfly, more details can be found in DSP text books such as [98] or [105].

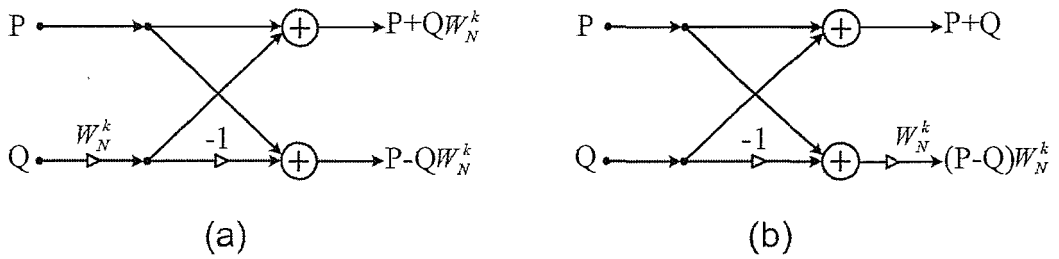


FIGURE B.5: Raxid-2 Butterfly Cell for FFT a)DIT Implementation b)DIF Implementation.

The crucial difficulty of FFT implementation is its inputs and outputs data access and arrangement. For example, consider Figure (B.6 which demonstrates an 8-point FFT implementation in different ways. Apparently, data passing between butterfly blocks makes an unavoidable computational overhead especially in a single bus systems.

Over the years, researchers have developed different forms of FFT for more efficient computation [107]. Radix-2 Butterfly is the smallest element for FFT implementation. The radix of FFT presents the number of inputs that are combined in a butterfly. FFT is usually explained around the radix-2 algorithm for simplicity; however, using high order radices saves more computational resources. This saving increases with radix, but there is a very little improvement above radix-4. In radix-4 FFT, each butterfly has 4 inputs and 4 outputs, essentially combining two stages of a radix-2 algorithm in one, see Figure (B.7).

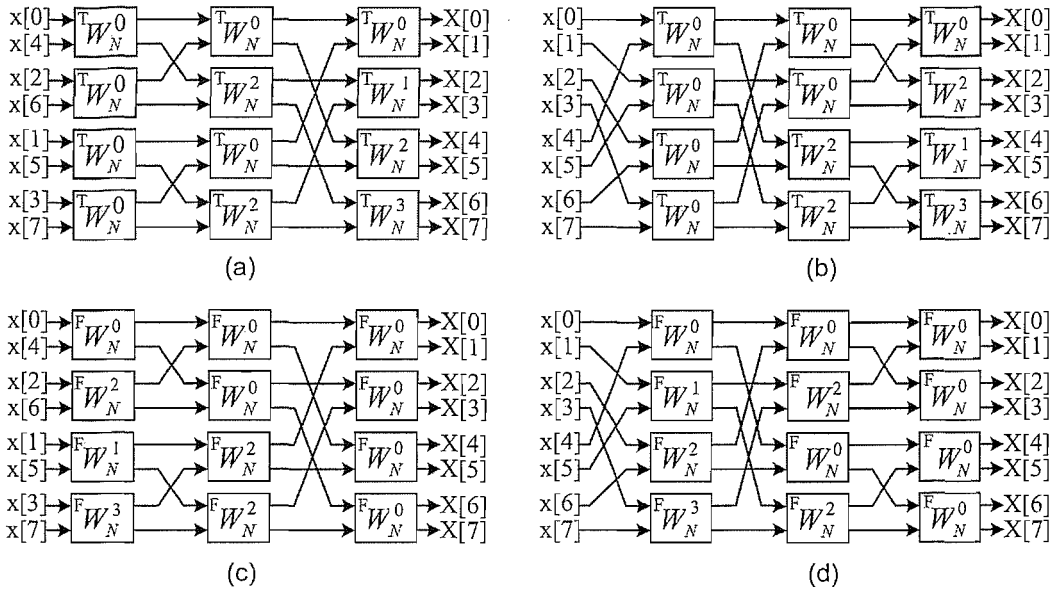


FIGURE B.6: 8-point implementation of FFT by radix-2 Butterfly cells a)DIT bit-reversed ordered inputs b)DIT normal ordered inputs c)DIF bit-reversed ordered inputs d)DIF normal ordered inputs (each block is a radix-2 butterfly cell) [98].

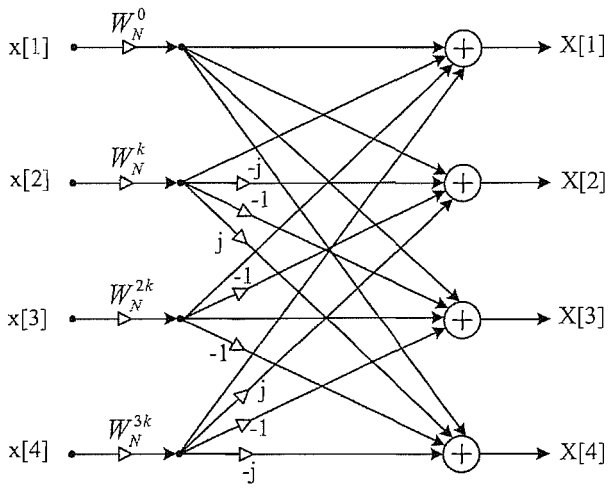


FIGURE B.7: Radix-4 Butterfly Cell for FFT implementation.

The matrix equation for a radix-4 FFT cell is:

$$\begin{bmatrix} X[1] \\ X[2] \\ X[3] \\ X[4] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \times \begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ x[4] \end{bmatrix} \tag{B.17}$$

and by separating the real and imaginary parts of the equation we see:

$$\begin{bmatrix} X[1] \\ X[2] \\ X[3] \\ X[4] \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix} + j \times \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \right) \times \begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ x[4] \end{bmatrix} \tag{B.18}$$

where by rewriting the equations we have:

$$\begin{cases} X[1] = P_1 + Q_1 \\ X[2] = P_2 - jQ_2 \\ X[3] = P_1 - Q_1 \\ X[4] = P_2 + jQ_2 \end{cases} \quad \text{where} \quad \begin{cases} P_1 = x[1] + x[3] \\ P_2 = x[1] - x[3] \\ Q_1 = x[2] + x[4] \\ Q_2 = x[2] - x[4] \end{cases} \tag{B.19}$$

In general the same structure of macro cells (MCs) can be used for radix-4 FFT implementation, however, the macro cells' internal scheduling and coefficients must be changed. If for some reasons, in tone detection systems for instance, only a few samples of the DFT are needed; the computational complexity can be reduced by omitting the non required frequencies computation tree. It means deleting the butterfly cells which have no effect on the required frequencies DFT computation.

B.2.3 Matrix Based Method

However it is possible to implement both the aforementioned methods by the proposed architecture, according to matrix expansion, there other ways to compute DFT by our proposed architecture depending on data conditions. Basically matrix multiplication is a set of multiplications and accumulations which can be implemented either in serial or parallel form. So we can write a matrix multiplication like Equation (B.20).

$$\begin{aligned} C_{M\times N} &= A_{M\times L} \times B_{L\times N}, \\ &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdot & a_{1L} \\ a_{21} & a_{22} & a_{23} & \cdot & a_{2L} \\ a_{31} & a_{32} & a_{33} & \cdot & a_{3L} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{M1} & a_{M2} & a_{M3} & \cdot & a_{ML} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & \cdot & b_{1N} \\ b_{21} & b_{22} & b_{23} & \cdot & b_{2N} \\ b_{31} & b_{32} & b_{33} & \cdot & b_{3N} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ b_{L1} & b_{L2} & b_{L3} & \cdot & b_{LN} \end{bmatrix} \end{aligned} \tag{B.20}$$

where $c_{ij} = \sum_{k=1}^{L} a_{ik} \cdot b_{kj}$. Having no special order in this computation, there are many possibilities for its implementation, see Figure (B.8). Decomposition to sub-blocks for instance which has been used in FFT where sub-blocks dimensions are 2 in radix-2 FFT or 4 in radix-4 FFT.

Due to the coefficients, special form, FFT implementation can be carried out over its rows and columns in an efficient way as well. In FFT the first matrix (A in Equation

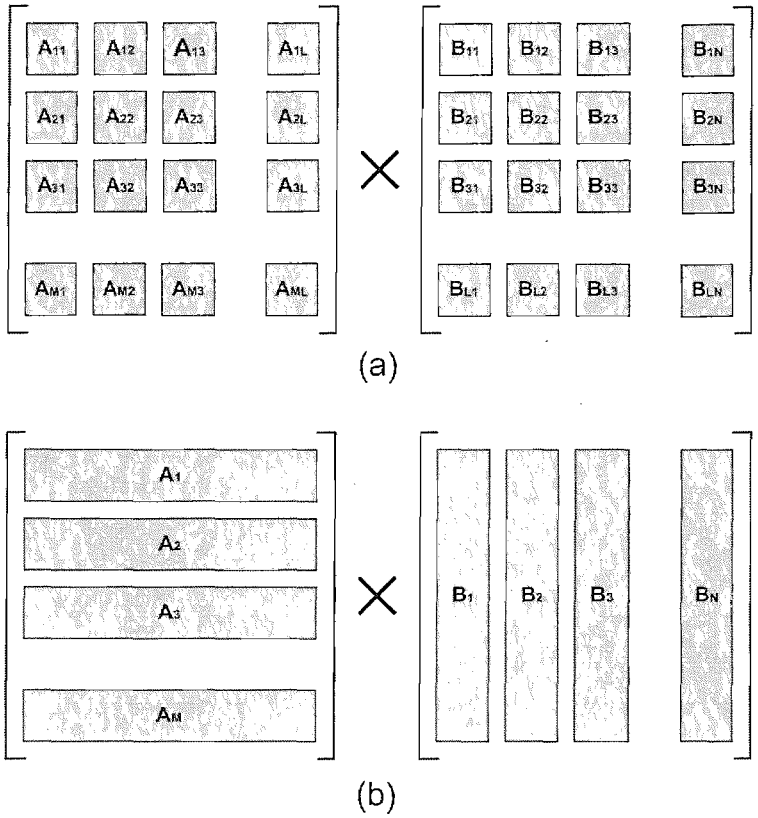


FIGURE B.8: Matrix decompositions for multiply implementation, a)Decomposition into sub-blocks, b)Decomposition into rows and columns.

(B.20)) is an $N \times N$ symmetric matrix in which its elements have special characteristics and the second matrix (B) is an $L \times 1$ matrix (a vector of length L). Exploiting these characteristics can make the computation very simple. In addition, depending on the data input, hardware resources can be managed to best usage. We considered each one separately.

Another vital problem of this kind of signal transforms is their massive input and output data. Two approaches can be imagined: data stream and stored data. In the first one a time spaced sequence of numbers will be the system input and in the second one it supposed that data has been stored, or cached, previously in a local memory.

Stream input:

In this approach inputs will be received one by one, which means each time we have $x[n]$ received at that time and preceding values. So it seems better to calculate the effect of that input on all inputs and wait for another input which means multiplying the corresponding row of in input and accumulate it with the previous results. See Figure (B.9).

By using this method one input of MC blocks is $x[i]$ always a real number. An interesting positive point of this design is the F_N elements which are called twiddles [105]. There is

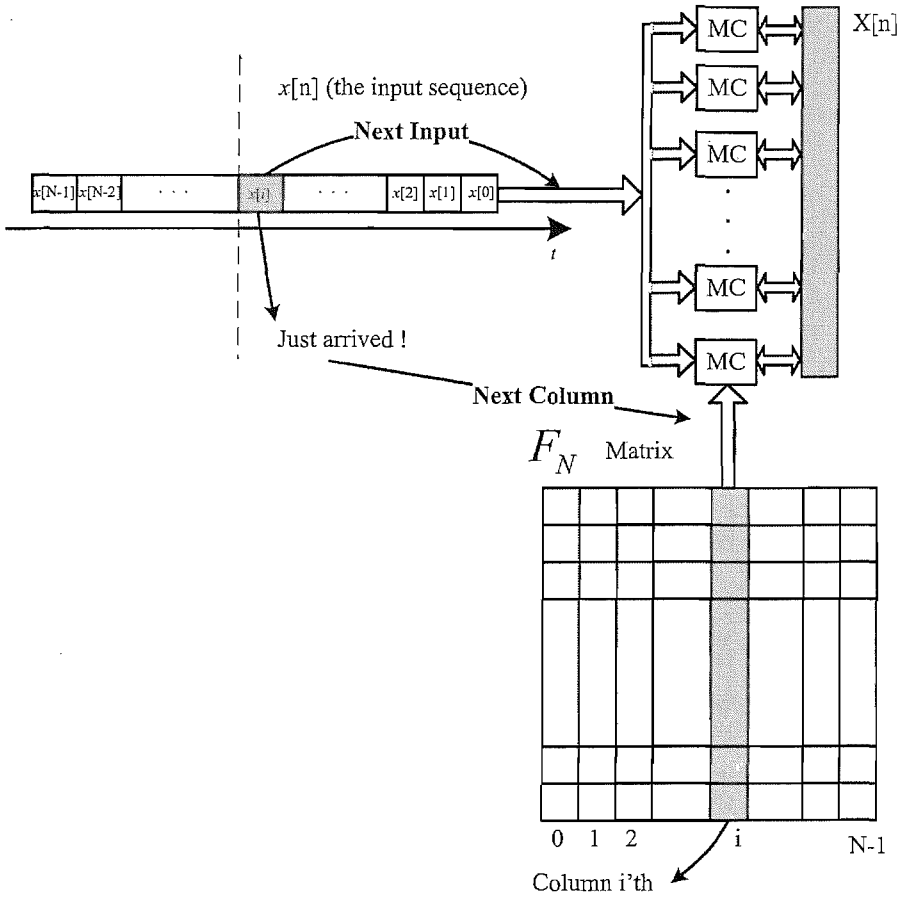


FIGURE B.9: The basic structure for Stream data vector-matrix multiplication.

a relationship between elements of each of the two neighbour columns of F_N by which the hardware can compute the next column for the next input faster. We can see:

$$a_{i(j+1)} = a_{i1} \cdot a_{ij} \qquad \text{where} \qquad i, j = 1, 2, 3, \cdots, N-1 \tag{B.21}$$

In other words, by having a_{i1} and multiplying it by each column we can find the next column values.

To make the design more efficient, we can reverse the sequence of input data, so a recursive formula will be achieved for $X[n]$ like:

$$X^i[n] = \left(X^{i-1}[n] + x[i]\right) \cdot F_N^1[n], \tag{B.22}$$

where $F_N^1[n] = W_N^n$. $n = 0, 1, 2, \cdot, N-1$ is the second column of the F_N and $X[n] = X^i[n]|_{i=N-1}$. Figure (B.10) shows a schematic implementation of this algorithm. The weak point of this design is that in the large N-point DFT, we need a large number of MCs, which can be impractical. To deal with this problem we can divide the $X[n]$ array into several sub-arrays with equal lengths and at each time compute just one of them.

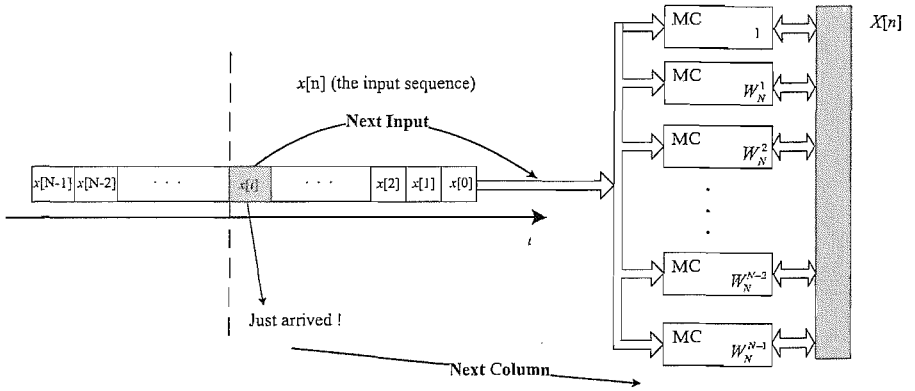


FIGURE B.10: Twiddle factors in the MC blocks for data stream inputs.

Stored data input:

In most of the hardware implementations of the DFT, it is supposed that the input data (vector) is stored in a local memory and is ready to use. However this assumption is reasonable and makes all $x[n]$ elements accessible, there is a massive data communication scheme in these kinds of structures. It means that, data reading and saving is a bottle-neck of the design efficiency. We again refer the reader to the basic matrix representation of DFT, here we have all $x[n]$ and the system can produce $X[n]$ elements simultaneously, see the Figure (B.11).

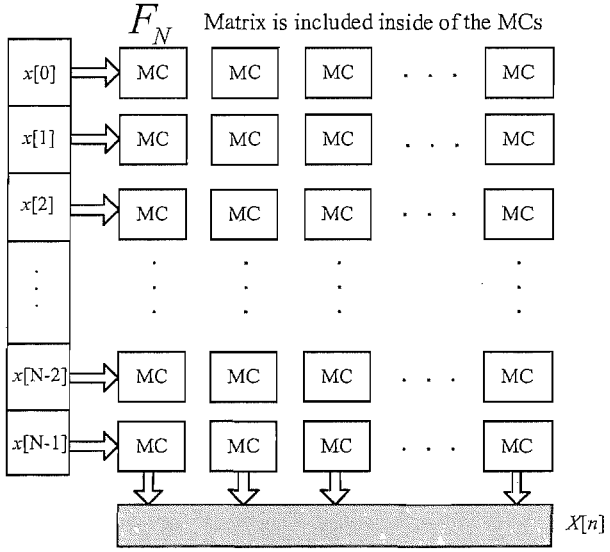


FIGURE B.11: Stored data DFT computation method.

Depending on the implementation structure, the number of MCs can be reduced. For example one may prefer to produce the $X[n]$ elements one by one, which is interesting when DFT in needed only in some frequencies; therefore, at each time, only one row of the matrix will be used and the number of required MCs will be reduced dramatically, see Figure (B.12-a). Another method is similar to the data stream processing method,

by which the input vector can be fed to the MCs from the end to start and as a result the number of MCs will reduced, see Figure (B.12-b).

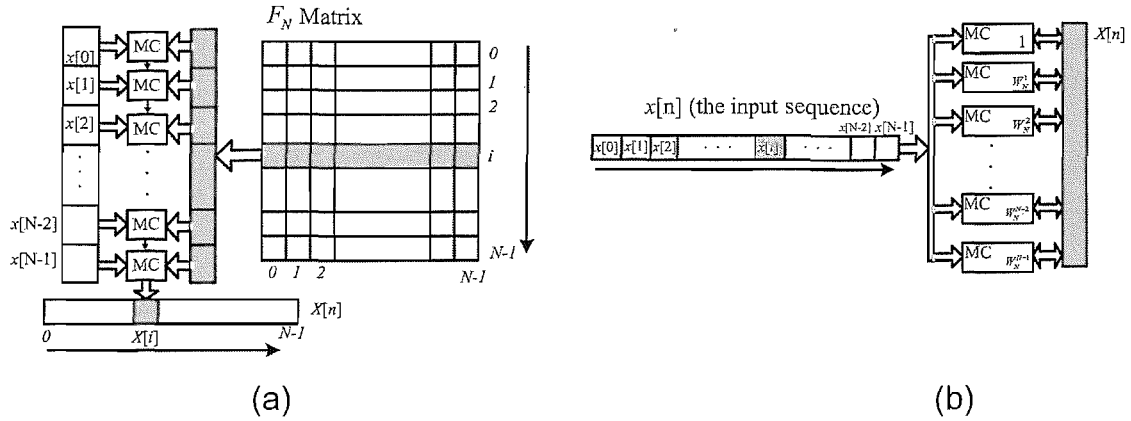


FIGURE B.12: Two basic structure for stored data input which produces one $X[i]$ each time. a)Computing DFT signal ($X[n]$) one by one b)Feeding input signal one by one.

B.3 Implementation

The following subsections give details of design and implementation of the basic algorithms which discussed in sections B.1 and B.2.

B.3.1 Filters

System coefficients (a_k and b_k in Equation (B.10)) can be implemented as firmware or software. If they are implemented as firmware it may increase the speed, decrease the complexity and programming delay time and also it is possible to simplify some functional blocks, for example multipliers which always multiply a constant to input data. On the other hand using software increases systems flexibility.

As Figure (B.13) shows, the system has been divided into sub-blocks; each block has two poles and two zeros. These sub-blocks should be able to be connected together in serial, parallel or mixed in form, and because of this there are some adders. All these sub-blocks are controlled by a controller. The controller produces sequencing signals for two pole systems and adders, these sequences can be implemented in firmware or software.

Inside each two pole system, there are some functional blocks like addition, multiplication and registers. As discussed in the algorithm's model there are many ways to implement a system by these functional blocks. There is a dilemma here, whether one of these structures should be chosen as the basic structure or these subsystems should have an arbitrary structure. By making these subsystems programmable, we provide

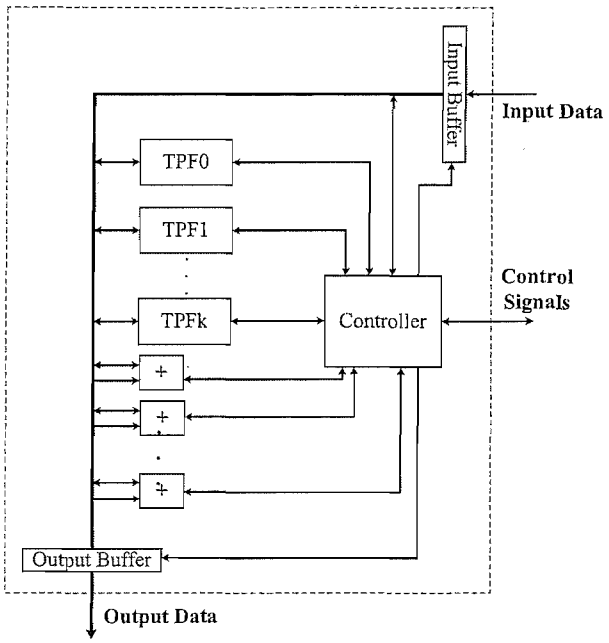


FIGURE B.13: Block diagram of general 2k-Pole System.

the capability of using these sub-blocks as the basis of other algorithms which use the Multiplier Accumulator (MAC) basic cell.

Clearly, according to IIR systems' implementation algorithms and structures, the building blocks of each Two Pole Filter (TPF) are:

- Multiplier
- Adder
- Delay Unit (Flip-Flops or Registers)
- Bus switches

In addition, if the filters coefficients are considered as well, we need another memory unit for them. The parameters that lead us to choose one of these structures are: the maximum memory requirement, critical path of the circuits, complexity (number of adders and multipliers which required) and the required accuracy of computation (word-length). The lattice architecture of IIR filters is not suitable in this case because it was initially designed to approximate high order filters, so it is not suitable for the two pole filters.

Figure (B.14-a) shows the basic implementation of a simple two-pole filter. Numbers indicate the scheduling sequence of operations and it means that in the best case this algorithm takes:

$$T_{TPF} = 3T_{ADD} + 2T_{MUL}, \tag{B.23}$$

assuming that $T_{MEM} \leq T_{MUL}$, where T_{ADD}, T_{MUL} and T_{MEM} stand for delay of adder, multiplier and the memory modules, respectively. Moreover, it has been supposed that the multipliers have a simple structure, with one constant input and another variable input. Figure (B.14-b) shows the hardware realisation of transposed form of TPF. The computation delay for Transposed TPF, according to Figure (B.14-b), is:

$$T_{TPF} = T_{ADD} + 3T_{MUL} + T_{MEM}, \qquad T_{MUL} \geq T_{ADD} \tag{B.24}$$

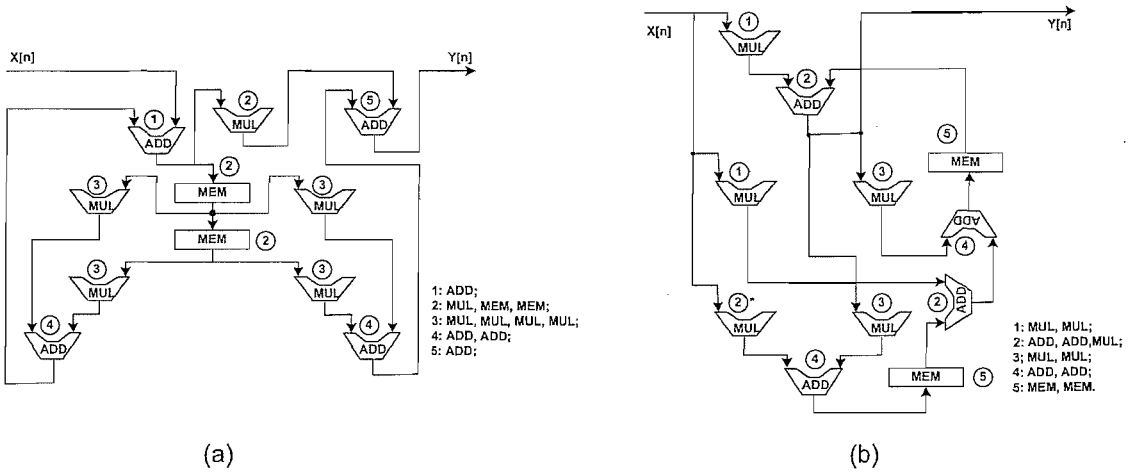


FIGURE B.14: Hardware realisation of TPF, numbers shows the sequence of operations. a)Simple form b)Transposed form. * This operation can be moved to stage 1 as well.

This design (we will call it type-1) needs two multipliers, two adders, two registers and other related circuits. Optionally there is another option (type-2) in which we move the second's stage MUL to first step. Using type-2 scheduling scheme means one more multiplication in the first stage and its delay will be:

$$T_{TPF} = 2T_{ADD} + 2T_{MUL} + T_{MEM}, \qquad T_{MUL} \geq T_{ADD} \tag{B.25}$$

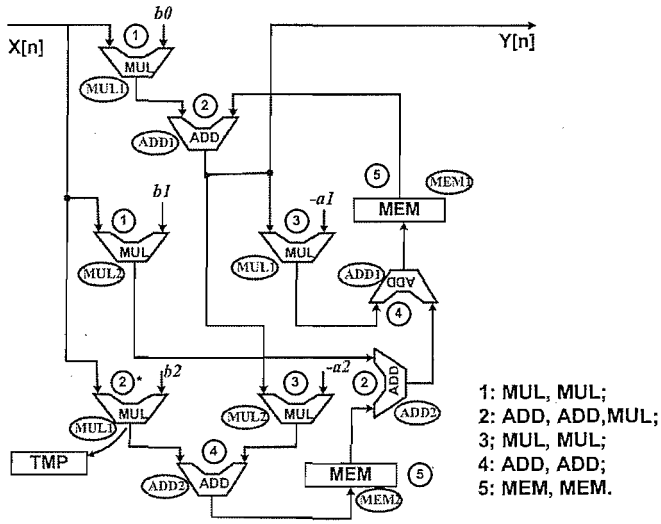
Implementation of the TPF by type-2 scheduling seems faster in comparison with type-1, but in turn it needs one multiplier more than type-1, thus we will use type-1 in our implementation. Figure (B.15) shows a block diagram of such a design. The sequencer, in Figure (B.15), produces required signals to control the data transfer between different blocks just in time, similar to a controller. Table (B.1) shows the details of sequencer signals[1]. In this table "$\rightarrow$" means data transfer and signal names refer to input and

[1]In the tables of this chapter delay of basic arithmetic operations are are assumed as follows:

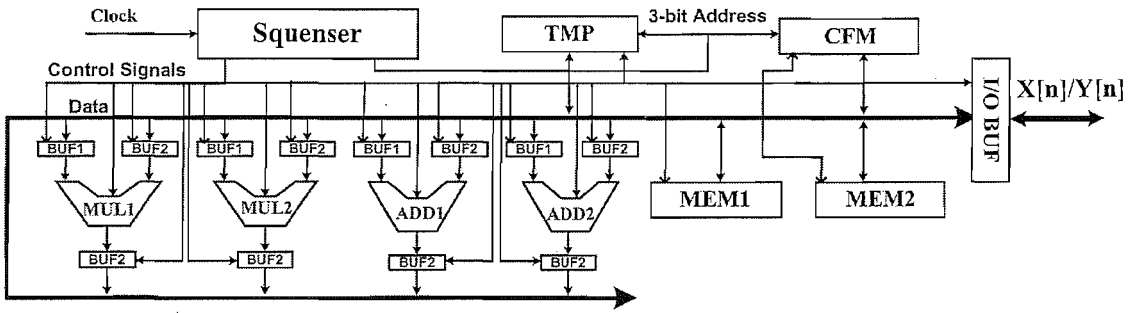
- r : Delay of adders,
- r^2 : Delay of multipliers,
- d : Basic delay of data transfer between registers.

It is also supposed that $r \geq d$.

output enabling signals. Clearly, having different operation delays, Table (B.1) not exactly show the timing of operation execution. These delays depend on the circuit design of each block.



(a)



STEP 1: (X , *b0*) → MUL1, (X , *b1*) → MUL2;
STEP 2: (MUL1 , MEM1) → ADD1, (MUL2 , MEM2) → ADD2, (X , *b2*) → MUL1 → TMP;
STEP 3: (ADD1 , *-a1*) → MUL1, (ADD1 , *-a2*) → MUL2;
STEP 4: (MUL1 , ADD2) → ADD1, (TMP , MUL2) → ADD2;
STEP 5: (ADD2) → MEM2, (ADD1) → MEM1.

(b)

FIGURE B.15: Block diagram of two pole system (Transposed type1) a) operations' sequence b) Block diagram.

B.3.2 Transforms

In this part we will try to implement a simple system of DFT transform by our soft-architecture to evaluate its ability. According to the general architecture, chapter 6, algorithm executers (AE) are responsible for execution of the different algorithms; the next subsections explain the corresponding AE design and structure.

AEs are the heart of the system in that they execute the algorithms of transforms, filters and any other functionality of the system. As a simple example we will implement an

N-point DFT by different methods. We suppose to $N = 2^k$ and we will implement an N-point DFT for the input signal $x[n]$. As mentioned, there are three basic methods: Goertzel Filters, FFT and matrix multiplication method. We will try to fit each method with the proposed architecture. Figure (6.3) shows the general structure of the Algorithm Executer (AE) and its Macro Cells (MC). Number of MCs in the AE is changeable and depends on costs or constraints.

TABLE B.1: Details of sequences for Two Pole Filter MC.

Steps	State No.	Register Transfer Signals	Clock Cycles	Total Delay
Step 1	1	X-OUT,MUL1-IN1,MUL2-IN1,	d	
	2	b0-ADS→CFM-AD,CFM-OUT,MUL1-N2,	d	
	3	b1-ADS→CFM-AD,CFM-OUT,MUL2-IN2,	r^2	$r^2 + 2d$
Step 2	4	MUL1-OUT, ADD1-IN1,	d	
	5	MEM1-OUT,ADD1-IN2,	d	
	6	MUL2-OUT,ADD2-IN1,	d	
	7	MEM2-OUT,ADD2-IN2,	d	
	8	b2-ADS→CFM-AD,CFM-OUT,MUL1-IN2,	d	
	9	ADD1-OUT,Y-IN,I/O,	r	
	10	MUL1-OUT,TMP-IN,	r^2	
	11	ADD1-OUT,MUL1-IN1,MUL2-IN1*,	d	$r^2 + r + 6d$
Step 3	12	-a1-ADS→CFM-AD,CFM-OUT,MUL1-IN2,	d	
	13	-a2-ADS→CFM-AD,CFM-OUT,MUL2-IN2,	d	
	14	ADD2-OUT,ADD1-IN1,	r	$r + 2d$
Step 4	15	1→MUL1-OUT,1→ADD1-IN2,	r^2	
	16	TMP-OUT,ADD2-IN1,	d	
	17	MUL2-OUT,ADD2-IN2,	r^2	
	18	ADD2-OUT,MEM2-IN,	r	$2r^2 + r + d$
Step 5	19	ADD1-OUT,MEM1-IN,	r	r
Total Delay			$4r^2 + 4r + 11d$	

Goertzel Filters:

As it has been explained in section B.2.1, this method can be implemented as a single pole or two pole IIR filter, see Figures (B.3) and (B.4). For a first order filter (Figure (B.3)), according to Equations (B.12) and (B.13), the computations involve multiplication by complex numbers and each complex multiplication results in four real multiplications and four real additions. On the other hand, the second order filter implementation has avoided complex multiplications by a mathematical simplification. Figure (B.16) shows a second order Goertzel filter implementation in more detail, by inspection, one can write Equation (B.26).

$$Q_m = x[m] + 2\cos\left(\frac{2\pi n}{N}\right) \cdot Q_{m-1} - Q_{m-2}, \tag{B.26}$$

and since it is known that $X[n] = y_n[N]$ and according to Figure (B.16) we have:

$$y_n[m] = Q_m - Q_{m-1} \cdot e^{-j\frac{2\pi}{N}n}, \tag{B.27}$$

Then after N iterations, $X[n]$ can be computed by the Equation (B.28).

$$X[n] = y_n[N] = Q_N - Q_{N-1} \cdot e^{-j\frac{2\pi}{N}n}, \tag{B.28}$$

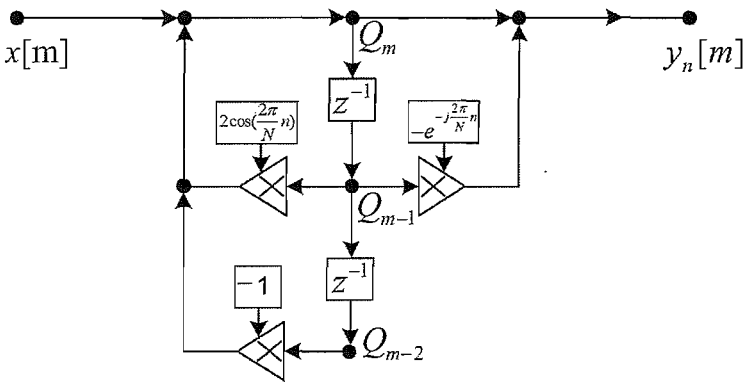


FIGURE B.16: Modified Goertzel algorithm block diagram.

Equation (B.28) means that complex multiplication is needed only once, for every $X[n]$, after the final step when Q_N has been computed. The structure of Figure (B.16) is very suitable for implementation by MCs. MC scheduling has been depicted in Figure (B.17) and Table (B.2) shows the corresponding data transfer signals.

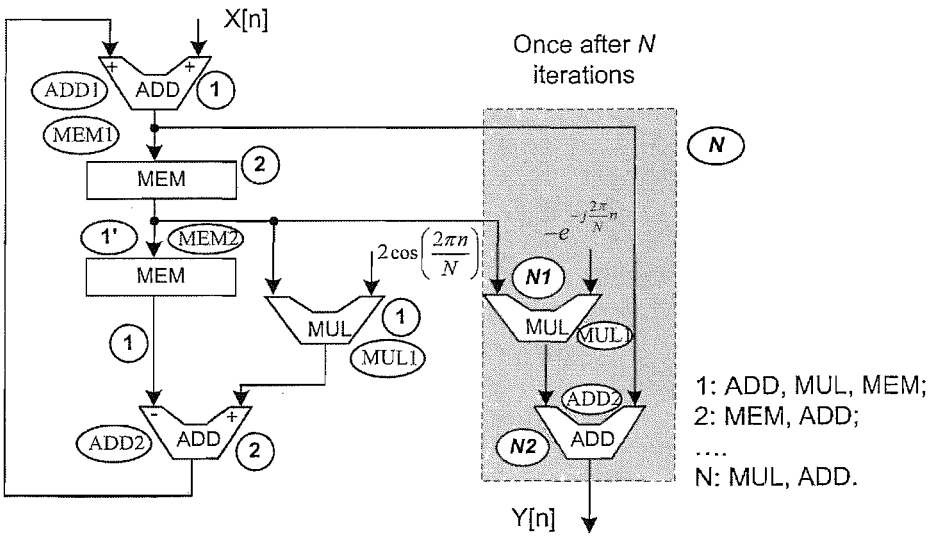


FIGURE B.17: Second order Goertzel Filter scheduling for MCs.

In Figure (B.17) it has been assumed that the needed twiddle factors [105] have been stored in the CFM (Coefficients Memory) of MCs. In addition, operations of MCs are controlled by another controller which is based on the structure of Figure (B.18). In

TABLE B.2: MC scheduling signals for Goertzel Filter.

Steps	State No.	Register Transfer Signals	Clock Cycles	Total Delay
Step 1	1	X-OUT,ADD1-IN1,BUF-IN,	d	
	2	ADD2-OUT,ADD1-IN2,	r	
	3	MEM2-OUT,ADD2-IN1,	d	
	4	MEM1-OUT,MEM2-IN,MUL1-IN1,	d	
	5	C1-ADS'CFM-AD,CFM-OUT, MUL1-IN2,	r^2	r^2+r+3d
Step 2	6	MUL1-OUT,ADD2-IN2,	r	
	7	ADD1-OUT,MEM1-IN,	d	$r+d$
After N iterations:				
Step N	N	MEM1-OUT,MUL1-IN,	d	
	N+1	C2-ADS'CFM-AD,CFM-OUT,MUL1-IN2,	r^2	
	N+2	ADD1-OUT,ADD2-IN1,	d	
	N+3	MUL1-OUT,ADD2-IN2,	r	
	N+4	ADD2-OUT,Y-IN,BUF-OUT,	d	r^2+2d
Total Delay : $(r^2+2r+4d)\times N+r^2+2d$				

Figure (B.18) inputs are applied to MCs one by one in serial form and each MC produces the DFT at one frequency.

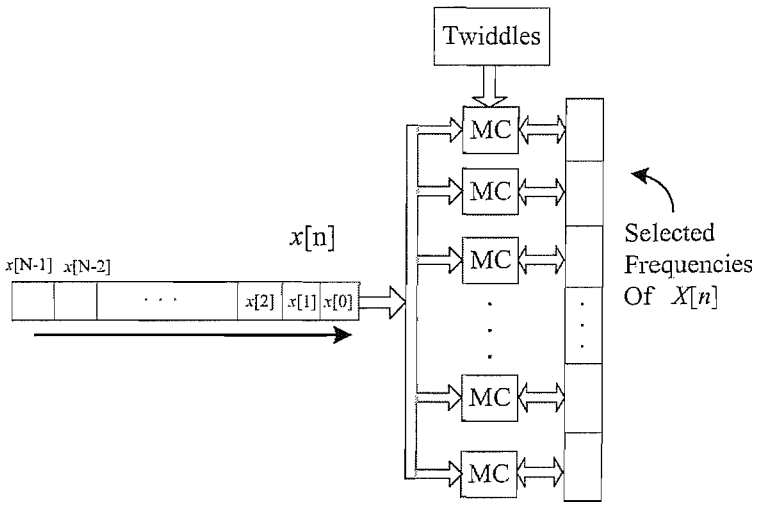


FIGURE B.18: A basic structure for the Goertzel Filters implementation.

FFT:

Two butterfly cells are possible for FFT: radix-2 and radix-4. Figure (B.19) shows the basic block diagram of a radix-2 butterfly cell. This block diagram can be implemented by MC, the scheduling diagram is depicted in Figure (B.20) and register transfer signals are shown in Table (B.3).

In this application it supposed that the twiddle factors have been saved in CFM which is not practical. In addition to the twiddle factors, the FFT implementation has two difficulties, the first is the number of MCs (Butterfly Cells) and the second is the data

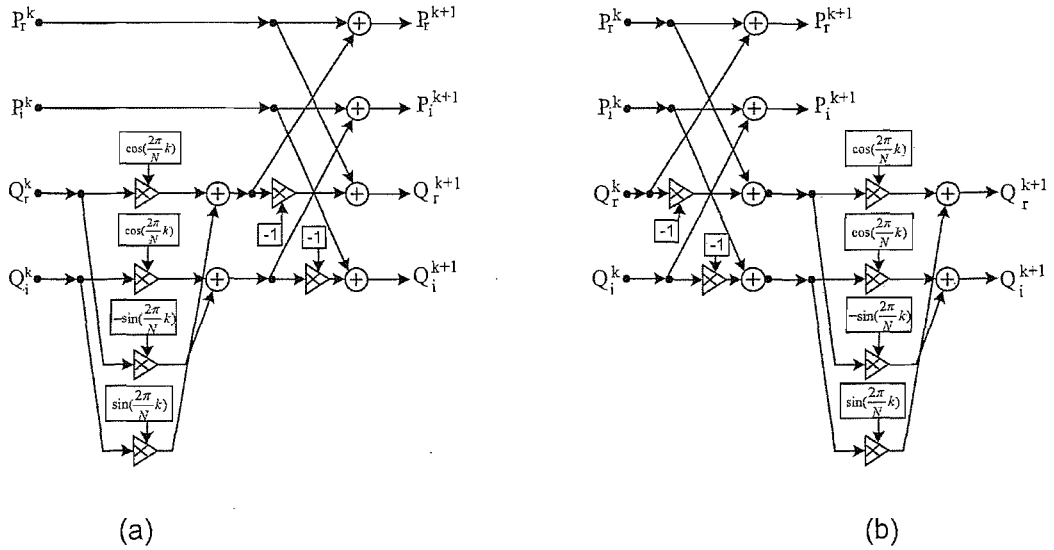


FIGURE B.19: Radix-2 FFT Butterfly Cell block diagram a) DIT form b) DIF form.

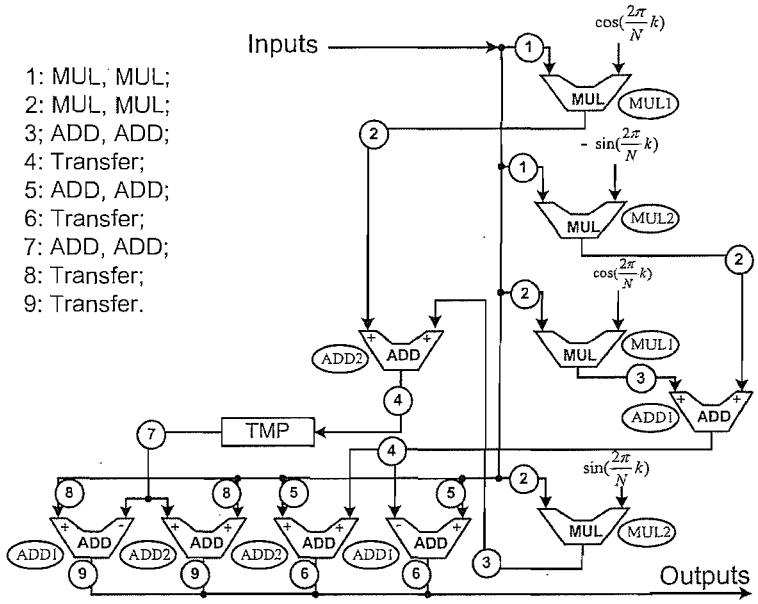


FIGURE B.20: Scheduling diagram for radix-2 butterfly DIT form.

passing scheme between them. To reduce the number of required MCs, a simple way is an in-place computation. It means that by the end of each state of computation the results will take the place of the previous input and will be the next stage input, see Figure (B.21-a).

In Figures (B.21), r is the number of MCs which can be equal or less than N (the number of input and output samples) but if $r < N$, the addressing of the memories and sequencing of operations will be more complicated. By using two set of memories (M_1 and M_2 in Figure (B.21)) addressing complexity and its overhead can be reduced dramatically.

TABLE B.3: MC scheduling for DIT radix-2 FFT Butterfly Cell.

Steps	State No.	Register Transfer Signals	Clock Cycles	Total Delay
Step 1	1	X-OUT,MUL1-IN1,MUL2-IN1,BUF-IN,	d	
	2	C1-ADS'CFM-AD,CFM-OUT,MUL1-IN2,	r^2	
	3	C2-ADS'CFM-AD,CFM-OUT,MUL2-IN2,	r^2	r^2+2d
Step 2	4	X-OUT,MUL1-IN1,MUL2-IN1,BUF-IN,	d	
	5	MUL1-OUT,ADD2-IN1,	d	
	6	C1-ADS'CFM-AD,CFM-OUT,MUL1-IN2,	r^2	
	7	MUL2-OUT,ADD1-IN1,	d	
	8	C3-ADS'CFM-AD,CFM-OUT,MUL2-IN2,	r^2	r^2+4d
Step 3	9	MUL1-OUT,ADD2-IN2,	r	
	10	MUL2-OUT,ADD1-IN2,	r	$r+d$
Step 4	11	ADD1-OUT,ADD1-IN1,ADD2-IN1,	d	
	12	ADD2-OUT,TMP-IN,	d	$2d$
Step 5	13	X-OUT,ADD1-IN2,ADD2-IN2,ADD1-SUB,BUF-IN,	r	r
Step 6	14	ADD1-OUT,MEM1-IN,	d	
	15	ADD2-OUT,MEM2-IN,	d	$2d$
Step 7	16	TMP-OUT,ADD1-IN1,ADD2-IN2,	d	d
Step 8	17	X-OUT,ADD1-IN2,ADD2-IN2,ADD1-SUB,BUF-IN,	r	r
Step 9	18	ADD1-OUT,Y-IN,BUF-OUT,	d	
	19	ADD2-OUT,Y-IN,BUF-OUT,	d	
	20	MEM1-OUT,BUF-OUT,	d	
	21	MEM2-OUT,BUF-OUT,	d	$4d$
Total Delay: $2r^2+3r+15d$				

Matrix Based Method

By using a matrix based method MCs must execute complex multiplication and accumulation, like Figure (B.22). Figure (B.23) presents the scheduling diagram and Table (B.4) shows the scheduling required for these operations.

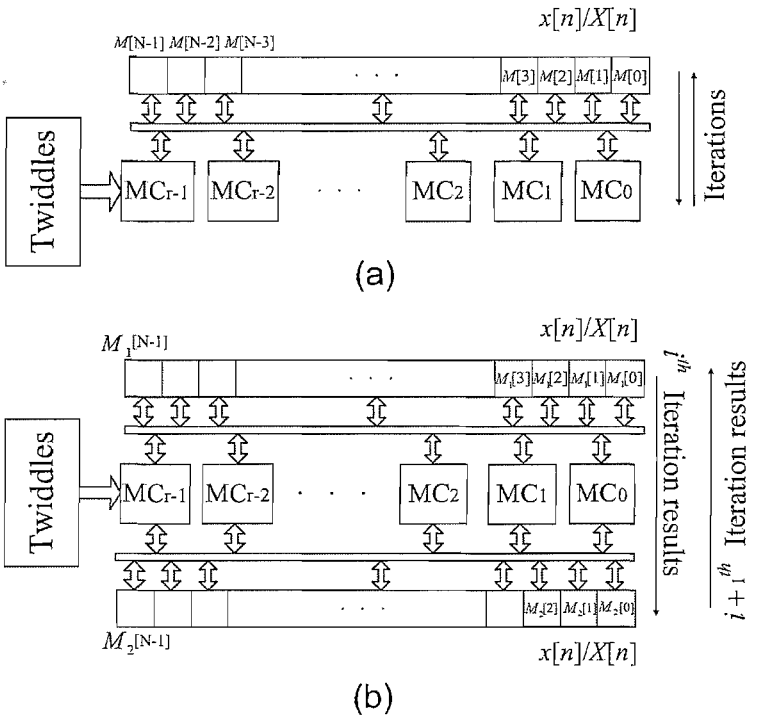


FIGURE B.21: In place computation of FFT by a limited number of MCs. a)with one set of memory b)with two sets of memories.

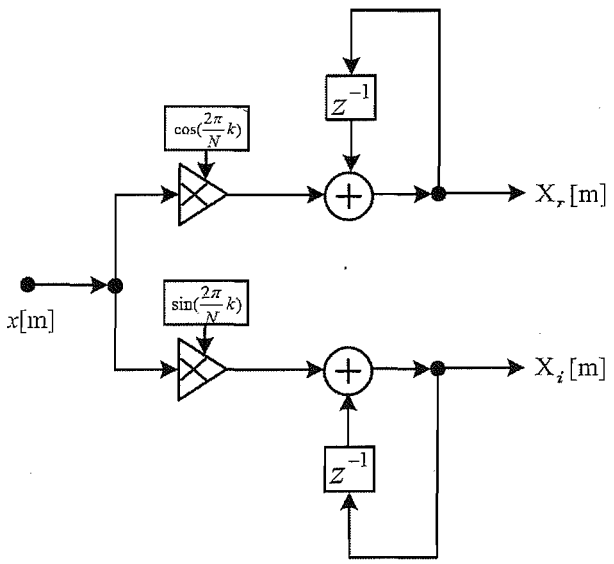


FIGURE B.22: Basic block diagram for Matrix Based signal manipulation by MCs.

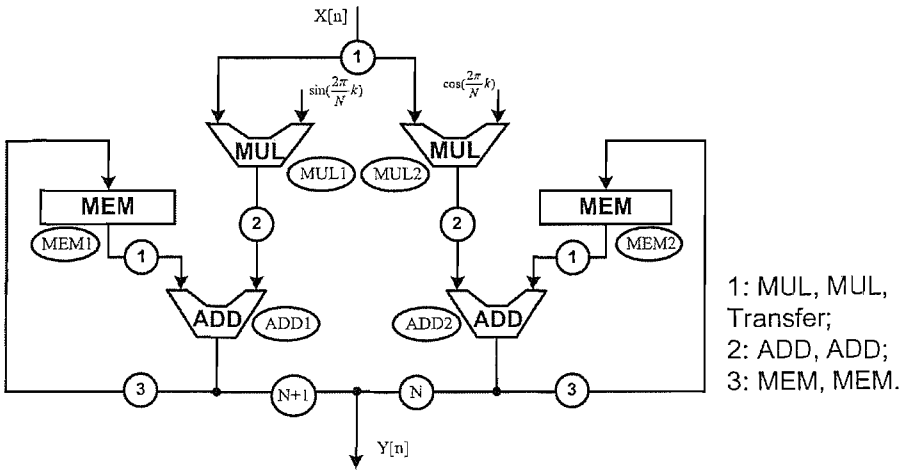


FIGURE B.23: Scheduling diagram for matrix based algorithm implementation by MCs.

TABLE B.4: Register Transfer Signals for MCs in a Matrix Based Transform Implementation.

Steps	State No.	Register Transfer Signals	Clock Cycles	Total Delay
Step 1	1	X-OUT, MUL1-IN1, MUL2-IN1, BUF-IN,	d	
	2	C1-ADS'CFM-AD, CFM-OUT, MUL1-IN2,	r^2	
	3	C2-ADS'CFM-AD, CFM-OUT, MUL2-IN2,	r^2	
	4	MEM1-OUT, ADD1-IN1,	d	
	5	MEM2-OUT, ADD2-IN1,	d	r^2+d
Step 2	6	MUL1-OUT, ADD1-IN2,	r	
	7	MUL2-OUT, ADD2-IN2,	r	$r+d$
Step 3	8	ADD1-OUT, MEM1-IN,	d	
	9	ADD2-OUT, MEM2-IN,	d	$2d$
After N inputs and N iteration				
Step N		ADD1-OUT, Y-IN, BUF-OUT,	d	
		ADD2-OUT, Y-IN, BUF-OUT,	d	$2d$
Total Execution Delay (after N iterations) $= (r^2+r+4d) \times N + 2d$				

References

[1] H. Alt. Comparison of arithmetic functions with respect to boolean circuit depth. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 466–470, New York, NY, USA, 1984. ACM Press. ISBN 0-89791-133-4.

[2] American National Standard Institute and The Institute of Electrical and Electronics Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard*, (Std 754-1985), 1985.

[3] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, and R. Uribe. Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design. In *FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 263–264, Washington, DC, USA, April 2003. IEEE Computer Society. ISBN 0-7695-1979-2.

[4] P. Banerjee, M. Haldar, A. Nayak, V. Kim, V. Saxena, S. Parkes, D. Bagchi, S. Pal, N. Tripathi, D. Zaretsky, R. Anderson, and J. R. Uribe. Overview of a compiler for mapping MATLAB programs onto FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):312–324, March 2004.

[5] E. H. Bareiss and J. L. Barlow. Roundoff error distribution in fixed point multiplication. *BIT Numerical Mathematics*, 20(2):247–250, July 1980. ISSN 1572-9125.

[6] C. W. Barnes, B. N. Tran, and S. H. Leung. On the statistics of fixed-point roundoff error. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(3):595–606, June 1985.

[7] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986. ISSN 0097-5397.

[8] D. Berleant. Automatically verified reasoning with both intervals and probability density functions. *Interval Computations*, (2):48–70, 1993.

[9] M. Berz. From Taylor series to Taylor models. In *In AIP Conference Proceedings 405*, page 123, 1997.

[10] F. Black and M. S. Scholes. The valuation of option contracts and a test of market efficiency. *Journal of Finance*, 27(2):399–417, May 1972.

[11] F. Black and M. S. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–54, May-June 1973.

[12] R. Boite, X.-L. He, and J.P. Renard. A comparison of fixed-point and floating-point realization of digital filter. In *EUROCON 88: 8th European Conference on Electrotechnics. Conference Proceedings on Area Communication*, pages 142–145, June 1988.

[13] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, January 1958.

[14] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23:242–251, 1976. ISSN 0004–5411.

[15] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. pages 151–176, 1976.

[16] G. Caffarena, G. A. Constantinides, P. Y. K. Cheung, C. Carreras, and O. Nieto-Taladriz. Otpimal combined word-length allocation and architectural synthesis of digital signal processing circuits. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(5):339–343, May 2006.

[17] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie. An automatic word length determination method. In *ISCAS'01:The 2001 IEEE International Symposium on Circuits and Systems*, volume 5, pages 53–56, 2001. ISBN 0-7803-6685-9.

[18] C. Carreras and M. V. Hermenegildo. Grid-based histogram arithmetic for the probabilistic analysis of functions. In *SARA '02: Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation*, pages 107–123, London, UK, 2000. Springer-Verlag. ISBN 3-540-67839-5.

[19] J. Y. Chen, W. B. Jone, J. S. Wang, H.-I. Lu, and T. F. Chen. Segmented bus design for low-power systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1): 25–29, 1999. ISSN 1063-8210.

[20] H. W. Chun. A distributed constraint-based search architecture for bus timetabling and duty assignment. In *APSEC '97: Proceedings of the Fourth Asia-Pacific Software Engineering and International Computer Science Conference*, page 82, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8271-X.

[21] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement. In *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, page 56, New York, NY, USA, 1999. ACM Press. ISBN 1-58113-121-6.

[22] C. A. C. Coello. An update survay of GA-based multiobjective optimization. *ACM Computing Survays*, 23(2):109–143, June 2000.

[23] C. A. C. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–87, June 2002.

[24] G. A. Constantinides. Perturbation analysis for word-length optimization. In *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM03)*, pages 81–90, 2003.

[25] G. A. Constantinides. Word-length optimization for differentiable nonlinear systems. *ACM Trans. Des. Autom. Electron. Syst.*, 11(1):26–43, 2006. ISSN 1084-4309.

[26] G. A. Constantinides, P. Y. K. Cheung, and Wayne L. Optimum and heuristic synthesis of multiple word-length architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(1):39–57, June 2005.

[27] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Synthesis of saturation arithmetic architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 8(3):334–354, 2003. ISSN 1084-4309.

[28] G. A Constantinides, P. Y. K. Cheung, and W. Luk. *Synthesis and Optimization of DSP Algorithms (Fundamental Theories of Physics S.)*. Kluwer Academic Publishers, 2004. ISBN 1402079303.

[29] G. A. Constantinides and G. J. Woeginger. The complexity of multiple wordlength assignment. *Applied Mathematics Letters*, 15(2):137–140, 2002.

[30] J. Daalder, P. W. Eklund, and K. Ohmori. High-level synthesis optimization with genetic algorithms. In *PRICAI '96: Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence*, pages 276–287, London, UK, 1996. Springer-Verlag. ISBN 3-540-61532-6.

[31] N. Dahnoum. *Digital Signal Processing Implementation.* Pearson Education, 2000. ISBN 0201619164.

[32] L. Henrique de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, December 2004. ISSN 1017-1398.

[33] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill Education, 1994. ISBN 0071132716.

[34] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley and Sons, 2001. ISBN 047187339X.

[35] J.-P. Deschamps, G. J. A. Bioul, and G. D. Sutter. *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems.* Wiley-Interscience, 2006. ISBN 0471687839.

[36] J. Detrey and F. de Dinechin. *Second Order Function Approximation Using a Single Multiplication on FPGAs*, volume 3203/2004 of *Book Series Lecture Notes in Computer Science*, pages 221–230. Springer Berlin-Heidelberg, 2004. ISBN 978-3-540-22989-6.

[37] P. S. R. Diniz, E. da Silva, S. L. Netto, and E. A. B. da Silva. *Digital Signal Processing: System Analysis and Design.* Cambridge University Press, 2002. ISBN 0521781752.

[38] N. Doi, T. Horiyama, M. Nakanishi, and S. Kimura. Bit-length optimization method for high-level synthesis based on non-linear programming technique. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(12):3427–3434, 2006. ISSN 0916-8508.

[39] P. Eles, K. Kuchcinski, and Z. Peng. *System Synthesis with VHDL.* KluwerAcademic, 1998.

[40] C. Ewering. Automatic high level synthesis of partitioned busses. In *ICCAD-90:IEEE International Conference on Computer-Aided Design*, pages 304–307, November 1990.

[41] C. F. Fang. *Probabilistic interval-valued computation: representing and reasoning about uncertainty in DSP and VLSI design.* PhD thesis, Pittsburgh, PA, USA, 2005.

[42] C. F. Fang, R. A. Rutenbar, and T. Chen. Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, pages 275–282, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 1-58113-762-1.

[43] E. Frank and T. Lengauer. APPlaUSE: Area and performance optimization in a unified placement and synthesis environment. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 662–667, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7213-7.

[44] E. Frank, S. Raje, and M. Sarrafzadeh. Constrained register allocation in bus architectures. In *DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 170–175, New York, NY, USA, 1995. ACM Press. ISBN 0-89791-725-1.

[45] D. D. Gajski, N. D. Dutt, A. C-H Wu, and S. Y-L Lin. *High-level Synthesis: Introduction to Chip and System Design*, page 376. Kluwer Academic Publishers, first edition, April 1992. ISBN 978-0792391944.

[46] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H.Freeman & Co Ltd, April 1979. ISBN 978-0716710455.

[47] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis.* Addison Wesley, 7 edition, July 2003. ISBN 978-0321133045.

[48] A. G. Glen, L. M. Leemis, and J. H. Drew. Computing the distribution of the product of two continuous random variables. *Computational Statistics and Data Analysis*, 4(3):451–464, January 2004.

[49] D. A. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1989. ISBN 0201157675.

[50] R. M. Gray. Quantization noise spectra. *IEEE Transactions on Information Theory*, 36(6):1220–1244, November 1990.

[51] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, October 1998.

[52] K. Han and B. L. Evans. Wordlength optimization using sensitivity information. *EURASIP Journal on Applied Signal Processing, special issue on Design Methods for DSP Systems*, (5):1–14, 2006.

[53] D. J. Higham. Black-scholes for scientific computing students. *Computing in Science and Engg.*, 6(6):72–79, 2004. ISSN 1521-9615.

[54] J. L. Hodges, E. L. Lehmann, and J. L. Hodges. *Basic Concepts Of Probability And Statistics.* Society for Industrial and Applied Mathematic, second edition, January 2005. ISBN 978-0898715750.

[55] C.-Ta Hsieh and M. Pedram. Architectural power optimization by bus splitting. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 612–616, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-244-1.

[56] C.-Ta Hsieh and M. Pedram. Architectural energy optimization by bus splitting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(4):408–414, April 2002.

[57] J. Jeon and K. Choi. Effective synthesis algorithm for partitioned bus architecture. *Electronics Letters*, 35(6):440–441, March 1999. ISSN 0013-5194.

[58] W.-B. Jone, J. S. Wang, Hsueh-I Lu, I. P. Hsu, and J.-Y. Chen. Design theory and implementation for low-power segmented bus systems. *ACM Trans. Des. Autom. Electron. Syst.*, 8(1):38–54, 2003. ISSN 1084-4309.

[59] B. KAMGAR-PARSI, B. KAMGAR-PARSI, and M. BROSH. Distribution and moments of the weighted sum of uniform random variables, with applications in reducing monte carlo simulations. *Journal of statistical computation and simulation*, 52(4):399–414, 1995. ISSN 0094-9655.

[60] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: a fixed-point design and simulation environment. In *DATE '98: Proceedings of the conference on Design, automation and test in Europe*, pages 429–435, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8359-7.

[61] K. Kim, K. Choi, and Y.-H. Jun. Hardware synthesis for stack type partitioned-bus architecture. In *ICVC'99:6th International Conference on VLSI and CAD*, pages 81–84, October 1999.

[62] S. Kim and W. Sung. A floating-point to fixed-point assembly program translator for the TMS 320c25. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41(11):730 – 739, November 1994.

[63] S. Kim and W. Sung. Fixed-point error analysis and word length optimization of 88 IDCT architectures. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(8):935 – 940, December 1998.

[64] P. Kollig and B.M. Al-Hashimi. Simultaneous scheduling, allocation and binding in high level synthesis. *Electronics Letters*, 33(18):1516–1518, August 1997.

[65] I. Koren. *Computer Arithmetic Algorithms*. A. K. Peters Ltd., Natick, MA, USA, 2001. ISBN 1568811608.

[66] K.-I. Kum, J. Kang, and W. Sung. A floating-point to integer C converter with shift reduction for fixed-point digital signal processors. In *ICASSP '99: IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2163 – 2166, March 1999.

[67] K.-I. Kum, J. Kang, and W. Sung. AUTOSCALER for C: an optimizing floating-point to integer c program converter for fixed-point digital signal processors. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47 (9):840–848, September 2000.

[68] Ki-Il Kum and W. Sung. Combined word-length optimization and high-level synthesis of digital signal processing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(8):921–930, Unknown 2001.

[69] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26 (2):203–215, February 2007. ISSN 0278-0070.

[70] Y.-K. Kwok. *Mathematical Models of Financial Derivatives.* Springer, 1 edition, May 1999. ISBN 978-9813083257.

[71] B. Le-Gal, C. Andriamisaina, and E. Casseau. Bit-width aware high-level synthesis for digital signal processing systems. In *IEEE International SOC Conference*, pages 175–178, September 2006.

[72] D.-U. Lee, A. Abdul-Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. Accuracy-guaranteed bit-width optimization. *IEEE transactions on computer-aided design of Integrated Circuits and Systems*, 25(10):1990–2000, OCTOBER 2006.

[73] D.-U Lee, A. Abdul-Gaffar, O. Mencer, and W. Luk. Optimizing hardware function evaluation. *IEEE Trans. Comput.*, 54(12):1520–1531, 2005. ISSN 0018-9340.

[74] D.-U Lee, W. Luk, J. D. Villasenor, and P. Y. K. Cheung. A Gaussian noise generator for hardware-based simulations. *IEEE Trans. Comput.*, 53(12):1523–1534, 2004. ISSN 0018-9340.

[75] S. Lee and K. Choi. Partitioned-bus architecture synthesis based on data transfer model. In *APCHDL 99: The 6th Asia Pacific Conference on Chip Design Language*, pages 144–149, 1999.

[76] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong. Automatic floating to fixed point translation and its application to post-rendering 3D warping. In *FCCM '99: Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 240, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0375-6.

[77] J. D. Ma and R. A. Rutenbar. Fast interval-valued statistical interconnect modeling and reduction. In *ISPD '05: Proceedings of the 2005 international symposium on Physical design*, pages 159–166, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-021-3.

[78] E. Macii, M. Pedram, and F. Somenzi. High-level power modeling, estimation, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1061–1079, 1998.

[79] K. Makino and M. Berz. Remainder differential algebras and their applications. *Computational Differentiation: Techniques, Applications, and Tools SIAM*, 1996.

[80] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou. Smart bit-width allocation for low power optimization in a systemc based ASIC design environment. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 618–623, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. ISBN 3-9810801-0-6.

[81] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou. Low-power optimization by smart bit-width allocation in a systemc-based ASIC design environment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(3):447–455, March 2007.

[82] G. Marsaglia and W. W. Tsang. The Ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

[83] M. C. McFarland, A. C. Parker, and R. Camposano. Tutorial on high-level synthesis. In *DAC '88: Proceedings of the 25th ACM/IEEE conference on Design automation*, pages 330–336, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press. ISBN 0-8186-8864-5.

[84] D. Menard, D. Chillet, F. Charot, and O. Sentieys. Automatic floating-point to fixed-point conversion for DSP code generation. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 270–276, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-575-0.

[85] D. Menard and O. Sentieys. *DSP Code Generation with Optimized Data Word-Length Selection*, volume 3199/2004 of *Lecture Notes in Computer Science*, pages 214–228. Springer Berlin Heidelberg, October 2004. ISBN 978-3-540-23035-9.

[86] O. Mencer, N. Boullis, W. Luk, and H. Styles. Parameterized function evaluation for FPGAs. In *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 544–554, London, UK, 2001. Springer-Verlag. ISBN 3-540-42499-7.

[87] R. Michard, A. Tisserand, and N. Veyrat-Charvillon. Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x. In *IEEE International Conference on Application-Specific Systems, Architecture Processors*, pages 334– 339, July 2005. ISBN 0-7695-2407-9.

[88] A. Mignotte and M. Crastes de Paulet. Resource assignment with different target architectures. In *Euro ASIC 91*, pages 172–177, May 1991.

[89] R. E. Moore. *Interval analysis.* Prentice-Hall, 1966.

[90] R. E. Moore and F. Bierbaum. *Methods and Applications of Interval Analysis.* Soc for Industrial & Applied Math, 1979. ISBN 0898711614.

[91] V. G. Moshnyaga, F. Ohbayashi, and K. Tamaru. A scheduling algorithm for synthesis of bus-partitioned architectures. In *ASP-DAC '95: Proceedings of the 1995 conference on Asia Pacific design automation (CD-ROM)*, page 7, New York, NY, USA, 1995. ACM Press. ISBN 0-89791-766-9.

[92] J. M. Muller. *Elementary Functions: Algorithms and Implementation.* Birkhauser Boston, second edition, October 2005. ISBN 0817643729.

[93] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs. In *Design, Automation and Test in Europe, 2001*, pages 722 – 728, March 2001.

[94] N. S. Nedialkov, V. Kreinovich, and S. A. Starks. Interval arithmetic, affine arithmetic, taylor series methods: Why, what next? *Numerical Algorithms*, 37(1-4): 325–336, December 2004. ISSN 1017-1398.

[95] S. N. Neftci. *Introduction to the Mathematics of Financial Derivatives.* Academic Press, 2nd edition, April 2000. ISBN 978-0125153928.

[96] A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence region. *Computing Supplementum*, 9:175–190, 1993.

[97] E. P. O'Grady. Hardware support for floating point map function generation. In *SS '99: Proceedings of the Thirty-Second Annual Simulation Symposium*, page 145, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0128-1.

[98] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing.* Pearson US Imports and PHIPEs, 1999. ISBN 0130834432.

[99] A. V. Oppenheim and C. J. Weinstein. Effects of finite register length in digital filtering and the fast Fourier transform. In *IEEE Proceedings*, pages AU–17:209–215. IEEE, 1972.

[100] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press Inc, January 2000. ISBN 978-0195125832.

[101] K. Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation.* John Wiley and Sons Inc, 1999. ISBN 0471241865.

[102] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *DAC '87: Proceedings of the 24th ACM/IEEE conference on Design automation*, pages 195–202, New York, NY, USA, 1987. ACM Press. ISBN 0-8186-0781-5.

[103] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, June 1989.

[104] J. Proakis, D. Manolakis, and D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications.* Pearson US Imports & PHIPEs, 1995. ISBN 0133942899.

[105] J. G. Proakis. *Digital Signal Processing.* Prentice Hall, 2006. ISBN 0131873741.

[106] Y. Pu and Y. Ha. An automated, efficient and static bit-width optimization methodology towards maximum bit-width-to-error tradeoff with affine arithmetic model. In *ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 886–891, New York, NY, USA, 2006. ACM Press. ISBN 0-7803-9451-8.

[107] M. Puschel, J.M.F. Moura, J.R. Johnson, D. Padua, M.M. Veloso, B.W. Singer, Jianxin Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo. SPIRAL: code generation for DSP transforms. *Proceedings of the IEEE*, 93(2):232–275, 2005.

[108] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits.* Prentice Hall, second edition, December 2002. ISBN 978-0130909961.

[109] H. Regan, S. Ferson, and D. Berleant. Equivalence of methods for uncertainty propagation of real-valued random variables. *International Journal of Approximate Reasoning*, 36(1):130, April 2004.

[110] N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64(1):135–154, July 2005.

[111] S. Roy and P. Banerjee. An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design. *IEEE Trans. Comput.*, 54 (7):886–896, 2005. ISSN 0018-9340.

[112] A. Rushton. STLplus library. http://stlplus.sourceforge.net/, October 2004.

[113] T. Seceleanu. Communication on a segmented bus platform. In *IEEE International SOC Conference*, pages 205–208, September 2004.

[114] T. Seceleanu, V. Leppanen, J. Suomi, and O. Nevalainen. Resource allocation methodology for the segmented bus platform. In *IEEE International SOC Conference*, pages 129–132, September 2005.

[115] T. Seceleanu, Ville Leppnen, Jyri Suomi, and Olli S. Nevalainen. On the organization of multisegmented bus. Technical Report 647, Turku, Finland, December 2004.

[116] T. Seceleanu, St. Stancescu, and V. Lazarescu. Distributed arbitration for the segmented bus platform. In *ISSCS'2005: International Symposium on Signals, Circuits & Systems*, pages 63–66, 2005.

[117] T. A. Severini. *Elements of Distribution Theory.* Cambridge University Press, first edition, August 2005. ISBN 978-0521844727.

[118] C. Shi. *Floating-point to fixed-point conversion.* PhD thesis, Department of EECS, University of California, Berkeley, USA, May 2004.

[119] C. Shi and R. W. Brodersen. Automated fixed-point data-type optimization tool for signal processing and communication systems. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 478–483, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-8.

[120] C. Shi and R. W. Brodersen. A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary inputs. In *ISCAS '04: Proceedings of the 2004 International Symposium on Circuits and Systems*, volume 3, pages III–373–6. IEEE, May 2004.

[121] A. Singhee, C. F. Fang, J. D. Ma, and R. A. Rutenbar. Probabilistic interval-valued computation: toward a practical surrogate for statistics inside CAD tools. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 167–172, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-381-6.

[122] A. B. Sripad and D. L. Snyder. A necessary and sufficient condition for quantization errors to be uniform and white. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(5):442–448, October 1977.

[123] J. Stolfi and L. de Figueiredo. *Self-Validated Numerical Methods and Applications*. Institute for Pure and Applied Mathematics (IMPA), Rio de Janeiro, 1997. Monograph for the 21st Brazilian Mathematics Colloquium (CBM'97), IMPA.

[124] D. Stroobandt. *A Priori Wire Length Estimates for Digital Design*. Kluwer Academic Publishers, 2001. ISBN 079237360X.

[125] N. Sulaiman and T. Arslan. A multi-objective genetic algorithm for on-chip real-time optimisation of word length and power consumption in a pipelined FFT processor targeting a MC-CDMA receiver. In *2005 NASA/DoD Conference on Evolvable Hardware*, pages 154–159, 2005.

[126] W. Sung and K. Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing*, 43(12):3087–3090, December 1995.

[127] S. Tarafdar and M. Leeser. The DT-model: high-level synthesis using data transfers. In *DAC '98: Proceedings of the 35th annual conference on Design automation*, pages 114–121, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-964-5.

[128] S. Tarafdar and M. Leeser. A data-centric approach to high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19 (11):1251–1267, November 2000. ISSN 0278-0070.

[129] C. D. Thompson. Area-time complexity for VLSI. In *STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 81–88, New York, NY, USA, 1979. ACM Press.

[130] I. Tokaji and C. W. Barnes. Roundoff error statistics for a continuous range of multiplier coefficients. *IEEE Transactions on Circuits and Systems*, 34(1):52–59, January 1987.

[131] M. Urabe. Roundoff error distribution in fixed-point multiplication and a remark about the rounding rule. *SIAM Journal of Numercal Analalysis 5, 202 (1968)*, 5 (2):202–210, June 1968.

[132] S. V. Vaseghi. *Advanced Signal Processing and Digital Noise Reduction.* John Wiley and Sons, 2005. ISBN 047009494X.

[133] I. G. Vladimirov and P. Diamond. A uniform white-noise model for fixed-point roundoff errors in digital systems. *Automation and Remote Control*, 63(5):753–765, May 2002. ISSN 0005-1179.

[134] J. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8:330–334, September 1959.

[135] H. Wang, A. Papanikolaou, M. Miranda, and F. Catthoor. A global bus power optimization methodology for physical design of memory dominated systems by coupling bus segmentation and activity driven block placement. In *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation*, pages 759–761, Piscataway, NJ, USA, 2004. IEEE Press. ISBN 0-7803-8175-0.

[136] C. Weinstein and A. V. Oppenheim. A comparison of roundoff noise in floating point and fixed point digital filter realizations. *Proceedings of the IEEE*, 57(6): 1181– 1183, June 1969. ISSN 0018-9219.

[137] H. S. Wilf. *Algorithms and Complexity.* A K Peters, second edition, November 2002. ISBN 978-1568811789.

[138] A. C. Williams, A. D . Brown, and M. Zwolinski. Simultaneous optimisation of dynamic power, area and delay in behavioural synthesis. *Computers and Digital Techniques, IEE Proceedings*, 147(6):383–390, November 2000.

[139] W.-F. Wong, R. S. Nikhil, D. L. Rosenband, and N. Dave. High-level synthesis: an essential ingredient for designing complex ASICs. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 775–782, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8702-3.

[140] G. Zhang, P.H.W. Leong, D.-U Lee, J.D. Villasenor, R.C.C. Cheung, and W. Luk. Ziggurat-based hardware Gaussian random number generator. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 275–280, August 2005.

[141] M. Zwolinski. *Digital System Design with VHDL.* Prentice Hall, 2003. ISBN 013039985X.