UNIVERSITY OF SOUTHAMPTON

INVESTIGATION INTO POWER MINIMISATION ALGORITHMS FOR BEHAVIOURAL SYNTHESIS

by

Marco A. Ochoa-Montiel

A thesis submitted for the degree of

Doctor of Philosophy

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science,

University of Southampton,

United Kingdom

February 2008

This thesis is dedicated to my *bebezinha*.

"The burden of suffering seems a tombstone hung about our necks,
while in reality it is only the weight which is necessary to
keep down the diver while he is hunting pearls"

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

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

Doctor of Philosophy

Investigation into power minimisation algorithms for behavioural synthesis

Marco A. Ochoa-Montiel

The rapid growth of mobile electronics has led power consumption to be considered as a critical design priority. This necessitates the development of algorithms and design tools that target power minimisation at all levels of the design abstraction. The work presented in this thesis addresses the problem of dynamic power minimisation at behavioural level. A detailed investigation into power reduction algorithms during behavioural synthesis is presented. The research undertaken has produced two novel power-aware algorithms: time constrained scheduling and datapath synthesis. The power-aware time constrained scheduling algorithm selects the clock period and operations throughput such that power consumption can be reduced by scaling the voltage until the slack of at least one of the design operations is zero. It has been shown that by carefully choosing the clock period and operations throughput, it is possible to produce a set of solutions with different power-area tradeoffs. To demonstrate the efficiency of the new scheduling algorithm in terms of solution quality, scheduling results of various benchmark examples have been included and compared with a multiple supply voltage (MSV) algorithm. It has been shown that the proposed algorithm is capable of obtaining schedules with single supply voltage (SSV) that have identical resource requirements and comparable power consumption to schedules obtained using a MSV algorithm. Using SSV avoids the difficulties of MSV, including area and power overhead due to required level shifters to transfer data between functional units operating at different voltages.
To solve the highly interrelated tasks of behavioural synthesis together with the power minimisation problem, an efficient algorithm for concurrent scheduling, binding, and clock and operations throughput selection has been introduced. This represents the second contribution of this work. Using a simulated annealing-based optimisation and a compound cost function, the exploration of different power-area tradeoffs is possible. The new scheduling and datapath synthesis algorithms have been incorporated into a power aware behavioural compiler (PABCOM). Synthesis results of various benchmark examples are included to demonstrate the higher solution quality when compared with a power-aware algorithm previously reported. Furthermore, to demonstrate the applicability of PABCOM in dealing with a real life design, two solutions for the motion vector reconstructor from MPEG-1 decoder have been implemented using 0.12μm technology. Power and area values for both solutions have been obtained using the reports generated after logic synthesis with Synplify ASIC and power analysis with PrimePower. The solutions dissipate 31% and 42% less power than if they were operated at the maximum supply voltage of the library components.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Bashir M. Al-Hashimi for his continuos trust, support and guidance throughout the course of this research. Many thanks are due to Dr. Peter Kollig who has been involved in the technical supervision of this project. I am also grateful for the technical advice and friendship from my research colleagues Dr. Arash, Dr. Bis, Dr. Karthik and Dr. Ivo.

Thanks to CONACyT (Consejo Nacional de Ciencia y Tecnologia, Mexico), who provided the scholarship to pursue my PhD studies. I also want to acknowledge the University of Southampton, who provided an extension to carry on with my PhD studies after the fire of the Mountbatten bulding.

Likewise I want to express many thanks to all the people who have contributed somehow in my life and have helped me to get where I am today. Particularly, thanks to my parents and my sister, for all the love, support and encouragement they have given me over the years. I am also extremely grateful with my wife Ana ("bebezinha") for her continuing love, care, advice, encouragement and patience. Te amo con todo mi corazon, con toda mi alma y con todo lo que tengo condenada!!! Fica comigo sempre ☺. You know that any problem looks so small when I see you smiling ... Finally, infinite thanks to G. O. D. for always being next to me.

Table of contents

List of Figures

List of Tables

List of abbreviations

ALAP	As Late As Possible
AR	Autoregressive filter
ASAP	As Soon As Possible
CAD	Computer Aided Design
CDFG	Control Data Flow Graph
CFG	Control Flow Graph
CG	Compatibility Graph
CLB	Configurable Logic Block
cp	Critical path
cs	cstep
cstep	control step
DCT	Discrete Cosine Transform
DCU	Dynamic Clocking Unit
DFC	Dynamic Frequency Clocking
DFG	Data Flow Graph
DIFFEQ	Differential Equation
EWF	Elliptical Wave Filter
FDLS	Force Directed List Scheduling
FDS	Force Directed Scheduling
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FU	Functional Unit
HLS	High Level Synthesis
IDCT	Inverse Discrete Cosine Transform
ILP	Integer Linear Programming

LP	Linear Programming
MSV	Multiple Supply Voltages
PABCOM	Power-Aware Behavioural Compiler
PATICS	Power-Aware Time Constrained Scheduling
RCS	Resource Constrained Scheduling
RTL	Register Transfer Level
SoC	System on Chip
SSV	Single Supply Voltage
TCS	Time Constrained Scheduling
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1 Motivation

Continuous improvement in silicon technology is resulting in chips that are faster, smaller and have more transistors than their predecessors. This has resulted in increased processing capacity per chip, as well as a steady growth of operating frequency, leading to higher power consumption [87]. Power consumption has turned into a high priority consideration in hand-held mobile electronic system design due to cost, reliability and portability concerns, making low power design a critical issue that is being continuously investigated by academia and industry. High power dissipation can lead to expensive designs since costly packaging and cooling techniques are required to avoid high operating temperatures, which lead to less reliable systems. Also, high power dissipation implies a short battery life.

To successfully design low power electronics systems, it is necessary to tackle the power consumption problem at each level of abstraction of the design hierarchy [24]. The main levels of design abstraction [71] are:

- System level. The system level is concerned with the overall system structure and information flow. At this level, the design may be modelled as a set of abstract communicating processes or tasks.
- Behavioural level. This level is also called Algorithmic Level. At this level the focus is on the operations performed by the system.
- Register transfer level. The system is viewed as a set of interconnect storage elements and functional blocks.

Power reduction at the different levels of abstraction can be implemented using different techniques [135], as shown in Figure 1.1. It can also be seen that the exploitation of power optimisation at various levels of the design hierarchy offers different power reductions. It is clear that power optimisation opportunities are significantly larger at the higher levels, thus reducing power consumption only at lower levels may not be enough to meet the designer power reduction requirements. For this reason, power optimisation techniques are being incorporated into the *synthesis* process at higher levels of abstraction as shown in [150], [45], [67], [79], [55] and [69].

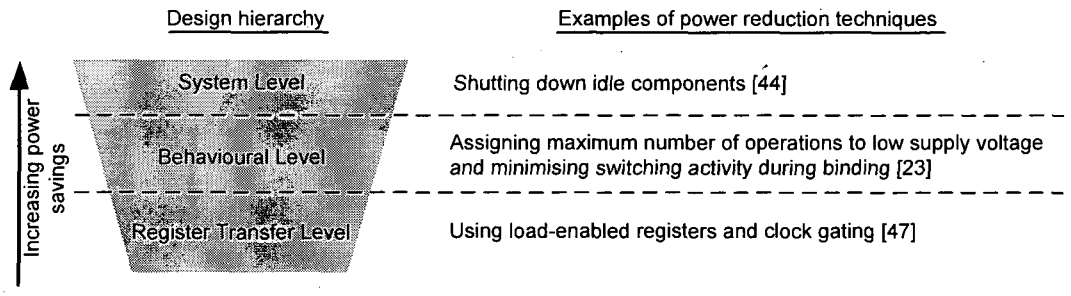


Figure 1.1 Power reduction at different levels of abstraction

Synthesis consist of finding a *structure* that implements the *behaviour* of the design while satisfying certain goals and constraints specified by the designer [21]. The *behaviour* of the design is seen as a black box where the relations between inputs and outputs are given regardless of their implementation. The *structure* refers to the set of interconnected components that constitute the design (described by a netlist). Finally, the *structure* must be mapped into a *physical* design, which gives information about the location of the structure subparts on a chip.

Different synthesis steps need to be realised before obtaining a physical design, as shown in Figure 1.2 [112]. System-level synthesis includes hardware-software partitioning, resource allocation, task scheduling and task resource binding [107]. High Level Synthesis (HLS), also called behavioural synthesis, is the process of mapping a behavioural description at the algorithmic level to a structural register transfer level (RTL) description. This RTL description is in terms of functional units, memory elements and interconnections (i.e. multiplexers and buses) [27]. Logic synthesis translates an RTL description into an optimised netlist by applying its two main tasks: logic optimisation and technology mapping. Logic optimisation

transforms a combinatorial or sequential logic function into an equivalent gate level specification more suitable for technology mapping. Technology mapping transforms such gate level specification into a network of specific primitives provided by a particular technology [18]. Physical or layout synthesis performs tasks such as placement and routing. Placement assigns locations to various circuit components on the chip and involves the optimisation of objectives such as wire length, timing and power [28]. Routing generates the interconnections between the various circuit components [124]. After physical synthesis, a complete layout of the hardware is obtained, from which masks can be extracted for fabrication.

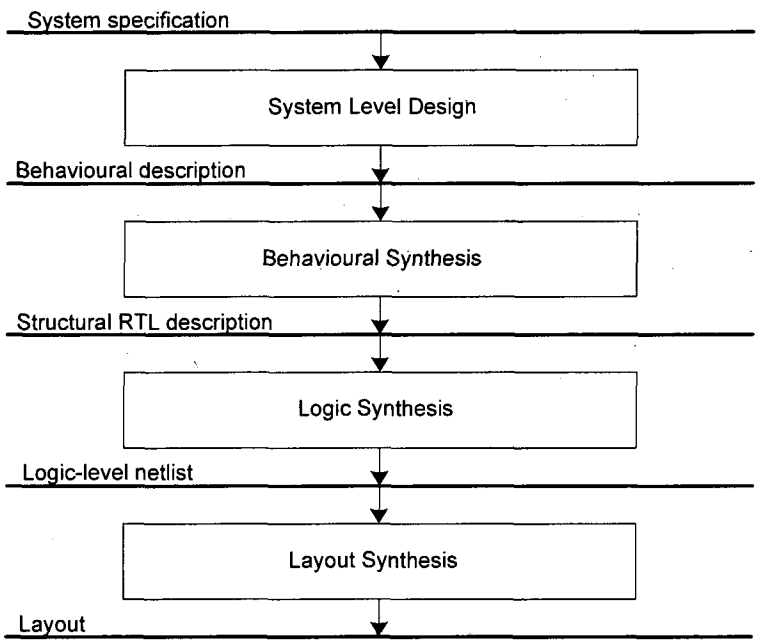


Figure 1.2 Synthesis flow and levels of abstraction [112]

Behavioural synthesis is becoming a mature research area that has been investigated for two decades now. However, a renewed interest in behavioural synthesis has been motivated due to the increasing complexity of digital systems, and the development of systems-on-a-chip (SoC) [35]. The use of a design methodology based on behavioural synthesis provides the following advantages [71], [19], [121]:

- Better complexity management. Designing at a high level of abstraction is an effective method to deal with the growing complexity of integrated circuits.
- Shorten verification/simulation cycle. The automatic generation of RTL code from behavioural descriptions avoids the slow and error-prone manual process, hence simplifying the design verification and debugging effort.

Moreover, shortening the design cycle allows meeting the aggressive time to market needs proper of most ASICs designs, lowering the development cost and enhancing the productivity.

- Ability to search the design space through the rapid exploration of different area-performance-power tradeoffs, focusing less on the details of logic and physical design.
- Higher quality of results can be achieved by integrating automatic high level optimisations together with physical information of logic and interconnects.

To demonstrate how a behavioural synthesis methodology generates efficient solutions and facilitates the design of electronic systems by using behavioural descriptions, consider the design of a RGB-YCrCb converter. Listing 1.1 shows a VHDL behavioural description of the RGB-YCrCb converter. As it can be seen, the description includes mathematical expressions and contains neither the cycle by cycle behaviour nor structure of the design. Assuming that this VHDL behavioural description is suitable for RTL synthesis using SynplifyPro, the generated design contains 4 combinational multipliers and 8 adders, utilising 105% of the total CLBs from a FPGA Xilinx Spartan XCS05. However, a design with only 1 multiplier, 1 adder and 1 subtractor, utilising only 50% of the same FPGA, can be obtained using the behavioural synthesis tool from Chapter 5.

Behavioural synthesis also offers a power optimisation opportunity due to the power analysis carried out at high level of abstraction. This is why researchers are still investigating and developing improved methods for low power behavioural synthesis as recently demonstrated in [7], [116] and [118]. Power analysis during behavioural synthesis allows evaluating the impact of various optimisations and design modifications on power, as well as validating that power budgets are met. Although power analysis carried out at high level tends to be less accurate than at lower levels of the design hierarchy, it is still helpful as it enables designers to forecast correctly the increase or decrease in power consumption after a design modification. Consequently, the role of lower level power analysis is limited to support lower level optimisations and assure that the design meets the power specification with a high level of confidence [112].

Listing 1.1 VHDL behavioural description of the colour space converter from RGB to YCrCb

```
library ieee;
use ieee.std_logic_1164.all;

package converter_defs is
        subtype vector is std_logic_vector(9 downto 0);
end converter_defs;
use work.converter_defs.all;

entity converter is
        generic(
                ACOEF, BCOEF, CCOEF, DCOEF: vector := "0101010101";
                YOFFSET, COFFSET: vector := "1010101010");
        port(
                R, G, B: in vector;
                Y, Cb, Cr: out vector);
end entity converter;

architecture converter_a of converter is
        signal y_: vector;
        process
                y_  <= ACOEF * (R - G) + G + BCOEF * (B - G);
                Y   <= y_ + YOFFSET;
                Cb <= CCOEF * (B - y_) + COFFSET;
                Cr  <= DCOEF * (R - y_) + COFFSET;
        end process;
end converter_a;
```

1.2 Contributions and thesis overview

This thesis presents an investigation into dynamic power minimisation algorithms for behavioural synthesis. This research makes the following contributions:

- a scheduler capable of producing a set of solutions with different power and resource requirements yet meeting the imposed time constraint [99].
- a datapath synthesis algorithm that performs simultaneously scheduling, binding, clock and operations throughput selection.
- a behavioural compiler based on the above scheduler and datapath synthesis algorithms that explores the design space finding good quality solutions in terms of area and power according to the user requirements [98].
- realisation of a motion vector reconstructor from the Berkeley MPEG-1 player developed in [40] to further validate the practical applicability of the proposed power-aware behavioural compiler.

This thesis is organised in seven chapters. Chapter 2 provides a review of behavioural synthesis concepts, outlines the different contributors of power consumption and presents combinatorial optimisation fundamentals that lay the

foundations of the algorithms developed in following chapters. Chapter 3 presents the literature review of low power behavioural synthesis. Chapter 4 introduces a new power-aware time constrained scheduling algorithm that explores efficiently the design space and obtains a set of power-area tradeoffs through voltage scaling after appropriate clock and operations throughput selection. Chapter 5 presents a power-aware behavioural compiler that given a time constraint, performs simultaneously scheduling, binding, clock and operations throughput selection using a simulated annealing based algorithm. Chapter 6 demonstrates the practical applicability of the algorithms developed in Chapter 4 and Chapter 5 through the realisation of the motion vector reconstructor from the Berkeley MPEG-1 player [40]. Finally, Chapter 7 summarises the main conclusions of the presented investigation and suggests future directions of research.

Chapter 2

Low power behavioural synthesis fundamentals

2.1 Introduction

To facilitate the understanding of low power behavioural synthesis, the basic concepts of behavioural synthesis and sources of power consumption are introduced in this chapter. Section 2.2 presents a general overview of behavioural synthesis. Section 2.3 describes the sources of power consumption together with some effective techniques proposed to reduce power consumption using behavioural synthesis. Section 2.4 introduces combinatorial optimisation with particular emphasis on simulated annealing, as such technique is employed in Chapter 5 to perform the design space exploration. Finally, concluding remarks of this chapter are given in Section 2.5.

2.2 Behavioural synthesis

In the context of behavioural synthesis, three different types of behavioural descriptions can be identified: data dominated, control dominated and control flow intensive [53]. Control flow intensive designs require a mix of control flow and data flow within the datapath, and may contain a large number of nested loops and conditionals. In data dominated designs such as digital signal processing and image processing applications, arithmetic operations are predominant. On the other hand, control dominated designs have very few arithmetic operations, being predominant nested conditional constructs, data dependent loops and comparisons. Data dependent loops make difficult to predict the number of times a loop body is executed. This means that the number of clock cycles required to execute a control

dominated design is highly dependent on the input data. In contrast, data dominated designs present loops that are data independent, facilitating the identification of the number of clock cycles required to execute the design. Area, power and delay are dominated by arithmetic units and registers in data dominated designs and by non arithmetic units, i.e. multiplexers, comparators, etc, in control dominated designs. Controllers in data dominated designs are simple and have very little impact on the area, delay and power of the circuit. However, in control dominated designs, controllers may significantly affect the total circuit delay and power [112].

Behavioural synthesis transforms the algorithmic or behavioural description (data dominated, control dominated or control flow intensive) of a circuit into a Register Transfer Level (RTL) implementation [120]. This implementation consists of a datapath and controller. The datapath executes the operations specified in the behavioural description and normally consists of functional units, registers and multiplexers. The controller provides to the datapath the necessary control signals, i.e. multiplexers select signals or registers load signals, to execute the operations.

The general steps of a behavioural synthesis design flow are shown in Figure 2.1. The first step requires the translation of the design behavioural description into an intermediate format that is more suitable for the synthesis process, such as data flow graph (DFG) and/or control flow graph (CFG), or a combination thereof known as CDFG [116]. The second step is scheduling, which assigns each operation to a specific control step (cstep) such that the design constraints are met. In the case of time constrained scheduling the aim is to reduce the resource requirements whereas in resource constrained scheduling the aim is to reduce the execution time. The third step involves allocation, which determines the functional units necessary to perform the operations and binding, which assigns operations to functional modules and values to registers. The aim of this step is to reduce the resources requirements such as registers and multiplexers. Using the information obtained after scheduling, allocation and binding, the controller that provides the signals to the datapath is designed. Finally, a gate level netlist is produced after synthesising the datapath and the controller using commercial logic synthesis tools.

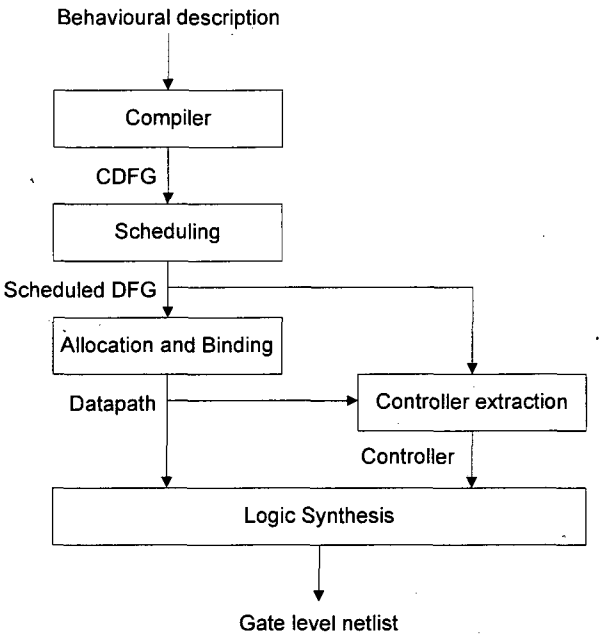


Figure 2.1 Behavioural synthesis design flow

2.2.1 Graph-based representation of behavioural descriptions.

The input of a behavioural synthesis system is a behavioural description, which can be specified in a hardware description language such as VHDL [116], Verilog [130], SystemC [16] or a high-level programming language like C [35]. The behavioural description expresses the system function in terms of mathematical algorithms or difference equations. For example, the function of the colour space converter from RGB to YCrCb [129] can be expressed like:

$$\begin{aligned} Y' &= ACOEF*(R-G)+G+BCOEF*(B-G) \\ Y &= Y'+YOFFSET \\ Cb &= CCOEF*(B-Y')+COFFSET \\ Cr &= DCOEF*(R-Y')+COFFSET \end{aligned} \tag{2.1}$$

where R, G and B are the RGB colour space inputs; Y, Cr and Cb are the YCrCb colour space outputs. $ACOEF$, $BCOEF$, $CCOEF$, $DCOEF$ are parameter values according to the standard, i.e. NTSC, PAL or YUV. $YOFFSET$ and $COFFSET$ are constants that facilitate offset compensation.

A VHDL behavioural description of the RGB-YCrCb converter was presented in Chapter 1 Listing 1.1 without the need of specifying its cycle by cycle behaviour or its structure. This limits the designer's role to specify the time interval within a particular operation has to be completed [151], allowing the behavioural synthesis

tool to determine the structure that better suits the specified requirements. The behavioural description is then compiled into an internal representation, i.e. DFG, which illustrates in a graphical way all the operations dependencies from the original behavioural specification. Figure 2.2 shows the DFG of the RGB-YCrCb converter, where operations are represented by circles and passing data values by directed edges.

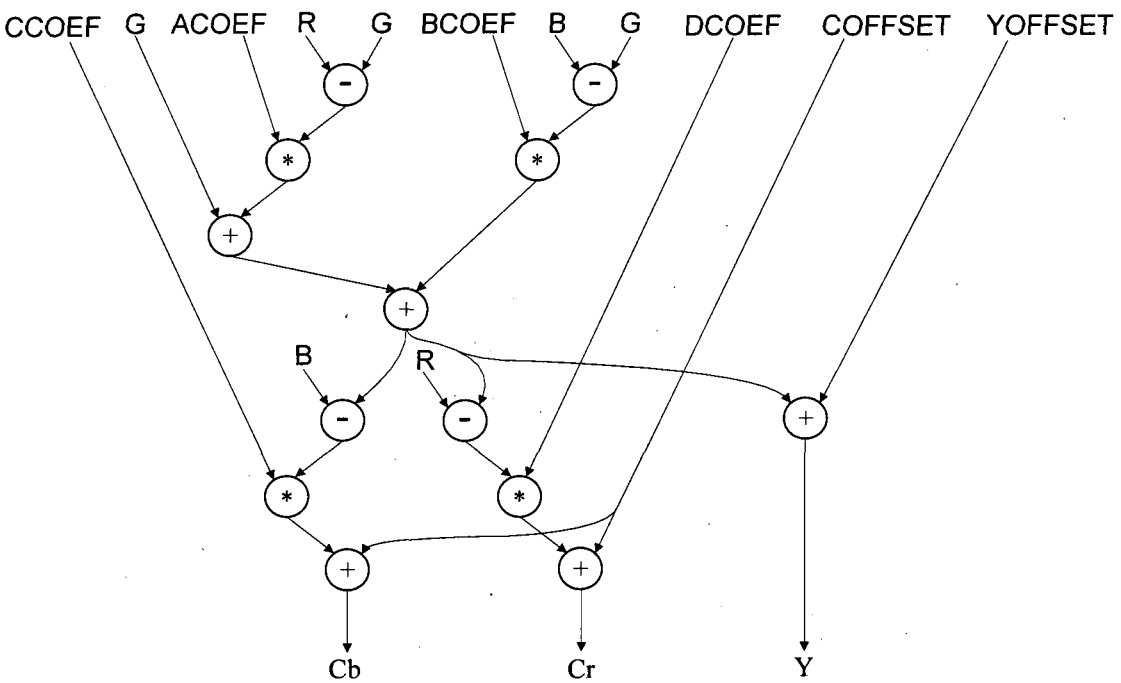


Figure 2.2 RGB-YCrCb converter DFG

2.2.2 Scheduling

Scheduling assigns the operations of the algorithmic description to control steps (csteps) defining the behaviour of the circuit cycle by cycle, i.e. specifying the order in which operations will be executed [60]. Each cstep corresponds to a time interval equal to the clock period. The number of csteps used to implement the specified behaviour is called schedule length (L_s). Various algorithms have been proposed to solve the scheduling problem in behavioural synthesis, the simplest ones being as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) [137]. Although both of them assume unlimited resources, they are important since they determine the fastest possible implementation, the critical path (in terms of control steps) and an upper bound on the number of required hardware resources. Moreover, ASAP and ALAP

are the basis for other algorithms that aim to minimise either the implementation area, i.e. time constrained scheduling (TCS), or the required schedule length to implement the behaviour, i.e. resource constrained scheduling (RCS). The following subsections illustrate in more detail the unconstrained schedules ASAP and ALAP, before presenting time constrained scheduling and resource constrained scheduling.

Unconstrained schedules

The ASAP scheduling algorithm assigns each operation of the DFG to the earliest possible cstep allowed by the data dependencies. For example, Figure 2.3 shows the ASAP schedule of the RGB-YCrCb converter considering that each operation can be executed in one clock cycle.

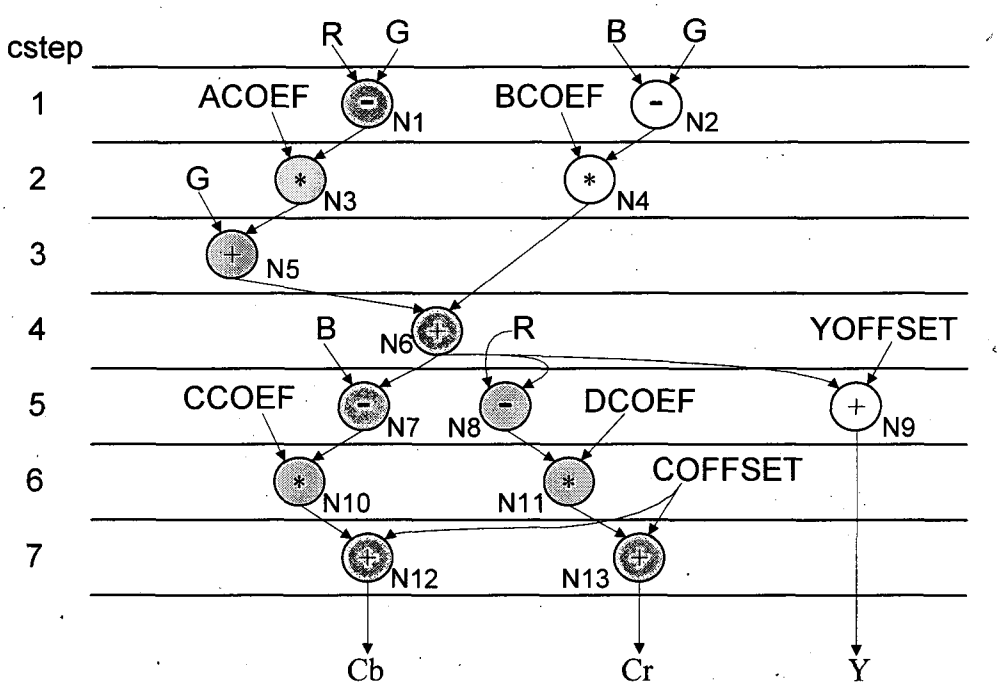


Figure 2.3 ASAP schedule of the RGB-YCrCb converter

Since ASAP schedules the operations in the earliest possible cstep, the implementation that executes the DFG using the least number of control steps is obtained, giving a lower bound on the schedule length. This lower bound of the schedule length defines the necessary number of csteps to execute all the operations in the critical path, i.e. 7 csteps in Figure 2.3. The operations that are in the critical path are N1, N3, N5, N6, N7, N10 and N12, or alternatively, N1, N3, N5, N6, N8,

N11 and N13. These operations can not be moved to any other cstep unless the data dependencies are violated or the schedule length is increased. In addition, ASAP gives the upper bound of required functional units which is equal to the maximum number of same type resources used in each cstep. In Figure 2.3, such upper bound consists of 2 multipliers, 2 adders and 2 subtractors.

To obtain the ALAP schedule it is necessary to define the schedule length in advance, and then assign each operation to the latest cstep possible respecting the data dependencies from the DFG. Figure 2.4 shows the ALAP schedule of the RGB-YCbCr converter considering a schedule length of 7 csteps.

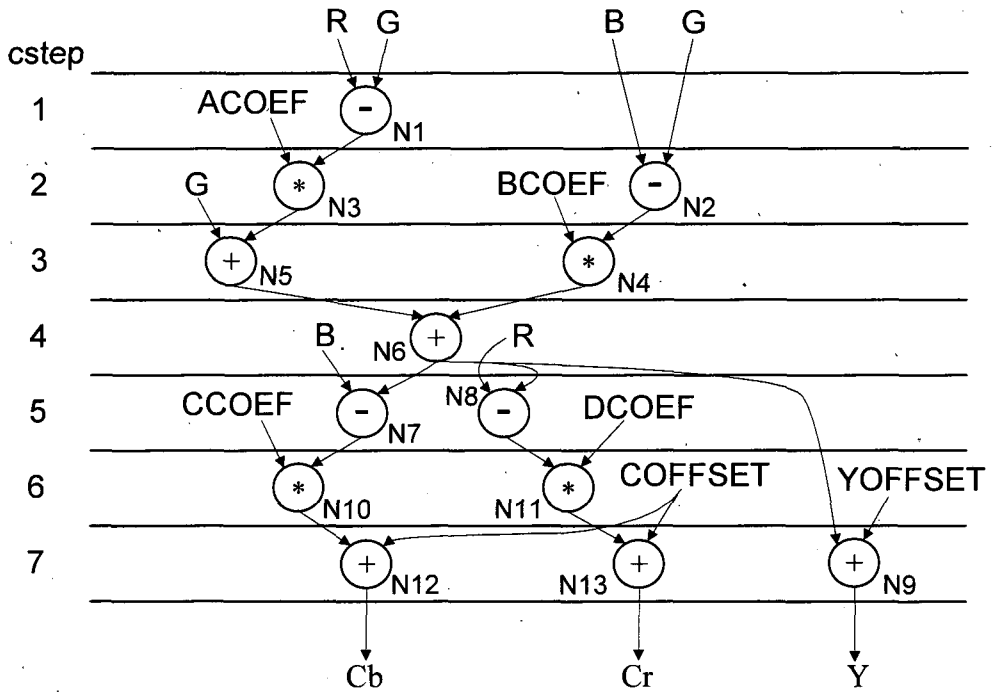


Figure 2.4 ALAP schedule of the RGB-YCrCb converter

Resource Constrained Scheduling (RCS)

The aim of resource constrained scheduling is to assign operations to control steps such that execution time is minimised for a given number of functional modules. Although the RCS problem can be formulated and optimally solved using Integer Linear Programming (ILP) [13], its computational complexity is exponential. To overcome this overhead, heuristic methods with polynomial time complexity are used. A commonly used heuristic algorithm to solve the RCS problem is list scheduling [97]. This algorithm puts all operations whose inputs are available into a

ready list. This list is then sorted according to a priority function. The operations in the sorted list are subsequently assigned until all operations are scheduled or all hardware resources have been used. The list scheduling algorithm is the base for more complex algorithms such as Force Directed List Scheduling (FDLS) [101]. Here, the selection of a candidate operation to be scheduled in a given time step is done by using the concept of force.

Time Constrained Scheduling (TCS)

The aim of time constrained scheduling is to assign operations to csteps such that the number of functional modules is minimized for a given execution time. As in the case of RCS, the time constrained scheduling problem can be formulated and solved optimally using ILP [13]. The TCS problem can also be solved using heuristic algorithms such as force directed scheduling (FDS) [101]. The calculation of the force is based on distribution graphs which represent the sum of the probability values that the operations are assigned to certain cstep. Unlike the FDLS algorithm that considers each schedule step at a time, FDS considers the operations one at a time for scheduling. Other algorithm based on the distribution graph concept is [58], where a least mean square error function is used to schedule operations in sequence. Although this algorithm obtains similar results than FDS [101], it does not evaluate the influence of all operations on the schedule before the most appropriate operation is selected and scheduled. This results in a simple implementation with low computational complexity.

Multicycled operations

During scheduling, the operations can be executed over one clock cycle (single cycle operations) or n clock cycles (multicycled operations), where n is any integer number greater than 1. In Figure 2.5, it can be seen that O1 is a single cycled operation whereas O2, and O3 are multicycled operations. Multicycling is a widely investigated technique for performance [137] and/or area [26] optimisation in behavioural synthesis. Scheduling DFGs that contain multicycled operations has been addressed in [101] and [58]. More recently, multicycled functional units (FUs) have been employed to reduce the power consumption of digital designs generated

using a behavioural synthesis methodology [31].

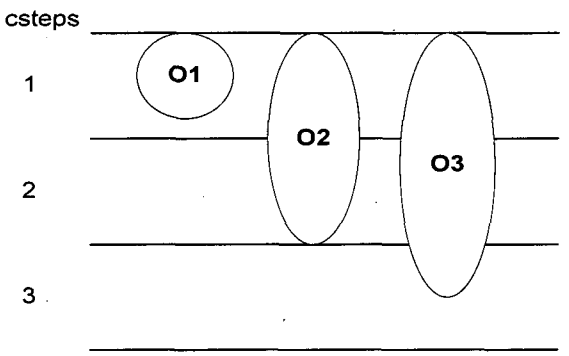
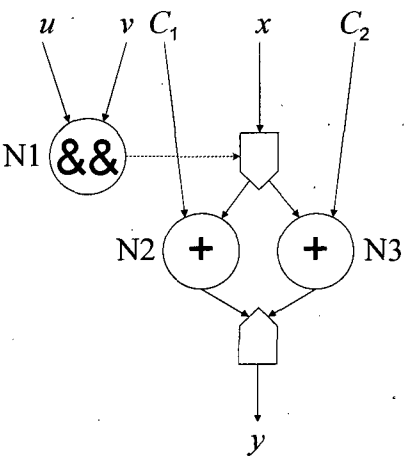


Figure 2.5 Single cycled and multicycled operations

Conditional branches

Behavioural descriptions that contain conditional statements, i.e. if-then-else, result in DFG with conditional branches where operations are mutually exclusive, which means that they are never executed at the same time. Consequently mutual exclusive operations can be scheduled in the same cstep without increasing the number of functional units required. Conditional statements are represented using fork and join nodes in the DFG [145]. Figure 2.6 shows a conditional statement and its respective representation in the DFG. Note that the branch to be executed depends on the value, i.e. true or false, obtained after evaluating a given condition, i.e. (*u and v*). Such value is represented in the DFG as a dotted line in Figure 2.6.

```
if (u and v) then
        y = x + C1;
else
        y = x + C2;
endif;
```



a) VHDL description b) representation in a DFG

Figure 2.6 Conditional statement

From Figure 2.6, it can also be seen that operations N2 and N3 are mutually exclusive and can be scheduled onto the same functional unit and into the same control step without increasing the resource requirements. Some of the algorithms that solve the problem of scheduling operations in a DFG with conditional branches are [58], [145], [54] and [136].

2.2.3 Allocation and binding

In this thesis, the terms allocation and binding are used according to the definition given in [119], which precisely distinguishes between the two. Allocation determines the amount of resource requirements (functional, storage, and interconnect units) that can be shared by operations and data transfers specified in the behavioural description. The allocation process is usually solved as two-step process, before or during scheduling and during binding. Module allocation is performed before scheduling in the case of resource constrained scheduling or during scheduling in the case of time constrained scheduling. The allocation of registers and multiplexers is performed during binding. Binding maps operations to functional modules and data transfers to storage units in the synthesised structural design, hence, determining the interconnect resources to implement the design. The combination of the tasks allocation and binding is referred to as datapath synthesis. Each of these tasks can be transformed into a graph-theoretical problem such as graph colouring, clique covering or clique partitioning [21]. The left edge algorithm [38] is capable of solving the colouring problem and has been used efficiently for register binding. This popular binding algorithm was used in the datapath synthesis system described in Chapter 5 to generate initial solutions, from which new solutions are generated during a simulated annealing process. The allocation and binding problem can also be formulated using Integer Linear Programming (ILP), which guarantees global optimum solutions but at the expense of exponential worst case complexity. This overhead can be overcome using heuristic techniques such as genetic algorithms or simulated annealing [117].

2.2.4 Clock selection

A key decision during behavioural synthesis is choosing the clock period to schedule the data flow graph (DFG) operations into control steps. Clock selection refers to the procedure of designating an appropriate clock period for the controller/datapath circuit [112]. The clock period together with the execution time of the DFG (or design time constraint), determine the length of the schedule as outlined in [96]:

$$T = Ls * Tclk \quad (2.2)$$

where T is the time constraint or DFG execution time, Ls is the length of the schedule in terms of *csteps* and $Tclk$ is the clock period.

The clock selection problem is not new in the context of behavioural synthesis and has been addressed in previous research, showing the significant effect of clock choice on the design in terms of area [5], [14], performance [96], [115], and power [73], [99]. Power consumption can be affected directly or indirectly by the choice of clock period in the following ways [112]:

- Larger values of clock period lead to a significantly increase in the glitching power consumption since the schedules obtained have more functional unit chaining. However, the increase in functional unit chaining helps to inhibit functional unit sharing, which may sometimes lead to larger and more power consuming datapaths.
- Larger values of the clock period lead to lower power consumption in the clock network and registers since the design requires fewer clock cycles to process each input.

An optimal clock period can be found by analysing all possible schedules with different clock lengths and then choosing the best value [115]. However, this would require considerable time which makes it highly undesirable. Clock selection is a key concept used throughout this thesis to minimise power in behavioural synthesis.

2.2.5 Controller design

Once the datapath has been obtained using behavioural synthesis, a controller needs to be synthesized to complete the design. This controller provides signals to the datapath to execute the operations according to the schedule obtained after behavioural synthesis. Such signals comprise for example, enable signals for

functional modules, load signals for registers and select signals for multiplexers. The most popular style of controller architecture for digital design is finite state machines (FSMs) [35]. The design of the finite state machine from scheduled designs starts with the specification of which operations will be executed in each control state. This process is simple in purely data flow designs, where no conditional branches are present and each cstep represents a control state. However, in presence of conditional branches, mutual exclusive operations that are executed in the same cstep can be either assigned to the same control state or to a control state based on the branch they are in. The former method is known as global slicing and the latter as local slicing [131]. Both slicing techniques are illustrated in Figure 2.7, where the states are demarcated by dashed lines and marked as *S1*, *S2*, *S3* and so on. The local slicing method has been used for the example in Figure 2.7a. It can be seen that operations that are executed in the same cstep and under the same branch are assigned to a unique state. Hence, operations N4 and N5 are assigned state *S3* and operation N6 is assigned to state *S6*. However, in global slicing, operations that are executed in the same cstep and on mutually exclusive conditional branches are assigned to the same state. Figure 2.7b shows that operations N4, N5 and N6 are all assigned to the same state *S3* after using global slicing.

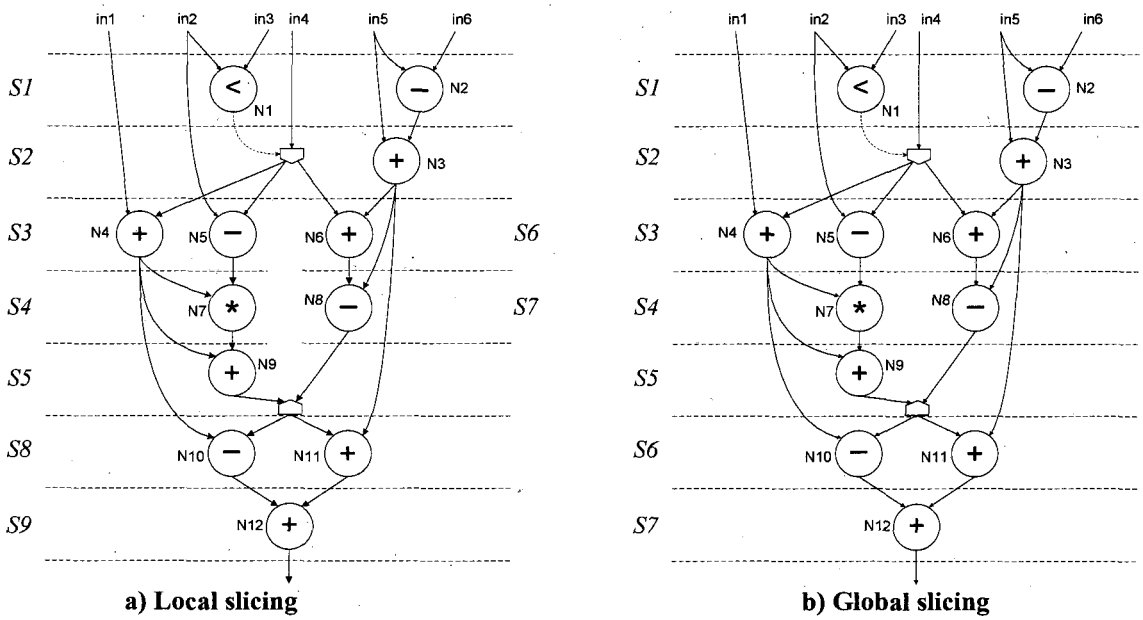


Figure 2.7 Assignment of operations to control states

Global slicing requires fewer states in the controller than local slicing, leading to smaller state machines. However, status registers are required to store the

information about which mutual exclusive operations will be executed in the state. In [139], the authors presented a complete study on these two types of controllers and concluded that using global slicing with status registers always results in designs with lower area. These results were later supported by [35], where the authors claim that the larger number of states required for local slicing leaded to poorer finite state machine optimisation.

2.3 Sources of power consumption

In the past, performance and area were the primary considerations in electronics system design. However, power considerations have become an increasingly dominant factor in the design of portable systems [70]. The design of portable devices certainly requires consideration of the peak power consumption for reliability and proper circuit operation, but the time averaged power is proportional to the battery weight and volume required to operate circuits for a given amount of time. Moreover, average power reduction results in [76]:

- extension of the battery life time, which depends on its A·h (ampere hour) rating. When the battery has high power dissipation its life time may reduce due to high ampere consumption.
- reduction of the chip operating temperature, increasing the system reliability. It is estimated that components failure rates nearly double for each 10°C increase in the operating temperature.
- reduction of cooling and packaging costs.

Average power dissipation has two main contributors, the capacitive switching power $P_{sw.cap.}$ and the leakage power $P_{leakage}$, both of equal importance for nanometre technologies [65]. $P_{sw.cap.}$ is due to the charge and discharge of the capacitances associated with each node of the circuit. Neglecting the internal capacitances, the power consumption of a generic CMOS gate is given by:

$$P_{sw.cap.} = \frac{1}{2}\alpha C_L V_{dd}^2 f \tag{2.3}$$

where α is the switching activity and it is the sum of the probabilities that a rising or a falling transition occurs on the output in each clock cycle, C_L is the load capacitance, V_{dd} is the supply voltage and f is the frequency. The capacitive

switching power is also called dynamic power. Dynamic power has usually been the dominant component of the total power consumption [104]. However, in many of today's designs, leakage power has become comparable to dynamic power [103]. For example, leakage power can contribute as much as 42% of the total power in the 90nm process technology generation [65].

$P_{leakage}$ is the product of the leakage current $I_{leakage}$ and the supply voltage V_{dd} [108]:

$$P_{leakage} = I_{leakage} V_{dd} \tag{2.4}$$

The leakage current $I_{leakage}$ has two sources: *i*) the current I_{diode} that flows through the reverse-biased diode junctions of the transistors located between the source or drain and the substrate, and *ii*) the subthreshold current $I_{subthreshold}$. The contribution of I_{diode} to the total leakage current is very small and can be ignored. On the other hand, $I_{subthreshold}$ grows exponentially as threshold and supply voltages are scaled down in today's processes. The current $I_{subthreshold}$ can be computed as outlined in [140]:

$$I_{subthreshold} = I_0 e^{\left(\frac{V_{gs}-V_{th}}{nV_t}\right)}\left(1 - e^{\frac{-V_{ds}}{V_t}}\right) \tag{2.5}$$

where V_{gs} is the gate-source voltage, V_{ds} is the drain-source voltage, V_{th} is the threshold voltage, V_t is the thermal voltage, n is the subthreshold slope coefficient, and I_0 is a technology constant.

Dynamic and leakage power can be technology dependent or/and design dependent. Some of the technological measures that reduce dynamic and leakage power are respectively the use of reduced voltage processes and the use of multiple voltage thresholds. Dynamic power can also be reduced by taking measures in the design, such as voltage scaling or switching activity reduction [134].

From equation (2.3), it can be inferred that dynamic power savings can be achieved by reducing one or more of the following parameters during behavioural synthesis:

- Supply voltage
- Clock frequency
- Switching activity
- Load capacitance

Some techniques developed for low power behavioural synthesis comprise:

- Applying transformations that allow supply voltage or switching capacitance reduction [8], [39]. Examples of such transformations are control step

reduction, operation reduction, operation substitution, loop shrinking, retiming and loop unfolding.

- Optimising the wordlength of functional units [3]. To facilitate this, fast and reliable word-level power models may be used to estimate the power consumption in the functional units [17].
- Switching off operators [123] or partitions of the chip [118] by disabling the clock signal during inactive periods.
- Using lower supply voltages that decrease quadratically the power consumption. Mixing voltages in a circuit is also possible, applying low voltages to the operations that are not in the critical path and high voltages to the critical path operations such that the time constraint is met [122].

Chapter 3 presents a literature survey on power minimisation algorithms for behavioural synthesis.

2.4 Combinatorial optimisation

Many of the optimisation problems encountered in Computer-Aided Design (CAD) for Very Large Scale of Integration (VLSI) are combinatorial [27]. This section introduces combinatorial optimisation and in particular provides an introduction to simulated annealing as such technique is employed in Chapter 5 to perform the design space exploration. The objective of an optimisation problem is to find a solution that can be measured in terms of a cost function such that its value is maximum or minimum [21]. If a specific case of an optimisation problem can be characterised by a finite set of discrete variables, i.e. they can only assume a finite number of distinct values, the problem is called a combinatorial optimisation problem [27]. Formally, a specific case of a combinatorial optimisation problem can be defined as $< S, f >$, where S represents the solution space and f is a cost function of the form f: $S \rightarrow \mathbf{R}$. In the case of minimisation, the problem is to find a global optimum solution $s_{opt} \in S$ which satisfies

$$f(s_{opt}) \leq f(s), \text{ for all } s \in S \tag{2.6}$$

In the case of maximisation, s_{opt} satisfies

$$f(s_{opt}) \geq f(s), \text{ for all } s \in S \tag{2.7}$$

The set S_s of all solutions that are in some sense close to a solution $s \in S$ is called the

neighbourhood N of s. $\hat{s} \in S$ is called a local optimum with respect to N if $\hat{s}$ is better than, or equal to, all its neighbouring solutions with regard to their cost. More specifically, in the case of minimization, $\hat{s}$ is called a local minimum if

$$f(\hat{s}) \leq f(s), \text{ for all } s \in S_{\hat{s}} \quad \textbf{(2.8)}$$

And in the case of maximisation, $\hat{s}$ is called a local maximum if

$$f(\hat{s}) \geq f(s), \text{ for all } s \in S_{\hat{s}} \quad \textbf{(2.9)}$$

Some combinatorial problems can be solved in polynomial time, therefore, called tractable. Otherwise, they are called intractable. There exist three possibilities when trying to solve an intractable problem: exact, approximation and heuristics algorithms [27]. Exact algorithms provide the exact solution but may have a high computational cost that prevents their use on typical size problems. The methods of this category are called "general purpose" since they can be applicable to almost any combinatorial optimisation problem. Some examples are: exhaustive search, backtracking with branch and bound, dynamic programming and integer linear programming. Approximation algorithms find a solution whose cost is within a certain margin of the optimal cost only when involving problem-specific issues in the analysis of the algorithm. Heuristic algorithms do not guarantee an optimal solution but seem to be the only way to solve problems in CAD for VLSI [117]. Some examples of heuristics techniques are: problem-specific, local search, tabu search, simulated annealing and genetic algorithms. In Chapter 5, a simulated annealing technique was employed to perform the design space exploration. Simulated annealing is relatively easy to implement when compared for example with genetic algorithms and provides the ability to solve combinatorial problems with polynomial time complexity [1]. Furthermore, simulated annealing is independent of the considered optimisation problem and can avoid poor local minimum while approaching to the global minimum.

2.4.1 Optimisation using simulated annealing

Simulated annealing is a combinatorial optimisation method based on the simulation of the annealing process [1]. In condensed matter physics, annealing denotes a thermal process for obtaining low energy states of a solid in a heat bath. As far back as 1953, Metropolis [74] presented a method to simulate the evolution to thermal

equilibrium of a solid in a heat bath. In this method, a new state with energy E_2 is generated after applying a perturbation mechanism to the current state with energy E_1. The new state with energy E_2 is accepted if the energy difference, $E_2 - E_1$, is less than or equal to 0. If the energy difference is greater than 0, the new state is accepted with probability

$$p = e^{\frac{E_1 - E_2}{k_B T}} \tag{2.10}$$

where T is the temperature of the heath bath and k_B is a physical constant known as Boltzmann constant. By repeating this process for a large number of perturbations, the system eventually evolves into thermal equilibrium, approaching to the Boltzmann distribution the probability distribution of the states [64].

The idea of applying an analogy that links the simulated annealing of solids with combinatorial optimisation was first published in [56], which also presented some applications for VLSI design automation, i.e. placement and global routing. The analogy is based on the following equivalences:

- solutions in a combinatorial optimisation problem are equivalent to the states of a physical system
- the cost of a solution *f(s)* is equivalent to the energy E of the state
- the control parameter c is equivalent to the factor $k_B T$
- the perturbation of the particles in the physical system then becomes equivalent to a trial in the combinatorial optimisation problem.

Then, the simulated annealing algorithm can be seen as an iteration of Metropolis algorithms evaluated at decreasing values of the control parameter. The acceptance criterion determines whether the transition from a solution s_1 to s_2 is accepted by applying the following acceptance probability:

$$P(s_1 \to s_2) = \begin{cases} 1 & \text{if } f(s_2) \le f(s_1) \\ e^{\frac{f(s_1) - f(s_2)}{c_k}} & \text{if } f(s_2) > f(s_1) \end{cases} \tag{2.11}$$

The pseudocode for the simulated annealing algorithm [1] is shown in Listing 2.1, where c_k represents the control parameter value and L_k is the number of transitions generated at the k^{th} iteration of the Metropolis algorithm.

A characteristic of simulated annealing algorithms is the acceptance not only of improvements in cost, but also deteriorations in cost to a limited extent. Initially, at large values of c, high increases in cost will be accepted; as c decreases, only small

increases in cost will be accepted and finally, as c approaches to 0, only improvements in cost will be accepted. This feature allows simulated annealing to escape from local minima. The probability of accepting solutions with higher cost is realised by comparing the value of $\exp((f(i) - f(j))/c)$ with a random number obtained from a uniform distribution on the interval [0,1). The convergence speed of the algorithm depends on the selection of the parameters L_k and c_k.

Listing 2.1 Pseudocode for simulated annealing [1]

```
procedure SIMULATED_ANNEALING;
begin
        INITIALISE (i_start, c_0, L_0);
        k = 0;
        i = i_start;
        repeat
                for l = 1 to L_k do
                        GENERATE ( j from S_i);
                        if f(j) ≤f(i) then i = j;
                        else if exp( (f(i) - f(j)) / c_k ) > random[0,1) then i = j;
                end for;
                k = k+1;
                CALCULATE_LENGTH (L_k);
                CALCULATE_CONTROL (c_k);
        until stopcriterion;
end;
```

A finite time implementation of the simulated annealing algorithm can be realised by generating homogeneous Markov chains of finite length at decreasing values of the control parameter. To achieve this, the parameters that rule the convergence of the algorithm must be specified. The selection of such parameters is referred as cooling schedule and determines:

- an initial value of the control parameter c_0
- a decrement function for decreasing the value of the control parameter
- a final value of the control parameter specified by a stop criterion
- a finite length of each homogeneous Markov chain

In [56], the authors developed a simple cooling schedule that has been used in many applications of the simulated annealing algorithm. This cooling schedule selects the initial c_0 such that approximately all new solutions are accepted. The decrement rule is given by

$$c_{k+1} = \alpha \cdot c_k \qquad (2.12)$$

where α is typically between 0.8 and 0.99. The execution of the algorithm is terminated if the value of the cost function of the solution obtained in the last trial of a Markov chain remains unchanged for a number of consecutive chains. The length of the Markov chain is such that equilibrium is reached for each value of the control parameter. This means that the length of the Markov chain may vary with the control parameter c_k.

2.5 Concluding Remarks

This chapter has presented the key principles of low power behavioural synthesis for the realisation of digital systems. The process of behavioural synthesis, its three main tasks: scheduling, allocation and binding, and interrelated tasks such as clock selection have been reviewed. It has also been discussed the key parameters that can be used to reduce dynamic power in the context of behavioural synthesis. Due to the quadratic dependence of power on voltage, reducing the voltage has a higher impact on power consumption than reducing the switching activity, capacitance or frequency. Finally, combinatorial optimisation and simulated annealing have been briefly outlined to lay the foundations for the algorithm developed in Chapter 5.

Chapter 3

Literature review of related work

3.1 Introduction

Power consumption minimisation can be carried out at a single task (scheduling, allocation, binding) or simultaneously at the various tasks of behavioural synthesis. This is reflected in the literature survey presented in this chapter. Section 3.2 and Section 3.3 outline respectively the main reported work carried out on the low power scheduling problem focusing at two sources of power reduction: supply voltage and frequency. Section 3.4 describes the proposed algorithms that reduce power consumption by decreasing the switching activity in the design during the binding task of behavioural synthesis. Section 3.5 considers the reported approaches that perform simultaneously scheduling and binding with the aim of reducing power dissipation in the design. Section 3.6 completes the chapter with concluding remarks.

3.2 Low power scheduling based on voltage reduction

Raje et al. [114] presented a time constrained scheduling algorithm that reduces power dissipation by using multiple supply voltages (MSV). Scheduling with MSV considers the assignment of as many operations as possible to modules that operate at low voltage and the remaining operations are assigned to modules that operate a higher voltage. The number of allowable voltages is set according to the technology and the designer's preference. Proofs for optimality of this algorithm are presented in [113] assuming that the supply voltage versus latency curve is the same for all functional units, i.e. adders and multipliers. This is a simplistic and rather unrealistic assumption that may lead to suboptimal solutions and was addressed by Chang et al.

[10]. Here, a dynamic programming algorithm that solves the time constrained MSV scheduling problem in both non-pipelined and functionally pipelined data-paths was presented. This algorithm includes the cost of level shifters, which are necessary to transfer data between functional units operating at different voltages. To calculate the energy dissipation of the solutions obtained, the algorithm uses respectively the energy values from Table 3.1 and Table 3.2 for the level shifter and functional units, which were implemented in 1μm technology. The authors consider that the propagation delay through a level shifter, i.e. 1ns, is negligible compared to the propagation delay through the modules. The delay cost for the level shifter is then absorbed into the delay of the functional units they follow, because in the module library, the minimum module delay is at least 20 times larger than the level shifter delay.

Table 3.1 Average energy of a 16-bit level shifter [10]

x/y	2.4V	3.3V	5V
2.4V	0	64.0	128.0
3.3V	49.6	0	142.4
5V	88.0	104.0	0

Table 3.2 Energy dissipation of datapath functional units [10]

	multiplier		*adder*	
Voltage	Energy (pJ)	Delay (ns)	Energy (pJ)	Delay (ns)
2.4V	3877.5	295.4	30.10	60.27
3.3V	7330.9	181.2	56.91	36.14
5V	16829	103.7	130.65	20.40

More recently, the problem of low power time constrained scheduling using MSV has been solved using novel formulations. For example, Tsai et al. [63] proposed a hybrid algorithm than combined simulated annealing with genetic algorithm to solve the low power scheduling problem using dual V_{dd} and dual V_{th}. The goal of this approach is to minimise the power and delay penalty of the design. Each operation in the DFG is represented by a chromosome that includes information such as V_{dd}, V_{th} and control cycles. An individual is represented with the same number of chromosomes as operations in the DFG. The scheduler starts producing the first generation, and generates many individuals. Then, it randomly selects two individuals as the parents, and performs the crossover and mutation operations to

generate two children. After that, the scheduler evaluates the power and delay of two parents and two children, and decides the Boltzmann trial winner.

MSV is a technique for power minimisation that can be applied not only during time constrained but also during resource constrained scheduling, as shown by Shiue et al. [126]. Here, two algorithms for power minimisation were described, a time constrained scheduling (TCS) scheme and a resource constrained scheduling (RCS) scheme. In the TCS algorithm, operations that have the same mobility form a group, and groups with the same mobility form a set. Then the groups in a set are sorted according to a priority function that includes the number of operations in the group. Groups with mobility zero are assigned to high voltage and groups with high priority are assigned to low voltage. The RCS algorithm is based on list scheduling, hence operations are sorted according to a priority function based on the depth, mobility and level shifter requirements. The algorithm first assigns operations with high priority to low voltage resources and the remaining operations to high voltage resources. The authors improved these algorithms in [127], considering the effect of switching activity on the power consumption of the functional units, the interconnection complexity [72] and the power consumed by level shifters. Both algorithms (TCS and RCS) try to reduce the number of level shifters using heuristics. These algorithms produce schedules where the operations executed at the same control step may be assigned to modules at different voltage complicating the design of the controller. This was addressed by Kumar et al. [61], who proposed a RCS algorithm that identifies available parallelism in an initial schedule and creates different zones while maximising the hardware sharing. Zones group parallel operations or smaller zones. Zones are able to be moved together from one voltage to the lower next reducing the power consumption at the expense of an increased latency. This increased latency is assumed to be compensated through pipelining. Zones allow scheduling all operations in a control step at the same voltage, simplifying the realisation of the controller.

So far, scheduling algorithms targeting the optimisation of a single objective, i.e. area (TCS) or performance (RCS) have been presented. Scheduling algorithms that aim towards the optimisation of two objectives, i.e. area and performance are described in the following. Johnson et al. [48] developed MESVS (Minimum Energy Schedule with Voltage Selection), an ILP algorithm that incorporates MSV selection

and level shifter costs into energy optimisation of schedules. The objective function is an estimate of datapath energy dissipation as a function of supply voltages. The ILP formulation was permitted to choose from voltages ranging from 1.5V to 5V in 0.5V increments and selected voltages were required to differ by at least 1V. Although this approach reduced power consumption significantly, better results can be obtained if voltages are selected from a continuous range instead of a discrete one. This is shown by Johnson et al. in [49], where an ILP algorithm called MOVER (Multiple Operating Voltage Energy Reduction) was proposed. MOVER initially finds one minimum voltage for an entire schedule. It then determines a second voltage for operations where there is still slack. New voltages can be introduced and minimised until no schedule slack remains. This paper also discusses the effects of using MSV on IC layout and power supply requirements. Some of such effects include additional power and ground pins, increase in area due to the routing of the supplies and partitioning the chip into separate regions. In [47], Johnson et al. provide further details about the resources (multiplier, adder, register and level shifter) used by MOVER. Typical energy dissipation of the level shifter was found to be on the order of 5 to 15pJ per switching event per bit, given a 0.1pF load. Typical propagation delay ranges were approximately 1ns for level conversions such as 3.3V to 5V or 2.4V to 3.3V. However, a level conversion from 2.5V to 5V had a delay of about 2.5ns, whereas a 2V to 5V conversion had a delay of nearly 5ns. Energy and delay values for other 16-bit resources used by MOVER are shown in Table 3.3. Power and delay values for each resource were obtained after simulation with HSPICE using 0.8μm MOSIS library models. Area of the resources used by MOVER is represented by the weights of Table 3.4, which are proportional to their transistor count.

Table 3.3 Energy and delay values used by MOVER

	Energy (pJ)	Delay (ns)
adder	84	12.0
multiplier	2966	18.5
register	312	0.48

Table 3.4 Proportional weights to the transistor count of the resources

	multiplier	adder	register	level shifter
weight	16	1	0.75	0.15

Lin et al. [66] presented an ILP model that explores the design space considering time constraints alone, resource constraints alone, and time and resource constraints together. The objective function of the ILP model is a function of two costs: hardware and power. Each cost is associated with a weighting factor, which is set by the user to express a preference of one term over the other. The ILP method is practical usually for small size DFGs and may turn out to be computationally very expensive due to its worst case exponential complexity. Consequently, efficient search methods for the dynamic power minimisation problem that are applicable to large size DFGs have been developed. For example, Lin et al. [66] described an heuristic model where scheduling is performed according to a delaying gain that is calculated based on a linear priority function that includes some operations characteristics, such as the power gain after using a lower voltage and the mobility. The priority function also considers the average utilisation of functional units in the schedule. The operation with the highest delaying gain is selected and a lower voltage is assigned to it. If a feasible schedule is obtained, the iterative procedure continues, otherwise an attempt to reduce the voltage in the operation with the second highest delaying gain is made.

Katkoori et al. [50] extended the FDLS algorithm proposed by Paulin et al. [101] to heuristically determine the best control step for an operation such that the overall power consumption is minimised without sacrificing the design throughput. While scheduling operations in any control step, if sufficient resources are not available then excess operations are deferred based on their "force". At the operation level, a power cost function that captures the effect of an operation's deferral on the total power consumption of the design was developed. For each operation, the power cost is combined with its force to yield a compound cost used to decide which operation to defer. The deferred operation must have the largest power consumption in current time-step and have the least force. Other scheduling algorithm that uses the concept of force was proposed by Gupta et al. [33]. Here, the concept of force is used to model the switched capacitance of combinations among DFG operations which could share a resource and the probability of selecting such a combination.

Manzak et al. [70] presented a low power scheduling algorithm that operates in two passes. In the first pass, the minimum computation time is obtained using resource

constrained scheduling. In the second pass, the difference between the time constraint and the minimum computation time is distributed among the DFG operations. The distribution procedure tries to implement the minimum energy relation derived using the Lagrange Multiplier method in an iterative fashion. In each iteration, increasing number of resources with high energy-delay ratio are disabled from the set of resources and operations are scheduled using a list-based algorithm.

Recently, Hariyama et al. [37] used a genetic based algorithm to solve the scheduling and module selection problem using multiple supply voltages to minimise power consumption under time and resource constraints. This algorithm considers only MSV but not partitioning the chip into voltage islands, which may cause some physical layout problems, i.e. complex routing of interconnections and supply voltage lines. Wang et al. [138] overcame this problem by presenting a hybrid algorithm that combined simulated annealing with tabu search to minimise power consumption under resource and timing constraints when performing simultaneously partitioning and scheduling. The simulated annealing algorithm searches for the operating voltage whereas the tabu search technique searches for the cluster of the operations in the DFG. This algorithm obtains solutions in less time but with higher power consumption than an algorithm based only on tabu search [68] developed by the same authors.

3.3 Low power scheduling based on frequency scaling

Krishna et al. [59] developed a time and resource constrained scheduling algorithm which combines the concepts of Dynamic Frequency Clocking (DFC) and Multiple Supply Voltages (MSV). DFC takes advantage on the fact that different functional units such as multipliers and adders can be clocked at different frequency according to their critical path delay. Consequently, all units can be driven by a single clock line that changes at run time depending on the functional unit active in that clock cycle. This algorithm consists of two stages. In the first stage, an initial schedule based on DFC is generated, where control steps are clocked at different frequencies and operations are grouped such that the fastest functional units operate concurrently. All the functional units perform single cycle operations. In the second stage, operations are moved from one control step to another with the objective of meeting

the time constraint and minimising power consumption using MSV. The target architecture for the scheduled design includes a datapath, controller and the dynamic clocking unit (DCU).

The principles of MSV and DFC can also be applied to solve the low power resource constrained scheduling problem, as shown by Radhakrishnan et al. [109]. Here, voltage is assigned to the operations according to their type, i.e. multiplications or additions, and whether they are in the critical path or not. The frequency for each control step is assigned according to the number of operations of each type that are in the control step. Recently, Murugavel et al. [92] proposed a game theoretic based approach for simultaneous voltage and frequency scaling with the aim of minimising power when scheduling under resource constraints. Game theory was also used by the same author to model the low power binding problem [91] and low power of simultaneous scheduling and binding [93].

The problem of low power scheduling using DFC and MSV was further investigated by Mohanty et al. [86] but considering only time constraints. The authors developed an algorithm that schedules lower frequency operators at earlier control steps and delays higher frequency operators to later control steps. Next, the schedule is modified by moving operations from one control step to another with the objective of meeting the time constraint. Then, the algorithm finds a suitable clock period and assigns appropriate voltage. In [83], Mohanty et al. developed an Integer Linear Programming (ILP) based scheduling approach that uses MSV and DFC principles to minimise the energy delay product in a design. Same principles were integrated by Mohanty et al. [80], in another ILP based scheduling algorithm to minimise a parameter called cycle power function, which captures the peak power, the peak power differential and the average power of the datapath. More recently, Mohanty et al. [77] developed ILP based algorithms to minimise simultaneously peak and average power. In [85], Mohanty et al. developed a low power resource constrained scheduling algorithm that employs multiple supply voltage and dynamic frequency clocking.

3.4 Low power allocation and binding

Raghunathan et al. [111] proposed an allocation and binding method for low power that attempts to reduce both the capacitance and switching activity. Capacitance is reduced during allocation by trying to minimise the number of functional modules, registers and multiplexers. Binding reduces power consumption by assigning operations to modules and variables to registers such that the switching activity is reduced. Register and module binding is performed simultaneously, while also aiming to minimise the amount of interconnect needed. The algorithm is based on a weighted graph called the compatibility graph (CG). Initially, each variable and operation corresponds to a node in the CG, with undirected edges connecting compatible pairs. Weights are assigned to edges in the CG to indicate the preference of two variables (or operations) for sharing the same resource. Weights are determined according to a function that combines the values of capacitance and switching activity.

Algorithms that solve only the low power register binding problem have also been developed. For example, Chang et al. [12] presented an algorithm that calculated the switching activity of a set of registers shared by different data values. Switching activity is calculated based on the probability distributions of the input data streams. After calculating the switching activity between pairs of values that could share the same register and knowing the registers number, the low power register binding problem is formulated as a minimum cost clique covering of a compatibility graph. The problem is then solved optimally using a max-cost flow algorithm. In [11], the same authors investigated the problem of minimizing the total power consumption during the binding of operations to functional units in a scheduled datapath with functional pipelining and conditional branching. The authors formulated the power optimisation problem as a max-cost multi-commodity flow problem that is solved optimally. Although [12] and [11] provided optimal solutions, their application is limited to small sized problems. This was addressed by Choi et al. [147], who developed a new heuristic algorithm that is applicable to practical designs while producing near optimal results. The proposed algorithm determines a feasible binding solution by partially utilising the computation steps for finding a maximum flow of minimum cost in a network and then refines it iteratively.

Some algorithms have also been proposed to perform low power module binding. For example, Shiue et al. [125] described a linear programming (LP) based algorithm that targets power consumption optimisation during resource binding by reducing the switching activity at the input of the functional units. The algorithm consists of creating a multistage graph with m stages (corresponding to m cycles in the schedule) and n nodes per stage (corresponding to n functional units of the same type). Each edge in the graph has a cost according to the switching activity caused if two nodes were mapped to the same functional unit. The low power resource binding problem is then solved by finding n disjoint paths such that the total cost of these paths is minimal. Although this algorithm reduces efficiently the power consumption, it may be very time consuming when applied to large designs. A faster binding algorithm that takes into account candidate pairs of operations between two consecutive control steps and maximises the resource sharing was developed by Liu et al. [149].

To further reduce power consumption, voltage assignment problem has included into the module binding task. For example, in [23], Chen et al. proposed a low power binding algorithm for dual V_{dd} designs that minimises switching activity and maximises the number of low V_{dd} operations under the assumption that voltages could be dynamically configured at run time for each functional unit. This algorithm may results in higher power savings but introduces extra area costs due to complex control logic and a full chip dual rail power supply system. This disadvantage was addressed by the same authors in [22], targeting architectures where the voltages of functional units are fixed during run time.

In [94], Musoll et al. presented a binding algorithm that reduces the spurious switching activity by using transparent latches at the input ports of the functional units. However these latches incur area overhead and may reduce the power savings due to their power dissipation. These disadvantages were overcome in the low power register binding algorithm proposed by Luo et al. [46], where spurious switching activity of a given scheduled behaviour and functional unit binding is reduced by using retentive multiplexers. Retentive multiplexers can preserve their previous select signal values in the control steps where the select signals are don't cares.

3.5 Combined scheduling and binding for low power

The work presented in the previous sections focus only at power reduction in a single behavioural task, i.e. scheduling or binding. However, research has also been carried out aiming at power optimisation when scheduling and binding are performed simultaneously. For example, Tang et al. [144] presented an integer linear programming (ILP) based approach for power optimisation that considers concurrently module selection, scheduling and binding. The objective function minimises switching energy consumption of all datapath components under time and resource constraints. Chabini et al. [7] proposed another ILP formulation that not only considers scheduling and binding but also retiming while reducing power consumption due to switching activities. Recently, Zhao et al. [148] considered multicycling and multiple supply voltages in an ILP based approach that performs scheduling and binding with the aim of minimising energy or peak power in a design. Algorithms with less computational complexity than ILP formulations for simultaneous scheduling and binding have also been developed. For example, Katkoori et al. [51] presented PDSS, a behavioural synthesis approach that minimises power consumption by reducing the switching activity in the design. This approach uses a profiler tool that simulates the DFG using data streams specified by the user, hence collecting profile data for various operations and variables during the simulation time. Using this profile data and the switching activity data of all modules in the library, power consumption is estimated for each feasible solution (obtained after scheduling and binding) that meets the user constraints. Then, the solution with the lowest power consumption is chosen and a clique partitioning algorithm is applied for register optimisation. Interconnect optimisation and controller generation are also included in this approach.

San Martin et al. [122] presented Power-Profiler, an approach based on a genetic algorithm that reduces average and peak power consumption during behavioural synthesis of ASICs. Power consumption is reduced by disabling the clock of modules when they are idle, using multiple supply voltages, increasing parallelism and using fast and power hungry operators only in the critical path. Power-profiler performs simultaneously scheduling and module binding and searches for the best combination of library modules that minimises the power consumption of the design. Although this approach obtains good solutions in terms of power, it does not consider tasks

such as clock selection and retiming that are directly related to the main behavioural synthesis tasks. This was addressed by Raghunathan et al. [110], who developed SCALP, an iterative improvement algorithm for low power behavioural synthesis that performs simultaneously scheduling, binding, retiming and functional pipelining, clock selection and module selection. The effect of these tasks on both supply voltage and switching capacitance is also considered simultaneously. SCALP identifies a set of candidate supply voltages that may lead to the lowest power datapath and then examines possible values of clock period for each supply voltage. For each combination of supply voltage and clock period, the iterative improvement synthesis phase is executed with the aim of obtaining the minimum switching capacitance datapath that meets the time constraint at the current voltage. The iterative improvement algorithm explores the solution space by applying two types of modifications: module selection with rescheduling and hardware sharing/splitting with rescheduling. SCALP reduces the power consumption in data dominated designs but can not handle control dominated designs. This was overcome by Williams et al. [141], who presented a system called MOODS that does not impose any restriction on the system architecture, i.e. supports the full spectrum of designs from data to control dominated. Further investigation on low power behavioural synthesis targeting data and control dominated applications was carried out by Ranganathan et al. [116]. Here, the authors presented CHESS, a low power behavioural synthesis approach that targets both data and control dominated applications. Scheduling is performed using an algorithm based on tabu search [4] that minimises the resource requirements under given time constraints. To perform the binding task, the authors extended the algorithm proposed by Murugavel et al. [93] to consider control constructs. Power reduction is due to functional unit sharing, which attempts to assign the same functional unit to operations with at least one common input, decreasing the number of changing inputs.

The problem of low power behavioural synthesis has also been formulated using genetic algorithms. For example, Elgamel et al. [25] proposed a genetic algorithm based approach that integrates scheduling, allocation and binding with multiple voltage assignment with the aim of minimising average and peak power. In [95], Muthumala et al. combined a local search for rescheduling with a genetic algorithm to determine appropriate supply and threshold voltage for the DFG operations with

the aim of minimising the leakage and dynamic energy in the functional units. Binding for interconnection simplification is performed simultaneously with scheduling and module selection to reduce the interconnection energy. Interconnection simplification is achieved by increasing the sharing of interconnections among functional units. Other approach that reduces dynamic and leakage energy was developed by Jianfeng et al. [45]. Here, dynamic and leakage power consumption are reduced by assigning lower supply voltage V_{dd} or higher threshold voltage V_{th} to a candidate operation. The candidate operation is defined as the operation with the highest power-savings in the DFG. Power-savings are calculated based on a priority function that includes three factors: operation power difference, operation delay difference and operation mobility. The candidate operation is scheduled using a modified list scheduling algorithm and a new binding is performed. Average power savings of 36% using three V_{dd} and three V_{th} are reported when compared to a single supply voltage level.

3.6 Concluding Remarks

The literature review has shown that there has been considerable work reported on how to minimise power in the context of behavioural synthesis. From this review, it has been identified that an efficient technique to reduce power consumption is the use of multiple supply voltages (MSV). However, MSV presents area/power/delay overhead due to routing of the supplies and the use of level shifters to transfer data between functional units operating at different voltages. The possibility of using single supply voltage during behavioural synthesis overcomes the area/power/delay overhead of MSV and may result in comparable power savings. This aspect underpins the development of the reported algorithms in the next chapters.

Chapter 4

Power-Aware TIme Constrained Scheduling (PATICS)

4.1 Introduction

In Chapter 2, it was stated that reducing the voltage has a higher impact on power consumption than reducing the switching activity, capacitance or frequency. However, voltage reduction can be limited (hence power savings) if the schedule slack is not fully exploited due to non-uniform path lengths, a fixed clock period and a fixed number of control steps [47]. To avoid scheduling with a fixed clock period (fixed number of control steps), it is necessary to perform appropriate clock selection such that the utilization of the available schedule slack when using single supply voltage can be improved. The aim of this chapter is to describe a new time constrained scheduling algorithm [99] that takes into account the influence of the combined selection of clock and operations throughput on the quality of the schedules in terms of power and area. Section 4.2 introduces the terms and concepts used throughout this chapter. Section 4.3 highlights the influence of an appropriate clock and operations throughput selection on power consumption by means of an illustrative example. The Power-Aware TIme Constrained Scheduling (PATICS) algorithm is described in Section 4.4 with the help of some examples. PATICS selects the clock period and operations throughput such that power consumption can be reduced by scaling the voltage until the slack of at least one of the operations is zero. Section 4.5 presents extensive experimental results on benchmarks including differential equation solver, elliptical wave filter and discrete cosine transform, to demonstrate the efficiency of the algorithm. Finally, the concluding remarks of this chapter are given in Section 4.6.

4.2 Preliminaries

The proposed algorithm targets data dominated designs, as they are common in the digital signal and image processing areas [8]. As mentioned in Chapter 2 Section 2.2, data dominated applications consist mainly of arithmetic operations such as additions, subtractions, multiplications and divisions. For example, the differential equation (DIFFEQ) benchmark [102] is a data dominated design whose DFG contains 6 multiplications, 1 adder, 3 subtractors and 1 comparator, as shown in Figure 4.1b.

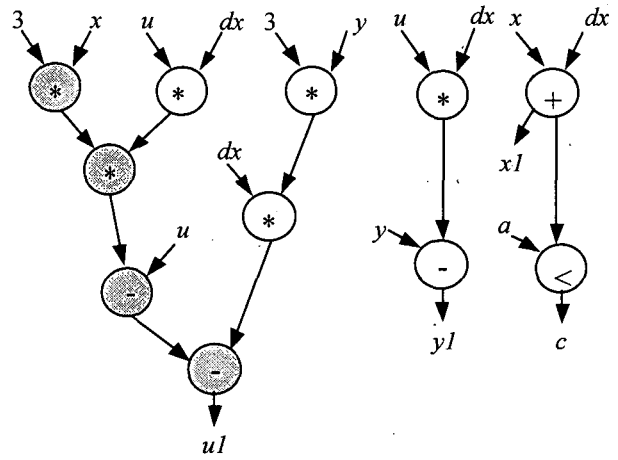
$x1 = x + dx$

$u1 = u - (3*x*u*dx) - (3*y*dx)$

$y1 = y + (u*dx)$

$c = x1 < a$

a) difference equations



b) DFG

Figure 4.1 DIFFEQ benchmark

An important characteristic of the data dominated applications is that their inputs arrive at a fixed rate, constraining the input sampling period. If the input samples are processed faster than the required rate, it is possible to reduce power consumption using supply voltage scaling [110]. The possibility of scaling the voltage is clearly determined by the schedule slack, which is the difference between the critical path and the sample period constraint. The critical path determines the time that takes each input sample to be processed, whereas the sample period constraint (henceforth referred to as time constraint T) is defined by the user. For example, in Figure 4.1, the shaded operations show the critical path, which can be calculated by adding the delay of 2 multiplications and 2 subtractions. Assuming that these operations are performed by the operators from Table 4.1 at 1.8V, the critical path takes 89.6ns to be executed. If a time constraint of 100ns is set by the user, a schedule slack of 10.4ns is obtained.

As shown in this simple example, the time required for the datapath to execute the critical path depends on the delay of the operations. This delay is considered as a typical register-to-register transfer that includes reading the operands from the registers, performing an operation on the operands and storing the results in another register [96]. Hence, the delay of an operation d_{op} can be calculated as [9]:

$$d_{op} = delay_{FU} + delay_{reg} + 2delay_{mux} \quad \textbf{(4.1)}$$

where $delay_{FU}$ is the delay of the functional unit, $delay_{reg}$ is the delay of the register and $delay_{mux}$ is the delay of the multiplexer.

The operation delay together with the clock period, determine the number of csteps required to execute such operation. In this thesis, the number of csteps required to execute an operation is refered as throughput and can be calculated by:

$$TP_{op} = \lceil d_{op} / Tclk \rceil \quad \textbf{(4.2)}$$

where TP_{op} is the throughput, d_{op} is the operation delay and $Tclk$ is the clock period. A throughput of 1 cstep means that the operation is single cycled, whereas a throughput greater than 1 cstep means that the operation is multicycled. For example, in Figure 4.2, operation N1 has a throughput of 1 cstep (single cycled operation), whereas operations N2, N3 and N4 have a throughput of 3 csteps (multicycled operations).

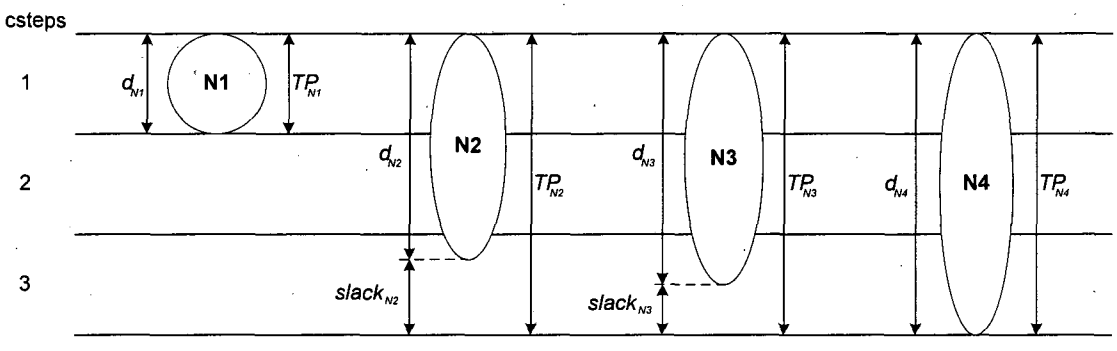


Figure 4.2 Operations with different throughputs and delays.

Although N2, N3 and N4 have the same throughput, i.e. $TP_{N2} = TP_{N3} = TP_{N4} = 3$, they present different delays, i.e. $d_{N4} > d_{N3} > d_{N2}$, which derive in some operation slack as in the case of N2 and N3. This operation slack is measured with respect to the throughput and can be calculated [2]:

$$slack_{op} = (TP_{op} * Tclk) - d_{op} \quad \textbf{(4.3)}$$

Assuming that each cstep in Figure 4.2 corresponds to an interval of length equal to $Tclk$ = 10ns, and operation delays d_{N2} = 22.5ns and d_{N3} = 25ns, operations slacks of 7.5ns and 5ns are obtained respectively for $slack_{N2}$ and $slack_{N3}$. Note that operations N1 and N4 have an operation slack equal to zero. The presence of operation slacks may lead to larger operations throughputs that can avoid meeting the design time constraint for some values of supply voltage, although the critical path can be executed in less or equal time than the set time constraint. This is better illustrated in the next section by means of an example.

4.3 Importance of clock and operations throughput selection

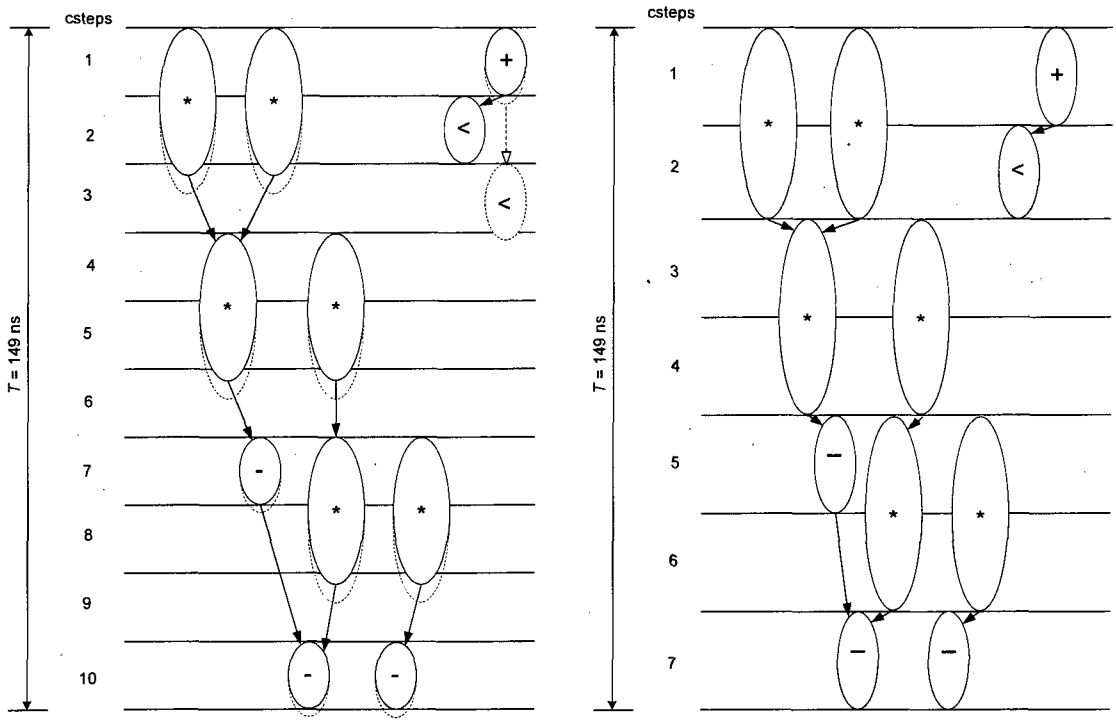
To demonstrate how the choice of the clock period and operations throughput affects the power consumption and functional resources requirements in behavioural synthesis, the following example is given. For the sake of explanation, consider throughout this section that a time constraint T of 149ns (10 times the minimum delay of the library shown in Table 4.1, the delay of an adder at 1.8V) has been set for the DIFFEQ benchmark, shown in Figure 4.1. In Table 4.1, the multiplier, adder, subtractor and comparator are symbolised by *, +, - and <, respectively. Throughout this chapter, it is considered that the addition, subtraction and comparison can be executed by the same functional unit. Hence the terms addition, subtraction and comparison are used indistinctively. The reported dynamic power and delay values of the library components are for 0.18μm technology [106]. It is assumed that the delay of the functional units (FUs) in Table 4.1 include estimates for register and multiplexer delays.

Table 4.1 0.18μm library component

*		
V (V)	$Pdyn$ (mW)	D (ns)
0.9	0.105	59.7
1.8	0.42	29.9
+, -, <		
V (V)	$Pdyn$ (mW)	D (ns)
0.9	0.008	29.8
1.8	0.031	14.9

As in [126], it is considered that the clock period *Tclk* is determined by the delay of the fastest functional unit when operating at maximum supply voltage. This allows the delays of the remainder FUs to be specified in multiples of the clock cycle. According to the functional unit library in Table 4.1, *Tclk* takes the value of 14.9ns and using the given time constraint T = 149ns, a schedule length (*Ls*) of 10 control steps (csteps) is obtained using equation (2.2) from Chapter 2, Section 2.2. With the clock period defined previously and the delays taken from the library shown in Table 4.1, it is possible to compute the throughput of the operations using equation (4.2). Hence, the multiplier of the library has a throughput of 3 and 5 csteps at 1.8V and 0.9V, respectively; whereas for the same voltages, the adder needs 1 and 2 csteps to complete an operation. Assuming that the FUs are operating with a maximum supply voltage of 1.8V, and using the schedule length obtained previously as an input to the scheduler described in Appendix 4, the schedule of Figure 4.3a was obtained. The scheduler used aims to determine the minimum number of hardware components required to perform the DFG operations when a time constraint is given. All the schedules in this section were obtained using such scheduler.

From Figure 4.3a, it can be seen that the functional resources requirement consist of 2* with throughput of 3 csteps and 2+ (assuming that additions and subtractions are executed by the same functional unit) with throughput of 1 cstep. The estimated power consumption for this schedule is 521μW and the critical path could be executed in 89.6ns, leaving a schedule slack of 59.4ns. Note that since the clock period *Tclk* is equal to the delay of adder, the slack of the addition is zero, whereas the slack of the multiplication is 14.8ns. Power dissipation values reported in this motivational example are calculated using the values from Table 4.1 and equation (4.5). The design parameters of the schedule from Figure 4.3a are summarized in the first row of Table 4.2, denoted as sch1. In the table header, *Ls* represents the length of the schedule, TP_M the throughput of the multiplier, TP_A the throughput of the adder and $slack_{sch}$ the schedule slack. The slack of the multiplication is $slack_*$, $slack_+$ is the slack of the addition, V the voltage, P the power and *#FUs* the number of functional units.



a) sch1 b) sch2

Figure 4.3 DIFFEQ schedules with different power-area tradeoffs

Table 4.2. Schedule characteristics using different power reduction techniques

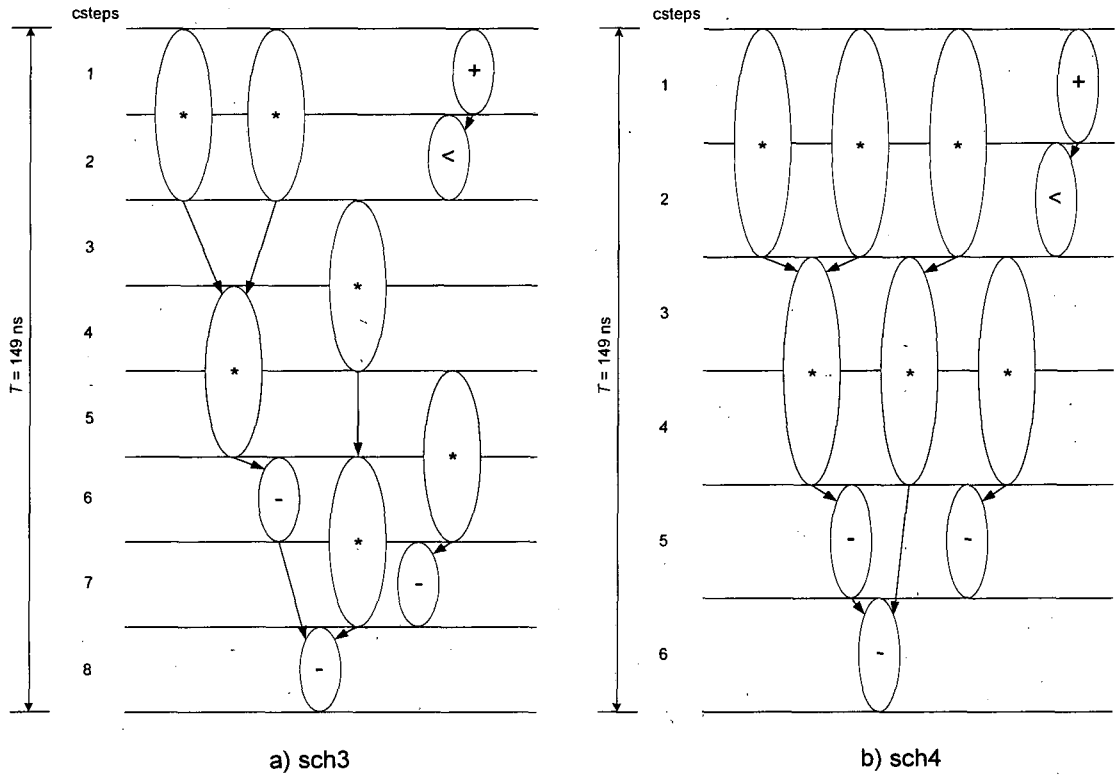
	Tclk (ns)	*Ls* (csteps)	TP_M (csteps)	TP_A (csteps)	$slack_{sch}$ (ns)	$slack_*$ (ns)	$slack_+$ (ns)	*V* (V)	*P* (μW)	*#FUs*
sch1	14.9	10	3	1	59.4	14.8	0	1.8	521	2* 2+
sch2	21.3	7	2	1	21.3	0	0	1.41	458	2* 2+
sch3	18.6	8	2	1	37.3	0	0	1.57	497	2* 1+
sch4	24.8	6	2	1	0	0	0	1.20	385	3* 2+

Now, consider taking advantage of the schedule slack, i.e. 59.4ns, through voltage scaling, hence reducing the power consumption. The application of voltage scaling would increase the delay of the operations as represented by the dotted lines in Figure 4.3a. This in turn would increase the addition throughput from 1 cstep to 2 csteps, violating the time constraint as can be seen in cstep 10. Thus, an inappropriate choice of the clock period may restrain the exploitation of the schedule slack, inhibiting the application of greater supply voltage scaling, which leads to higher power consumption solutions. However, it is possible to take better advantage of the schedule slack and decrease significantly the power consumption through voltage reduction, by not constraining the clock period to the fastest functional unit delay, as in the case of sch1, but through appropriate choice of clock period and operations throughput. For example, consider now the time constraint divided in 7 csteps (*Tclk*

= 21.3ns) and multiplier and adder throughputs of 2 csteps and 1 cstep respectively, see sch2 in Table 4.2. It can be seen that with this new *Tclk* and operation throughputs, the schedule slack has been reduced from 59.4ns to 21.3ns, and the multiplication and addition slack has decreased to 0ns. The supply voltage has experienced a reduction of 0.39V, which allows lower power consumption than in sch1 (from 521μW to 458μW). Note that despite this power decrease, the number of used FUs remains the same that in sch1, as shown in the schedule of Figure 4.3b.

Although sch2 provides a solution where power consumption has been effectively reduced, often designers need guidelines about the possible power-area tradeoffs for a given design. By carefully choosing the operations throughput and employing clock selection, the design space can be efficiently explored, obtaining different power-area tradeoffs, which is the main motivation of the proposed algorithm. For example, consider sch3 in Table 4.2, where the time constraint has been divided into 8 csteps (*Tclk* = 18.6ns), and the multiplication and addition have throughputs of 2 csteps and 1 cstep respectively. For the above settings, the minimum voltage required to meet the time constraint is 1.57V, which leads to a higher power consumption with respect to sch2 (from 458μW to 497μW). However, the number of functional units is reduced by one adder as shown in the schedule for this solution in Figure 4.4a. It can also be seen that although the schedule slack is increased to 37.3ns, the multiplication and addition slack remains the same, i.e. 0ns.

For the same time constraint, i.e. 149ns, further power reduction can be obtained by assigning different values to the operations throughput and the clock period, and by determining the corresponding voltage, as illustrated for sch4 in Table 4.2. Although sch4 presents lower power consumption (385μW) than sch3, the functional resources requirement is increased by one multiplier and one adder as shown in the schedule of Figure 4.4b. This solution presents the lowest operating voltage possible for this schedule, i.e. 1.20V, reducing the schedule slack to 0ns. Notice that the time constraint in Figure 4.3 and Figure 4.4 has remained the same (149ns) and that only the schedule length has changed according to the selected clock period.



a) sch3 b) sch4

Figure 4.4 DIFFEQ schedules with different power-area tradeoffs

4.4 Power-aware time constrained scheduling algorithm

Section 4.3 has demonstrated the influence of clock and operations throughput selection on power dissipation during the scheduling task. This section presents an algorithm capable of identifying the appropriate clock period, operations throughput and the scaled supply voltage such that solutions with different resource requirements and reduced power can be obtained within a specified time constraint. The key idea of the proposed algorithm is taking a better advantage of the operation slack by scaling the supply voltage (hence reducing power consumption) until the delay of at least one of the operations in the design fits exactly into its throughput. This will be explained in more detail when describing the algorithm later in this section. The inputs of the algorithm are: a time constraint, a maximum clock frequency and a design DFG. The algorithm also requires the specification of a library component characterised for power and delay, e.g. Table 4.1. The outputs of the algorithm are a set of solutions that consist of schedules with different power-area tradeoffs, from which the designer can choose the solution that best suits his/her needs.

The proposed power-aware time constrained scheduling algorithm (PATICS) is shown in Listing 4.1. The terminology adopted during the explanation of the algorithm is the following: T is the time constraint in ns, *max_f* is the maximum clock frequency, *Tclk* is the clock period in ns, *Ls* is the schedule length, V is the supply voltage and *listop* is a list containing the operations ordered according to their power contribution. The operation delay is represented as D_{op}, TP_{op} is the operation throughput, $maxD_{op}$ is the maximum operation delay (taken from the library given in Table 4.1) and *op* can be of category *power_op* or *remain_op*.

To better illustrate the function of the algorithm, consider again DIFFEQ benchmark (Figure 4.1b) with a time constraint of 149ns and a maximum clock frequency *max_f* of 500MHz. These values have been arbitrarily set for illustration purposes. The algorithm starts calculating the total power of the design at maximum voltage *max_V* (line 1) using equation (4.5) and the values from the library component. For the DIFFEQ example, the algorithm calculates a total power of 521μW at maximum voltage 1.8V. The contribution of multipliers and adders to the total power dissipation is 506μW and 15μW respectively. Then, the category *power_op* is assigned to the type of operation which contributes the most to the total power and the category *remain_op* is assigned to the remaining operations (line 2). In this case, the category *power_op* is assigned to the multiplication, whereas the additions are assigned to category *remain_op*. Using the power contribution information defined before, the operations are ordered in terms of maximum power consumption in the list *listop* (line 3). This list is used by the scheduler described in Appendix 4 so that power consumption is considered when ordering the operations to be scheduled. The operations are then ordered in the list *listop* as follows: multiplications first and then additions. Now, *Ls* is initialised (line 4) with a schedule length equal to the csteps of the critical path, which was calculated with the ASAP scheduling algorithm assuming all the operations single cycled. For the DIFFEQ example, *Ls* = 4 csteps since the critical path has four operations, as shown in Figure 4.1b. With this value of *Ls* and the time constraint T, the clock period *Tclk* can be computed (line 5) as explained in Chapter 2, Section 2.2.4, using equation (2.2). The clock period *Tclk* has a value of 37.25ns in this example. The minimum clock period *min_Tclk* that indicates to the algorithm when to stop is calculated using the maximum frequency *max_f* (line 6), i.e. *min_Tclk* = 2ns for the current case.

Listing 4.1 Pseudocode of the power-aware time constrained scheduling [99]

```
1 calculate total power at max_V
2 identify power_op and remain_op
3 sort operations in listop
4 initialise Ls = critical_path_csteps = ASAP(1, 1)
5 calculate Tclk
6 determine min_Tclk = 1/max_f
7 while Tclk ≥ min_Tclk do
8       calculate TP_power_op (D_power_op(max_V))
9       do
10              increase D_power_op
11              if D_power_op ≤ maxD_power_op do
12                      calculate V
13                      calculate D_remain_op(V)
14                      calculate TP_remain_op (D_remain_op)
15                      critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
16                      if Ls ≥ critical_path_csteps do
17                              schedule (Ls, TP_power_op, TP_remain_op, listop)
18                              calculate total_power
19                              calculate total_area
20                      end if
21                      calculate TP_power_op (D_power_op(min_V))
22                      calculate TP_remain_op (D_remain_op(min_V))
23                      critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
24                      if Ls ≥ critical_path_csteps do
25                              schedule (Ls, TP_power_op, TP_remain_op, listop)
26                              calculate total_power
27                              calculate total_area
28                      else do
29                              while (Ls < critical_path_csteps) do
30                                      decrease TP_power_op
31                                      calculate D_power_op
32                                      calculate V
33                                      calculate D_remain_op
34                                      calculate TP_remain_op (D_remain_op)
35                                      critical_path_csteps = ASAP(TP_power_op, TP_remain_op)
36                                      if Ls ≥ critical_path_csteps do
37                                              schedule (Ls, TP_power_op, TP_remain_op, listop)
38                                              calculate total_power
39                                              calculate total_area
40                                      end if
41                              end while
42                      end if
43              end if
44      while ( D_power_op ≤ maxD_power_op)
45      increase Ls
46      calculate Tclk
47 end while
```

Next step is to evaluate the throughput TP_{power_op} (line 8) as shown in equation (4.2), using D_{power_op} calculated at the maximum voltage in the library *max_V*. For the DIFFEQ example, the delay $D_{power\_op}(max\_V)$ corresponds to the delay of the multiplier from Table 4.1 at 1.8V, i.e. 29.9ns, and the throughput TP_{power_op} takes a value of one cstep, as shown in Figure 4.5a. The delay D_{power_op} is then increased

(line 10) with the aim to meet exactly the same number of csteps of the throughput TP_{power_op}. An increase of the delay D_{power_op} allows a voltage reduction that has a higher impact on the power consumption of operations that are more power hungry, leading to a decrease in the total power dissipated by the schedule. The increased delay D_{power_op} is denoted in Figure 4.5a with the dotted lines, where the new value of D_{power_op} is 37.25ns (calculated with equation (4.2)). The increased delay D_{power_op} is then compared with the maximum delay allowed $maxD_{power\_op}$ (line 11). In case D_{power_op} is greater than $maxD_{power\_op}$, the algorithm increases the schedule length (line 45), calculates a new clock period (line 46) and attempts to select new operations throughputs. In case D_{power_op} is lower or equal than $maxD_{power\_op}$, the supply voltage V is computed (line 12), i.e. 1.57V for the DIFFEQ case. The delay of the remaining operations D_{remain_op} (line 13) is then evaluated at the supply voltage V, and the throughput TP_{remain_op} (line 14) is computed.

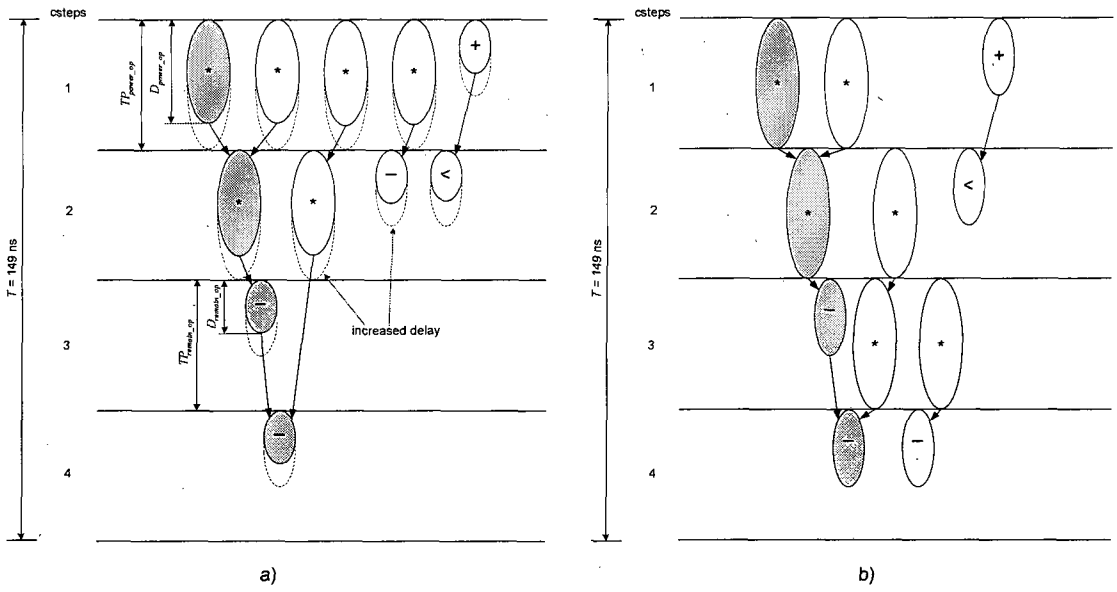


Figure 4.5 a) ASAP schedule, b) time constrained schedule, for DIFFEQ with 4 csteps

According to [47], the propagation delay of CMOS is approximately proportional to $V/(V-V_{th})^2$, where V_{th} is the threshold voltage. This approximation suggests that the delay is inversely proportional to the supply voltage V. Consequently, the relationship between delay and voltage can be modelled using a 1st order Lagrange interpolating polynomial [6]:

$$D(V) = \sum_{k=0}^{1} d(V_k) L_k(V)$$
$$L_0(V) = \frac{(V - V_1)}{(V_0 - V_1)} \quad (4.4)$$
$$L_1(V) = \frac{(V - V_0)}{(V_1 - V_0)}$$

where V is the supply voltage, V_o and V_1 are the minimum and maximum voltage available in the library, i.e. 0.9V and 1.8V according to Table 4.1, $d(V_0)$ is the delay at voltage V_0, $d(V_1)$ is the delay at voltage V_1, and $D(V)$ is the delay at voltage V. Note that using Lagrange or any interpolating polynomial leads to similar results since only two points are considered for interpolation, i.e. (V_0, $d(V_0)$) and (V_1, $d(V_1)$). Using equation (4.4) and the supply voltage V obtained previously, i.e. 1.57V, operation delays of 18.66ns have been obtained for the addition, subtraction and comparison for this example. The throughputs for the addition, subtraction and comparison are one cstep, as shown in Figure 4.5a.

Now, having TP_{power_op} and TP_{remain_op}, the critical path can be computed (line 15). Then, the feasibility of using such throughputs without violating the schedule length Ls is verified (line 16). In this case, the critical path composed by the shaded operations in Figure 4.5a can be executed in the same number of csteps that the schedule length, so there is no violation of the schedule length. If the schedule length is not violated, the DFG operations are then scheduled (line 17) using the modified scheduler with the current schedule length Ls and operations throughputs TP_{power_op} and TP_{remain_op}. Figure 4.5b shows the schedule for the DIFFEQ example with $Ls = 4$, $TP_{power\_op} = 1$ and $TP_{remain\_op} = 1$. Now, the total power consumption *total_power* of the design is evaluated (line 18) as in [122]:

$$P(V) = \frac{\sum_{AllFUs} N_{FU} \cdot P_{FU}(V) \cdot D_{FU}(V)}{T} \quad (4.5)$$

where $P(V)$, $D_{FU}(V)$ and $P_{FU}(V)$ are respectively the design power consumption, the functional unit (FU) delay and FU power, all at voltage V. N_{FU} is the number of times each type of FU is used and T is the time constraint. To calculate the delay $D_{FU}(V)$, equation (4.4) is used, i.e. $D_{FU}(V) = D(V)$. To compute the power $P_{FU}(V)$, the quadratic dependency of power on voltage is modelled using a 2nd order Lagrange interpolating polynomial [6]:

$$
\begin{aligned}
P(V) &= \sum_{k=0}^{2} p(V_k) L_k(V) \\
L_0(V) &= \frac{(V - V_1)(V - V_2)}{(V_0 - V_1)(V_0 - V_2)} \\
L_1(V) &= \frac{(V - V_0)(V - V_2)}{(V_1 - V_0)(V_1 - V_2)} \\
L_2(V) &= \frac{(V - V_0)(V - V_1)}{(V_2 - V_0)(V_2 - V_1)}
\end{aligned}
\tag{4.6}
$$

where V is the supply voltage, V_o is 0V, V_1 is the minimum voltage of the library, i.e. 0.9V, V_2 is the maximum voltage of the library, i.e. 1.8V, $p(V_0)$ is 0mW, $p(V_1)$ is the power at voltage V_1, $p(V_2)$ is the power at voltage V_2, and $P(V)$ is the power at voltage V. The power $P_{FU}(V)$ is then calculated using equation (4.6), i.e. $P_{FU}(V) = P(V)$. In Listing 4.1, the area of the design *total_area* (line 19) is estimated as in [122], checking the maximum number of FU of each type at every cstep. Hence, the DIFFEQ example presents a power dissipation of 497μW and an area of 2 multipliers and 2 adders, as shown in Figure 4.5b.

So far, the algorithm has examined the solution only in the lower bound of the operation throughput TP_{power_op}. For every *Tclk*, the lower bound of the operation throughput may lead to a schedule with low resources requirement. However, the proposed algorithm not only targets area optimization, but also power optimization. For this reason, it is also necessary to analyse the upper bound of the operations throughput for a certain clock period. For every *Tclk*, the upper bound of the operation throughput leads to a schedule with low power dissipation. To better illustrate the impact of the operations throughput bounds on a schedule in terms of power and area, consider Figure 4.6, where a time constraint of 149ns is divided into 12 csteps resulting in *Tclk* = 12.4ns. Using this *Tclk* and the algorithm described in Listing 4.1, the operations throughput are $TP_{multiplier} = 3$ and $TP_{adder} = 2$ for the lower bound (see Figure 4.6a), and $TP_{multiplier} = 4$ and $TP_{adder} = 2$ for the upper bound (see Figure 4.6b). From Figure 4.6a it can be seen that using the lower bound of the operations throughput results in lower resource requirements. This is because smaller throughputs may allow operations to be better distributed along the schedule length, avoiding the overlapping of their execution times, hence, reducing the resource requirements. From Figure 4.6b it can be seen that larger operations throughput mean larger delays which result from a lower voltage, hence lower power consumption.

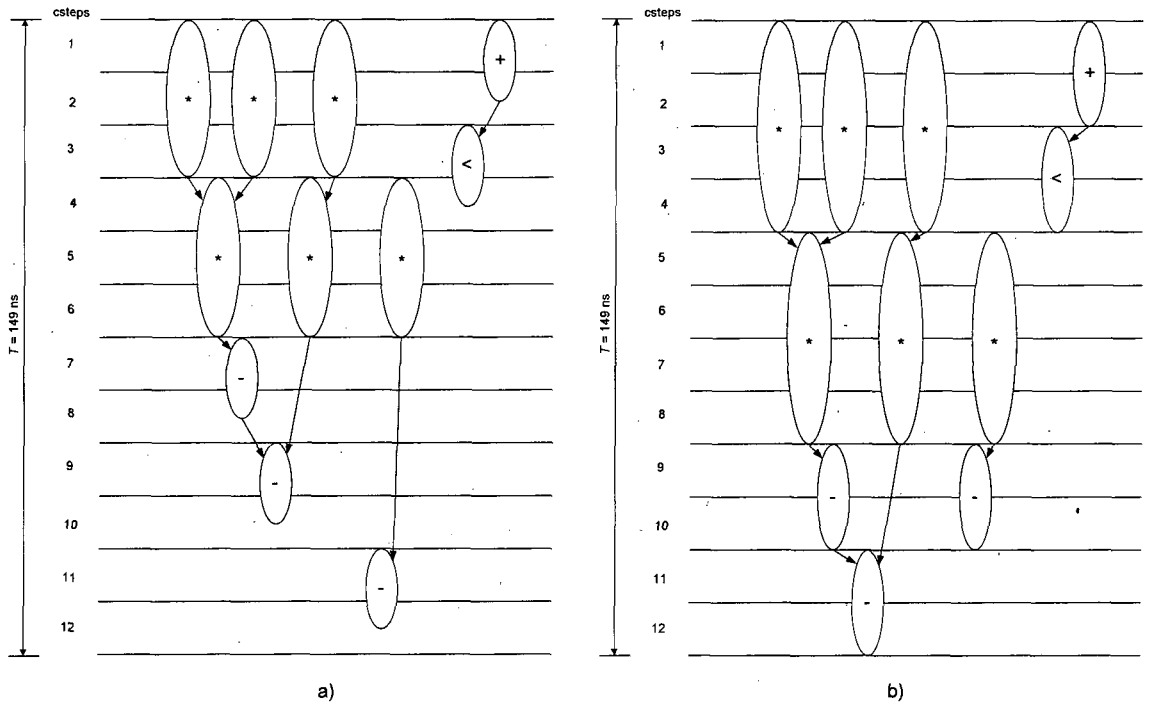


Figure 4.6 Schedules with two values of throughputs a) lower bound b) upper bound

In Listing 4.1, the upper bound of TP_{power_op} (line 21) is evaluated as shown in equation (4.2), using D_{power_op} calculated at the minimum voltage in the library *min_V*. For example, consider that the algorithm is analysing a solution for DIFFEQ benchmark with a schedule length *Ls* of 10 csteps and the upper bound of TP_{power_op}. The delay D_{power_op} corresponds then to the delay of the multiplier from Table 4.1 at 0.9V, i.e. 59.7ns, and the throughput TP_{power_op} takes a value of four csteps, as shown in Figure 4.7a. The throughput TP_{remain_op} (line 22) is computed based on the delay D_{remain_op}, which is also calculated at *min_V*. In this case the delay D_{remain_op} correspond to the delay of the adder from Table 4.1 at 0.9V, i.e. 29.8ns, leading the throughput TP_{remain_op} to take a value of two csteps, as shown in Figure 4.7a. With TP_{power_op} and TP_{remain_op} the critical path is calculated (line 23), and later used to verify if the schedule length *Ls* has been violated (line 24). If the schedule length is not violated, the DFG operations are then scheduled (line 25), and the power (line 26) and area (line 27) of the schedule are calculated as described previously. In the current case, the critical path can be executed in 12 csteps (see Figure 4.7a), which is larger than the schedule length of 10 csteps defined at the beginning of this illustrative example. In case the schedule length *Ls* is violated, the throughput is decreased TP_{power_op} (line 30) and the delay D_{power_op} is calculated (line 31) to obtain later the voltage *V* (line 32). Then the delay D_{remain_op} (line 33) and the throughput

TP_{remain_op} (line 34) are computed. For the DIFFEQ example with schedule length Ls = 10, the throughput TP_{power_op} is decreased to three csteps (as denoted in Figure 4.7 with the dotted lines) leading to a D_{power_op} of 44.7ns with a scaled voltage of 1.35V. With this voltage, a delay D_{remain_op} of 22.4ns is obtained, which leads to a throughput TP_{remain_op} of 2 csteps for all the remaining operations.

The critical path is again evaluated with these new values for the throughputs (line 35) and later used to verify that the schedule length Ls is not violated (line 36). If so, the operations are scheduled (line 37) and the power (line 38) and area (line 39) of the schedule are computed. In the current case, the critical path can be executed in 10 csteps, which is equal to the schedule length Ls. Hence, the schedule length is not violated and the operations are scheduled as shown in Figure 4.7b, requiring 3 multipliers and 2 adders to execute the design. The power dissipation of this schedule is 438μW. In case of violation of schedule length Ls, the procedure (lines 30-40) is repeated until obtaining feasible operations throughput for the schedule length Ls. The previous steps (lines 10-43) are repeated while D_{power_op} is less than, or equal to, $maxD_{power\_op}$. The next step is to increase the value of Ls for the design (line 45), calculate $Tclk$ (line 46), and repeat the external loop (lines 8-46) until the clock period $Tclk$ exceeds the inverse of the maximum clock frequency max_f.

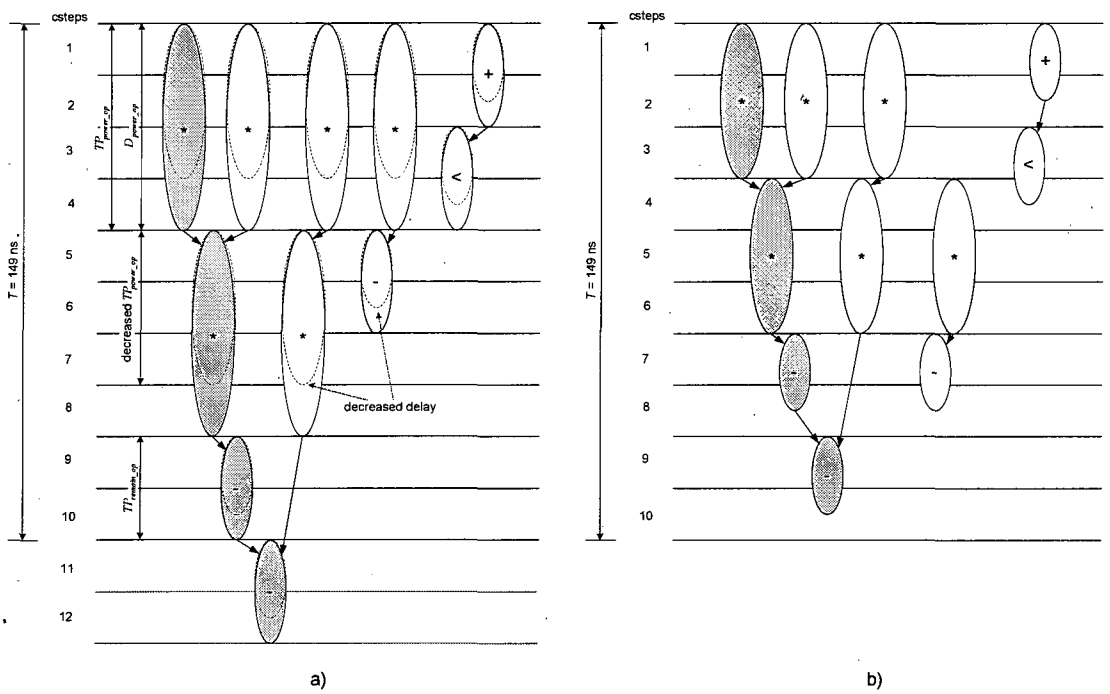


Figure 4.7 a) ASAP schedule, b) time constrained schedule, for DIFFEQ with 10 csteps

Note that the algorithm described in Listing 4.1 calculates a reduced voltage based on the delays of operations in category *power_op*. A voltage reduction has a higher impact in operations of category *power_op* than in *remain_op* if the power consumed by all the operations in *power_op* is higher than in *remain_op*, as it is the case of the benchmarks used in this thesis. If this is not the case, a reduced voltage should be derived from operations in *remain_op*. To do this, it would be necessary to increase the delay of all types of operations until it fits exactly into the number of csteps of the throughputs, and then calculate the respective voltages. The lowest of these voltages is chosen and then used to calculate the throughputs of all the operations before performing scheduling.

The key concepts of the algorithm described in Listing 4.1, i.e. clock and operations throughput selection, can not be applied to control dominated designs with data dependent loops. Data dependent loops make difficult to determine the execution time of the critical path in control dominated designs. If the execution time of the critical path is not known, the algorithm can not determine the schedule slack, and consequently how much voltage can be reduced. On the other hand, the clock and operations throughput selection algorithm described in Listing 4.1 can still be applied to data dominated designs with data independent loops, since the execution time of the critical path can be identified. Note that there is a significant difference in scheduling straight line code versus nested loop code due to the ability to overlap loop iterations in a schedule, which allows functional pipelining. In functional pipelining, the algorithm description is divided into sequences of operation stages that operate concurrently. Successive stages are streamed into the pipe so that different algorithm instances are executed in an overlapping fashion on a single data path [101]. For a given latency L, operations scheduled into cstep i in an instance and into csteps $i + kL$ ($k = 1, 2, 3 \ldots$) in the successive instance, run concurrently. Consequently, the task of scheduling a pipelined algorithm can be solved with the scheduler used in Listing 4.1 just by combining the distribution graphs of all the instances, and balancing them across all groups of concurrent csteps.

4.5 Experimental results

The proposed time constrained scheduling (TCS) algorithm that takes into account the impact of operations throughputs and clock period on the power-area tradeoffs in behavioural synthesis has been implemented in C++. Three experiments have been conducted on a Pentium 4, 2.2GHz, 512 MB RAM under different time constraints for the DIFFEQ, elliptical wave filter (EWF) and discrete cosine transform (DCT) benchmarks. Experiment 1 reports the power-area tradeoffs obtained when PATICS has been used. Experiment 2 provides a comparison between the power consumption values obtained with PATICS using a single supply voltage (SSV) and those obtained using a multiple supply voltage (MSV) algorithm. Experiment 3 compared the solutions obtained by PATICS and an area optimised scheduler that does not take power consumption into consideration [58]. Power consumption reported later in this section is based on equation (4.5) and Table 4.1, whereas the area requirement is estimated considering the maximum number of FU of each type at every cstep [122].

4.5.1 Power-area tradeoffs analysis

DIFFEQ

So far, the power-area tradeoffs of DIFFEQ for a single time constraint have been shown in Section 4.3. Figure 4.8 shows the power-area tradeoffs of the DIFFEQ for different time constraints. Two interesting results were observed. The first result is that for the same functional resources requirement, longer time constraints result in lower power consumption, as expected. For example, for 2* 2+, the power consumption is 545μW when the time constraint is 134ns, however it reduces to 306μW when the time constraint increases to 179ns. To illustrate the causes of this power reduction, consider Table 4.3, which shows some solutions parameters with a resource requirement of 2* 2+ for different time constraints. It can be seen that as the time constraint increases the schedule slack is larger allowing the use of lower operating voltages, thus reducing the average energy consumption (E) of the design. The energy consumption E is calculated as the product of power P and time T, $E = P \cdot T$. For 134ns, the schedule slack is 44.4ns allowing a scaled voltage of 1.54V, which leads to an energy of 73.0pJ. This is reduced to 54.8pJ for 179ns because the larger schedule slack, i.e. 89.4ns, allows a lower voltage application, i.e. 1.16V.

Hence, the power reduction is due not only to the increase of the time constraint, but also to a decrease in the average energy E.

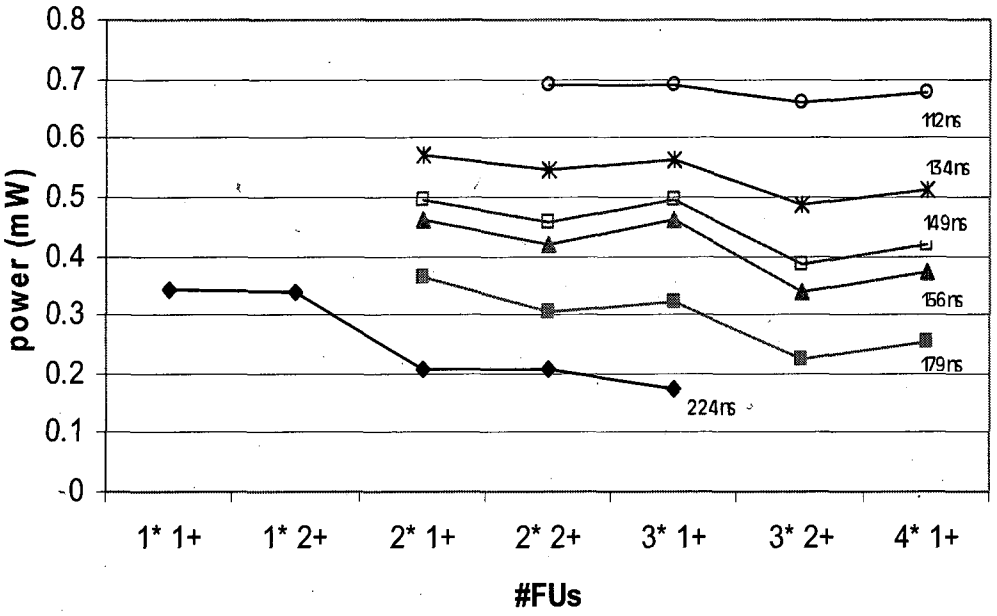


Figure 4.8 Power-area tradeoffs for DIFFEQ with different time constraints

Table 4.3 Voltage, power and energy consumption for DIFFEQ with 2* 2+

T (ns)	$slack_{sch}^{*}$ (ns)	V (V)	E (pJ)	P (μW)
112	22.4	1.73	77.1	688
134	44.4	1.54	73.0	545
149	59.4	1.41	68.2	458
156	66.4	1.35	65.1	417
179	89.4	1.16	54.8	306
224	134.4	1.01	45.9	205

* obtained considering that all the operations are executed at 1.8V

The second result that can be obtained from Figure 4.8 is that a greater number of FUs can lead either to lower or higher power consumption. For example, for a time constraint of 134ns, increasing the area from 2* 1+ to 2* 2+ allows reducing the power consumption from 570μW to 545μW. However, by changing the resources from 2* 2+ to 3* 1+ the power dissipation is increased from 545μW to 563μW. Hence, the use of more FUs does not necessarily lead to lower power consumption. Similar results can be seen for other time constraints such as 149ns, 156ns and 179ns.

To explain the increase or decrease in the power consumption when increasing the resources, consider Table 4.4, which provides an insight into the power area tradeoffs for 134ns. It can be seen that the power variations are due to the application of different operating voltages. For example, there is a voltage reduction from 1.68V to 1.54V when changing the resources from 2*1+ to 2*2+, and a voltage increase from 1.54V to 1.63V when changing the resources from 2* 2+ to 3* 1+. The different operating voltages are possible due to the combination of clock period and operations throughput, which also lead to different schedules.

Table 4.4 Characteristics of power-area tradeoffs for DIFFEQ, T = 134ns

	Ls (csteps)	*Tclk* (ns)	TP_M (csteps)	TP_A (csteps)	*V* (V)	*P* (μW)	#FUs
tradeoff1	8	16.8	2	1	1.68	570	2* 1+
tradeoff2	21	6.4	6	3	1.54	545	2* 2+
tradeoff3	19	7.0	5	3	1.63	563	3* 1+
tradeoff4	18	7.4	6	3	1.34	487	3* 2+

Figure 4.9 shows the schedules obtained for tradeoff1 and tradeoff2 from Table 4.4. It can be seen that the schedule length is smaller in Figure 4.9a than in Figure 4.9b, i.e. 8 csteps and 21 csteps respectively, however the time constraint is the same, i.e. 134ns. This is because the selected clock periods have values of 16.8ns and 6.4ns correspondingly. Note that although the multiplications are multicycled in both schedules, their throughputs are different, i.e. 2 csteps in Figure 4.9a and 6 csteps in Figure 4.9b. In the case of the additions, they are single cycled with TP_A = 1 in Figure 4.9a but multicycled with TP_A = 3 in Figure 4.9b. The combination of these operations throughputs with the schedule lengths of 8 csteps and 21 csteps results in the resource requirements shown in Table 4.4.

From Table 4.4 it can also be clearly noted that increasing the resource usage will not always lead to significant power reduction. For example, when comparing tradeoff1 and tradeoff3, a power saving of only 1.2% is obtained by adding one multiplier to the design; and hence a significant area increase. On the other hand, adding one adder (tradeoff3 and tradeoff4) leads to a power reduction of 13.5%.

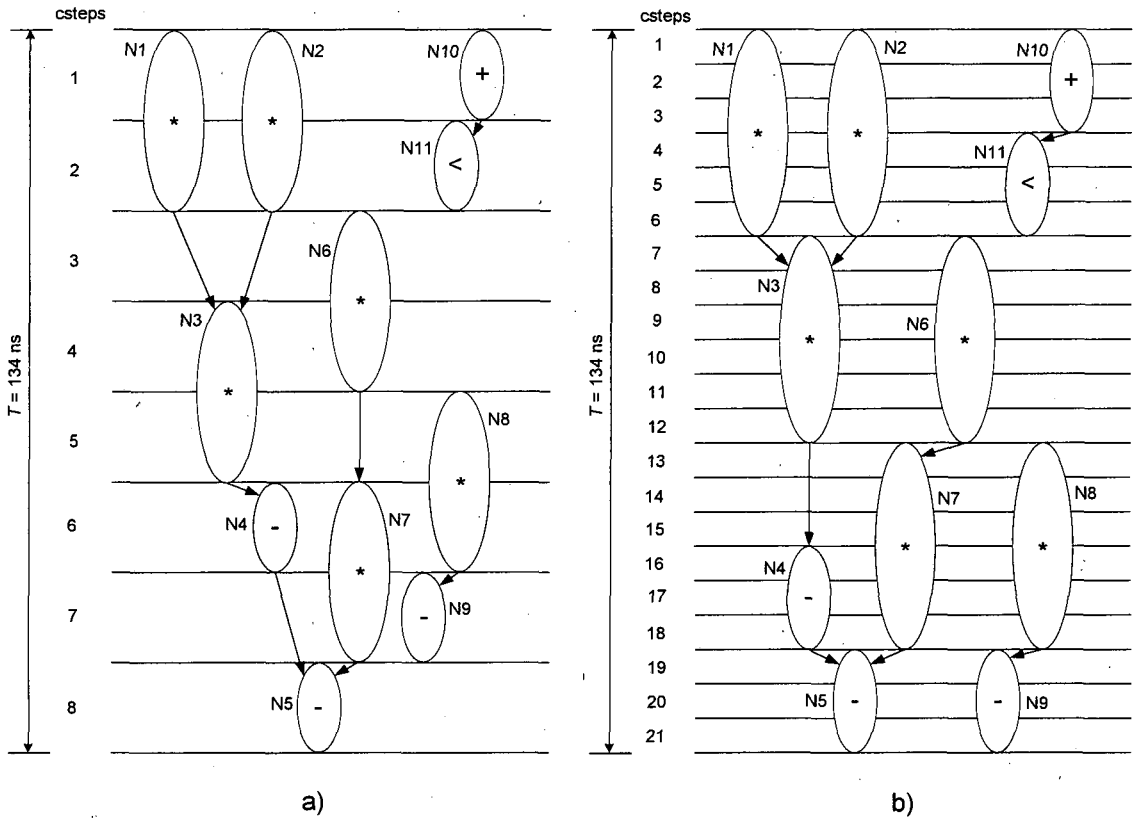


Figure 4.9 DIFFEQ schedules for , a) tradeoff1, b) tradeoff2

DCT

This benchmark is useful to investigate the capability of scheduling designs with massive data parallelism. The DCT data flow graph [58] consists on 25 additions, 7 subtractions and 16 multiplications. Figure 4.10 shows the power-area tradeoffs of the DCT for different time constraints. It can be seen that as in the case of DIFFEQ, longer time constraints lead to lower power consumption for the same resources. For example, for 6* 6+, the power consumption reduces from 1.6mW at 130ns to 0.4mW at 261ns. As explained previously, this power reduction is mainly due to the use of lower operating voltages that reduce the average energy consumption *E*, as shown in Table 4.5. Note that the voltage has been reduced from 1.77V at 130ns to 0.91V at 261ns, decreasing the energy from 214.2pJ to 109.4pJ.

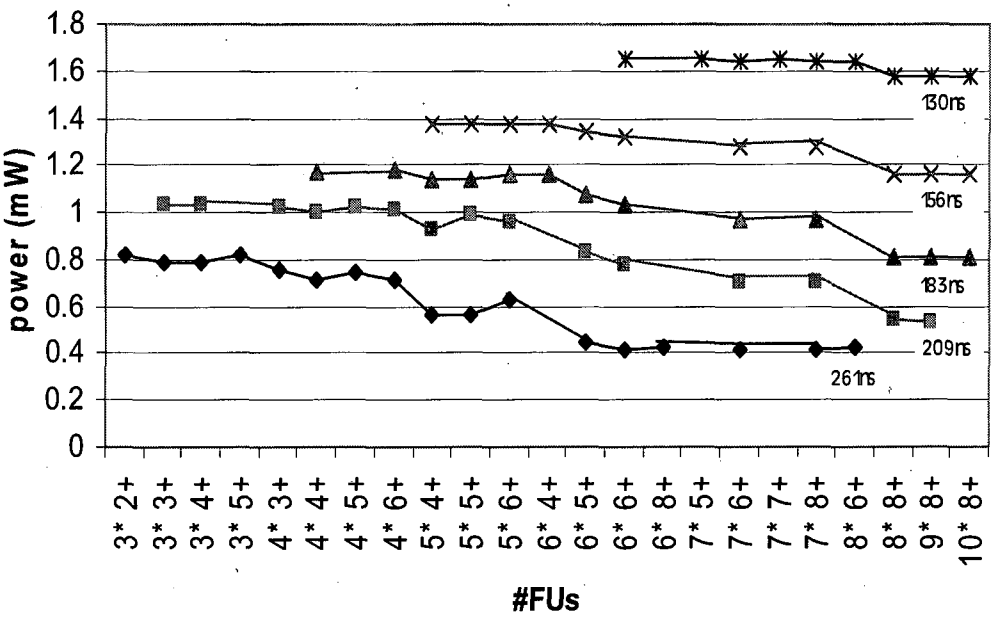


Figure 4.10 Power-area tradeoffs for DCT with different time constraints

Table 4.5 Voltage, power and energy consumption for DCT with 6* 6+

T (ns)	$slack_{sch}^{*}$ (ns)	V (V)	E (pJ)	P (mW)
130	25.6	1.77	214.2	1.6
156	51.6	1.59	206.1	1.3
183	78.6	1.40	188.7	1.0
209	104.6	1.22	162.2	0.8
261	156.6	0.91	109.4	0.4

* considering that all the operations are executed at 1.8V

From Figure 4.10, it can also be seen that it is not possible to establish a relation between area and power. For example, at 183ns, power increases (from 1.17mW to 1.18mW) when resources change from 4* 4+ to 4* 6+ and decreases (from 1.18mW to 1.14mW) when resources change from 4* 6+ to 5* 4+. As explained for DIFFEQ, these power variations are due to different operating voltages, as shown in Table 4.6 for 183ns. Note that schedules with 4* 4+, 4* 6+ and 5* 4+ operate with voltages of 1.70V, 1.78V and 1.59V respectively, and that as of 6* 5+, the voltage is reduced steadily as the resources increase, resulting in a power consumption decrease.

Table 4.6 Characteristics of power-area tradeoffs for DCT, T = 183ns

	Ls (csteps)	$Tclk$ (ns)	TP_M (csteps)	TP_A (csteps)	V (V)	P (mW)	$\#FUs$
tradeoff1	33	5.5	6	3	1.70	1.17	4* 4+
tradeoff2	18	10.1	3	2	1.78	1.18	4* 6+
tradeoff3	10	18.2	2	1	1.59	1.14	5* 4+
tradeoff4	20	9.1	4	2	1.59	1.14	5* 5+
tradeoff5	16	11.4	3	2	1.66	1.16	5* 6+
tradeoff6	43	4.2	8	4	1.67	1.16	6* 4+
tradeoff7	9	20.3	2	1	1.47	1.08	6* 5+
tradeoff8	17	10.7	4	2	1.40	1.03	6* 6+
tradeoff9	8	22.8	2	1	1.32	0.97	7* 6+
tradeoff10	40	4.5	10	5	1.32	0.97	7* 8+
tradeoff11	7	26.1	2	1	1.12	0.81	8* 8+
tradeoff12	21	8.7	6	3	1.12	0.81	9* 8+
tradeoff13	56	3.2	16	8	1.12	0.81	10* 8+

From Table 4.6 it can also be seen that in solutions with the same operations throughput, i.e. tradeoff3 and tradeoff7, a larger schedule length leads to schedules that require less number of functional units. This can be better illustrated with the help of Figure 4.11 and Figure 4.12, which show the schedules for tradeoff3 and tradeoff7 respectively. Note that in both figures the multiplications are multicycled (with throughput $TP_M = 2$) and the additions are single cycled (with throughput $TP_A = 1$). It can also be noted that both schedules meet the time constraint of 183ns although the schedule length is different, i.e. 10 csteps in Figure 4.11 and 9 csteps in Figure 4.12. For the throughputs $TP_M = 2$ and $TP_A = 1$, the schedule length of 10 csteps allows a better distribution of the operations than a schedule length of 9 csteps because fewer operations overlap their execution times. For example, in Figure 4.11 a maximum of 5 multiplications are overlapping their execution times in csteps 3, 5 and 6, whereas a maximum of 4 additions are overlapping their execution times in csteps 8, 9 and 10. In Figure 4.12, a maximum of 6 multiplications are overlapping their execution times in csteps 3 and 5, whereas a maximum of 5 additions are overlapping their execution times in cstep 9.

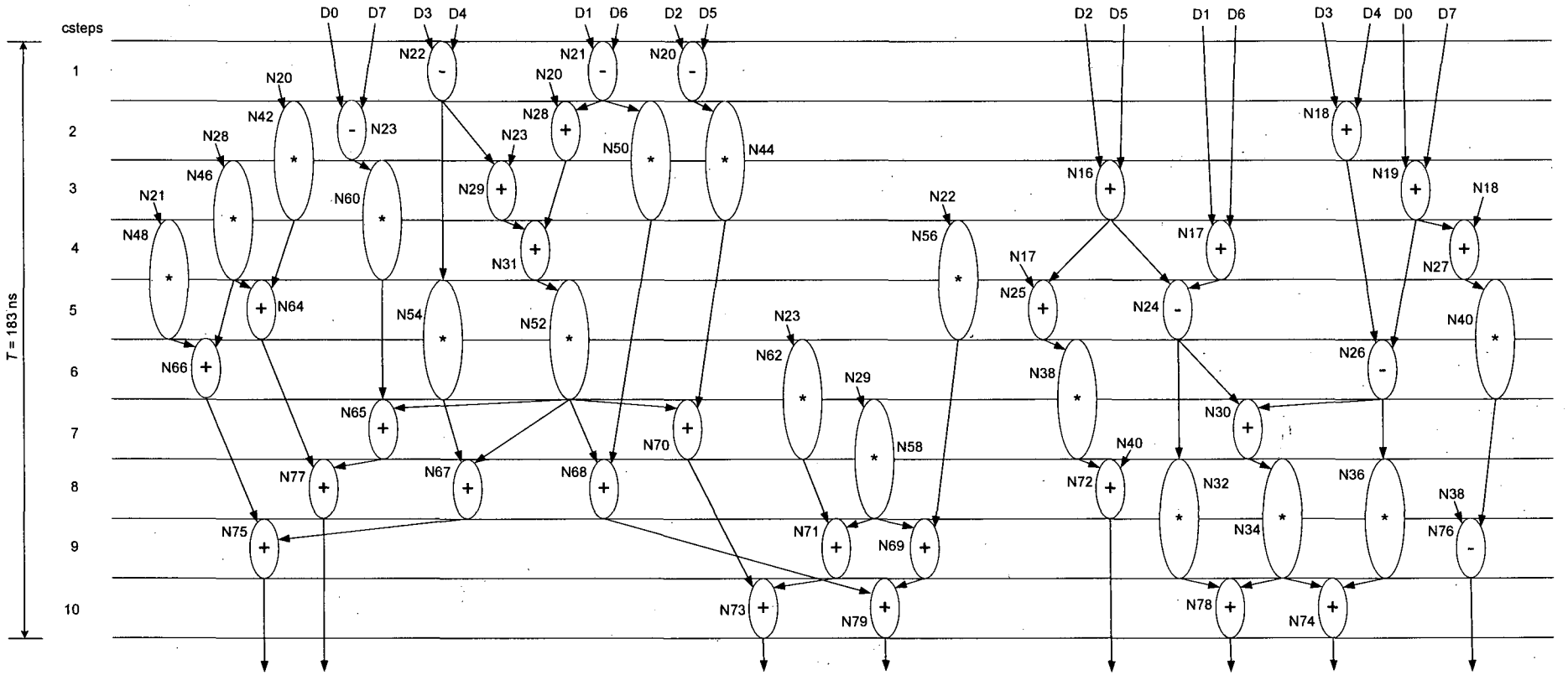


Figure 4.11 DCT schedule of tradeoff3

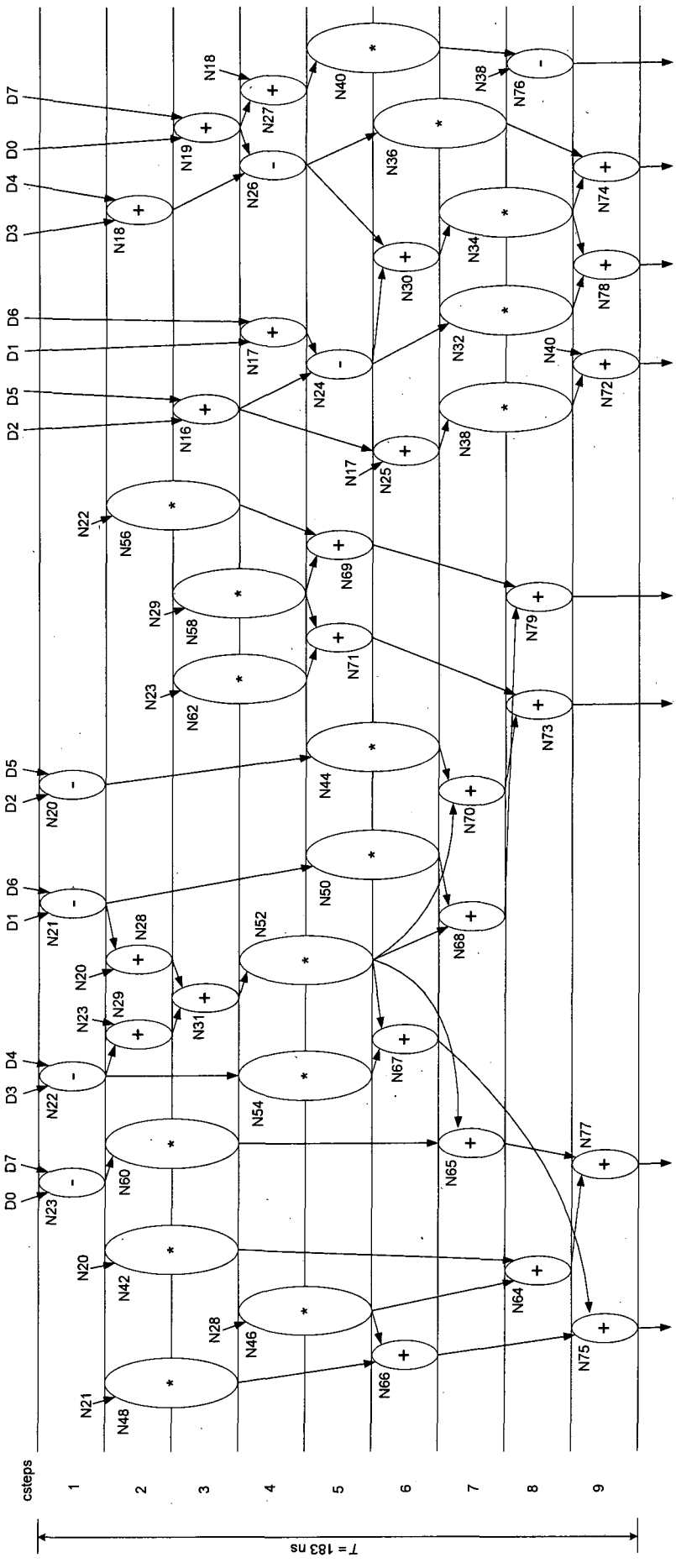


Figure 4.12 DCT schedule of tradeoff7

EWF

This benchmark presents complex data dependencies that make difficult to obtain an optimal solution. The EWF data flow graph [58] consists on 26 additions and 8 multiplications. The power-area tradeoffs of EWF for different time constraints are shown in Figure 4.13. As in the case of DIFFEQ and DCT, the power decreases gradually when the time constraint increases for the same resource requirements. For example, for 1* 2+, the power consumption reduces steadily from 354μW at 317ns to 89μW at 634ns due to a gradual voltage reduction from 1.79V to 0.90V, as shown in Table 4.7.

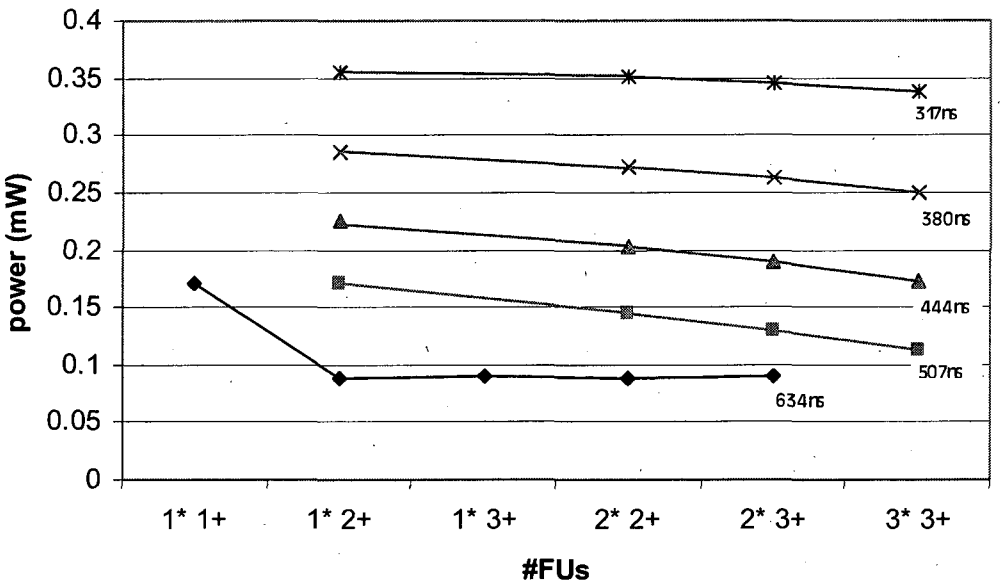


Figure 4.13 Power-area tradeoffs for EWF with different time constraints

Table 4.7 Voltage, power and energy consumption for EWF with 1* 2+

T (ns)	$slack_{sch}^{*}$ (ns)	V (V)	E (pJ)	P (μW)
317	63.4	1.79	112.2	354
380	126.4	1.61	108.7	286
444	190.4	1.42	99.9	225
507	253.4	1.24	86.2	170
634	380.4	0.9	56.4	89

* considering that all the operations are executed at 1.8V

From Figure 4.13, it can be seen that unlike DIFFEQ and DCT, as the number of FUs increases the power consumption decreases. To illustrate the cause of this power

reduction consider Table 4.8, which shows the solutions obtained for a time constrained of 444ns. It can be seen that as the resources increase from 1*2+ (denoted as tradeoff1) to 3*3+ (denoted as tradeoff4) the operating voltage decreases steadily from 1.42V to 1.12V, reducing the power consumption from 225μW to 173μW. Note that the values of clock period and operations throughput from Table 4.8 determine not only the operating voltage, but also the number of functional units used by the schedule. For example, consider Figure 4.14 and Figure 4.15, which show the schedules for tradeoff1 and tradeoff4 respectively. It can be seen that the operations are better distributed in Figure 4.14 than in Figure 4.15, due to a larger schedule length, i.e. 21 csteps when compared with 17 csteps. This results in no multiplications overlapping their execution times and a maximum of 2 additions overlapping their execution times (for example in cstep 1) in Figure 4.14. In Figure 4.15, a maximum of 3 multiplications are overlapping their execution times (for example in cstep 15), and a maximum of 3 additions are overlapping their execution times (for example in cstep 17).

Table 4.8 Characteristics of power-area tradeoffs for EWF, T = 444 ns

	Ls (csteps)	*Tclk* (ns)	TP_M (csteps)	TP_A (csteps)	*V* (V)	*P* (μW)	*#FUs*
tradeoff1	21	21.1	2	1	1.42	225	1* 2+
tradeoff2	95	4.7	10	5	1.29	204	2* 2+
tradeoff3	18	24.7	2	1	1.21	190	2* 3+
tradeoff4	17	26.1	2	1	1.12	173	3* 3+

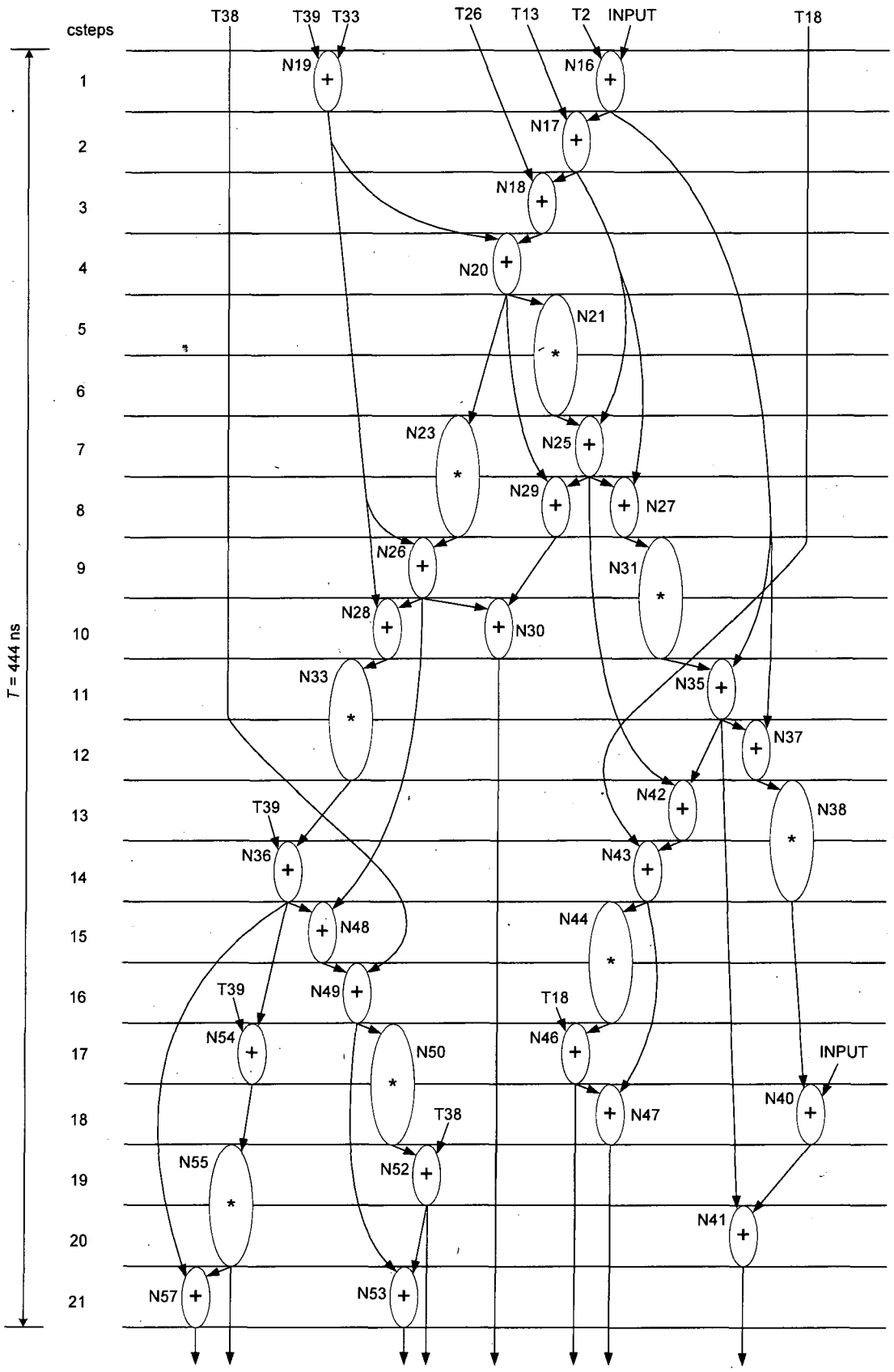


Figure 4.14 EWF schedule of tradeoff1

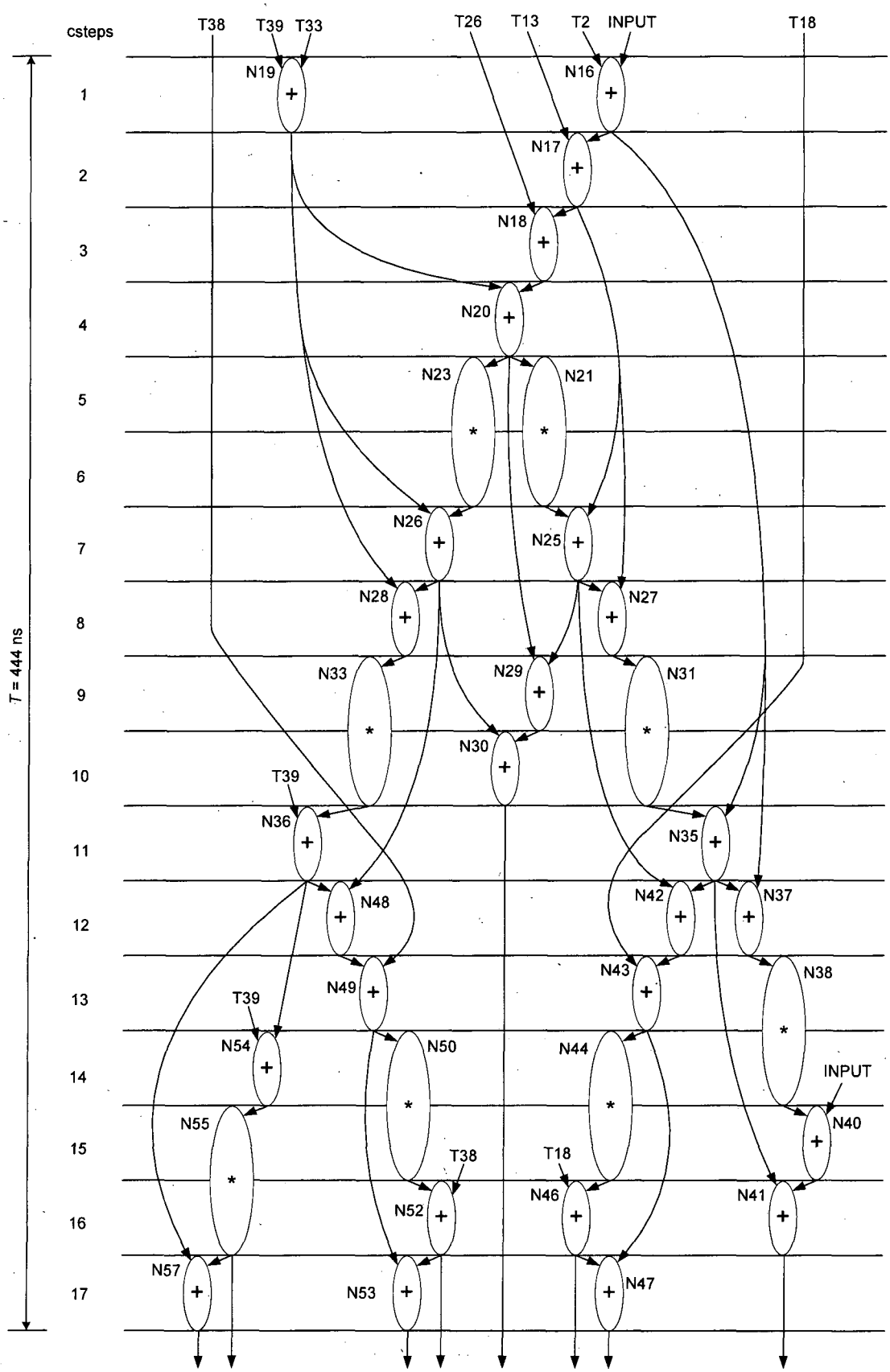


Figure 4.15 EWF schedule of tradeoff4

Computational time

A general observation from Figure 4.8, Figure 4.10 and Figure 4.13 is that the relation between power and area is non-linear and varies depending on the benchmark, the time constraint and the functional resources used. This makes the design space exploration more complex and hence time consuming, and an efficient algorithm is needed. Due to the iterative nature of PATICS, the running times for large designs may be high. Therefore a sought feature of the algorithm from [58] is its low computational complexity ($O(nl)$), where n is the number of operations and l is the schedule length. During the execution of PATICS, the number of times the modified scheduler (based on [58]) is used is ($T*max\_f - cp$), where cp is the critical path of the design, and T and max_f are respectively the time constraint and maximum frequency set by the user. Hence, the complexity of the proposed algorithm is ($O(nl(T*max\_f - cp))$). PATICS has reasonably low computational times considering the number of solutions obtained, as shown in Table 4.9, Table 4.10 and Table 4.11. For example, the algorithm needs 154s to obtain the set of solutions of Table 4.6 (DCT with a time constraint of 183ns). It can also be seen that for all the benchmarks the computational time increases as the time constraint increases. This is due to the fact that the number of clock periods to be analysed by the algorithm increases as the time constraint increases. For example, assuming a minimum clock period *min_Tclk* of 2ns and a time constraint of 317ns for EWF, the proposed algorithm analyse 144 clock periods. However, by changing the time constraint to 444ns, the number of clock periods analysed increased to 208.

Table 4.9 Run times for DIFFEQ

Time Constraint (ns)	Computational time (s)
112	1
134	2
156	3
179	4
224	9

Table 4.10 Run times for EWF

Time Constraint (ns)	Computational time (s)
317	77
380	193
444	366
507	610
634	1511

Table 4.11 Run times for DCT

Time Constraint (ns)	Computational time (s)
130	44
156	93
183	154
209	232
261	485

4.5.2 Comparison with MSV

This experiment demonstrates that the single supply voltage (SSV) solutions obtained by PATICS present comparable power consumption with the solutions obtained by a multiple supply voltage (MSV) algorithm [126] that uses two voltages, 1.8V and 0.9V. Comparable power values have been achieved without increasing the functional resource usage whilst meeting the imposed time constraint. Besides the comparable power consumption, the proposed algorithm avoids the problems associated with MSV, such as high routing cost of the supply lines and area/delay overhead of required level shifters [10]. As can be seen from Figure 4.16, the proposed algorithm produces solutions of comparable quality in terms of power than those generated using MSV for DIFFEQ. For example the power consumption for 2* 1+ for a time constraint of 134ns is 0.57mW with the presented algorithm, whilst it is 0.53mW with MSV. In the case of EWF, a power dissipation of 0.337mW for 1* 2+ at 328ns is obtained with MSV, whereas the proposed algorithm obtains a schedule that dissipates 0.341mW, as can be seen from Figure 4.17. Overall, it can be concluded that the proposed algorithm obtains comparable power values with the MSV algorithm.

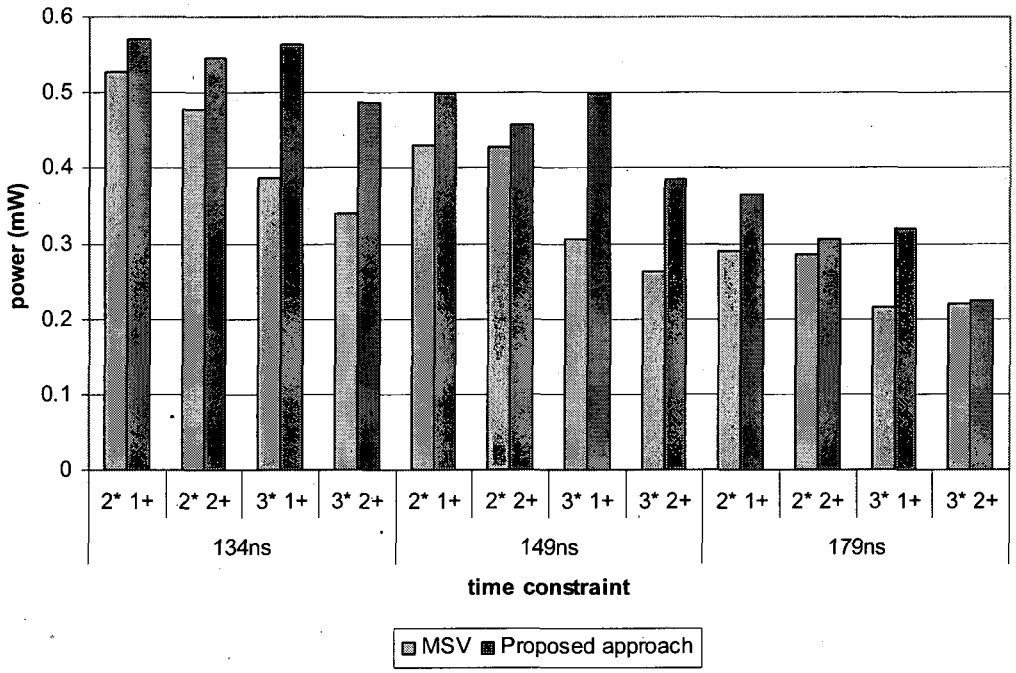


Figure 4.16 Power consumption of DIFFEQ using MSV [126] and the proposed algorithm

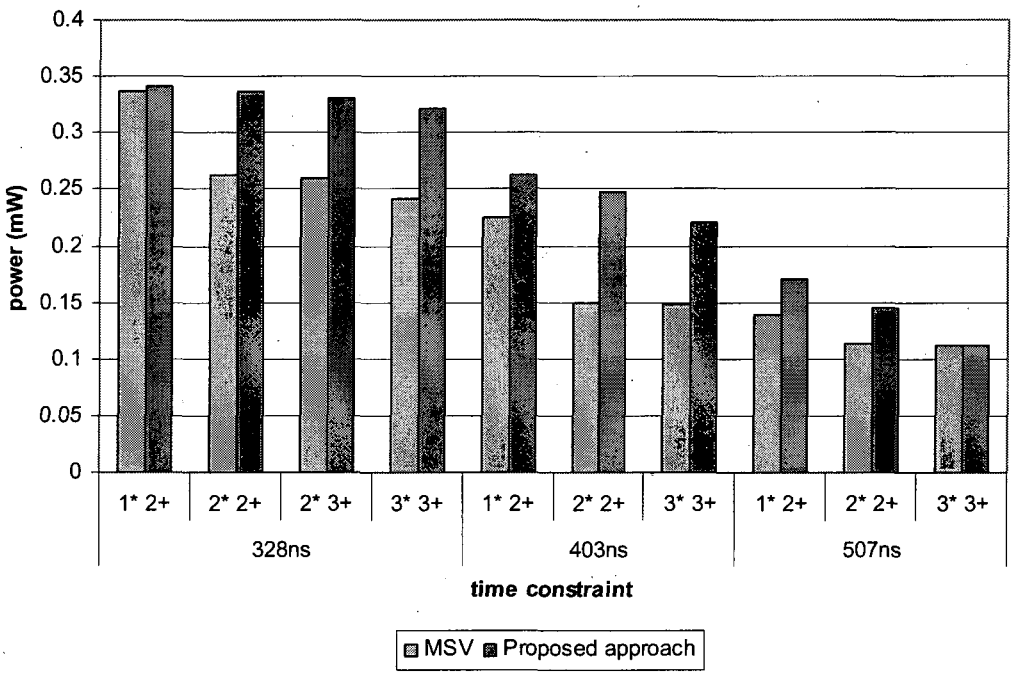


Figure 4.17 Power consumption of EWF using MSV [126] and the proposed algorithm.

4.5.3 Comparison with an area optimised scheduler

The aim of this section is to demonstrate the benefits of using the proposed algorithm compared with a time constrained scheduling (TCS) algorithm that targets area optimisation but that is not power-aware. Although many TCS algorithms have been proposed, the algorithm developed in [58] has been chosen because of its good quality solutions and low computational complexity. The schedules obtained with [58] consider that all the FUs are operating at maximum supply voltage, i.e. 1.8V, and that the clock period was set to the fastest functional unit from Table 4.1 when operating at maximum supply voltage. Figure 4.18 shows the power saving that can be achieved when applying the proposed algorithm to the benchmarks DIFFEQ, EWF and DCT. Note that all the power savings were obtained with the same number of FUs that [58]. For example, in the case of DIFFEQ with 2* 1+, the power saving is approximately 15% when the time constraint is 179ns. For EWF, a power saving of 35% is obtained at 507ns with 2* 2+, whereas DCT with 4* 3+ experiences a power reduction of 8% at 261ns. This improvement in power consumption is due to the low operating voltage obtained after the appropriate selection of the clock period and operations throughput. The operating voltages for DIFFEQ 2* 1+, EWF with 2* 2+ and DCT with 4* 3+ are 1.35V, 1.09V and 1.49V respectively. From Figure 4.18, it can also be seen that the power savings increase when increasing the time constraint, as previously explained in Section 4.5.1.

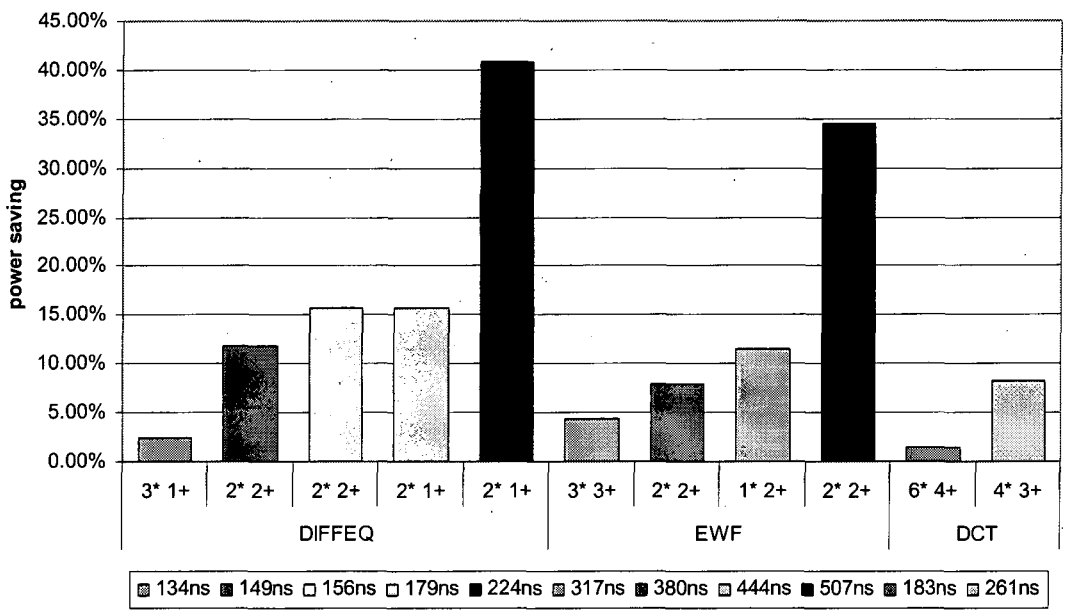


Figure 4.18 Power savings compared with TCS [58] without increasing the number of FUs

The proposed algorithm also obtains other solutions with better power reduction than the ones shown in Figure 4.18, but with greater number of functional resources when compared to [58]. For example, Table 4.12, Table 4.13 and Table 4.14 present respectively the power savings for DIFFEQ, EWF and DCT with an extra multiplier or adder.

Table 4.12 DIFFEQ

	Additional FUs	% power saving
134ns	1+	15.8
149ns	1*	25.9
156ns	1*	31.7
179ns	1+	29.3
224ns	1*	49.8

Table 4.13EWF

	Additional FUs	% power saving
380ns	1+	11.4
444ns	1*	19.4
507ns	1+	41.8

Table 4.14 DCT

	Additional FUs	% power saving
183ns	1+	8.7
261ns	1+	13.5

4.6 Concluding Remarks

This chapter presented a new TCS algorithm capable of exploring the design space and finding trade-offs between power consumption and area. It has been shown that power consumption and area have a non-linear relation, thus resulting in a large and complex search space. The proposed algorithm is capable of exploring this search space in an efficient way and with reasonable computational time. Relevant power-area tradeoffs are possible because of the careful choice of clock period and operations throughput, and the generated single supply voltage. The combination of these three parameters in the proposed algorithm is essential to obtain low power and area designs. It has been shown that power savings comparable to those obtained by MSV algorithms are achievable whereby the proposed algorithm leads to lower implementation complexity (single supply voltage versus multiple supply voltages). For example, some solutions obtained for DIFFEQ and EWF have respectively less

than 8% and 2% power increase when compared to solutions that use MSV [126]. Moreover, when compared with an area optimised scheduler [58], the proposed algorithm meets the same time constraint with the same resource requirements but with less power. For example, a power saving of 13% averaged over DIFFEQ, EWF and DCT with different time constraints was obtained.

Chapter 5

Power-Aware Behavioural Compiler (PABCOM)

5.1 Introduction

An essential task when synthesising a design from a behavioural description is selecting the clock period to schedule the DFG operations into control steps. Chapter 4 has illustrated the significant effect that clock and operations throughput selection has on the scheduling task in terms of power. However, determining the clock period has also an interaction with binding [96], i.e. resource sharing. Hence, to find good solutions in terms of power and area, the behavioural synthesis tasks (scheduling and binding) should be performed simultaneously with clock and operations throughput selection. This chapter presents a Power-Aware Behavioural COMpiler (PABCOM) that considers the interrelation between the behavioural synthesis tasks (scheduling and binding) and the clock and operations throughput selection. For a given time constraint, PABCOM achieves low power datapaths by determining a low supply voltage after using the improved algorithm for clock and operations throughput selection described in Section 5.2. Further power reduction in the multiplexer based interconnections of the datapath is achieved by employing the techniques described in Section 5.3. The proposed algorithm is described in Section 5.4 including the compound cost function that allows obtaining different power-area tradeoffs according to the optimisation goal set by the user. Section 5.5 demonstrates the efficiency of the algorithm through extensive experimental results using a number of benchmarks. The concluding remarks of the chapter are given in Section 5.6.

5.2 Improved algorithm for clock and operations throughput selection

PABCOM obtains power savings by reducing the interconnection complexity and the operating voltage of the datapath. The complexity of multiplexers-based interconnections is reduced using the techniques presented in Section 5.3, whereas the low operating voltage is obtained after using a modified version of the algorithm developed in Chapter 4. This section presents the main modifications done. The first modification eliminates redundant clock and operations throughput, hence decreasing the computational time of PABCOM. The second modification aims to obtain a lower operating voltage by exploring the throughputs of operations that do not necessarily consume most of the power in the design. This modification was implemented in a function called *upper_bound* whereas the first modification was implemented in the function *add_to_list*. These two functions are included in Listing 5.1, which shows the modified algorithm for clock and operations throughput selection. This modified algorithm is integrated into PABCOM as shown later in Section 5.4.

Listing 5.1 Modified algorithm for clock and operations throughput selection

```
1 Lines 1 to 8 from Listing 4.1 (Chapter 4)
2 Lines 10 to 15 from Listing 4.1 (Chapter 4)
3               if Ls >= critical_path_csteps do
4                       (Ls, TP_power_op, TP_remain_op, V) to l_b_struct
5                       upper_bound (Ls, min_V, u_b_struct)
6                       add_to_list (l_b_struct, u_b_struct, clock_throughputs)
7               end if
8       else
9               calculate TP_power_op (D_power_op(min_V))
10              calculate TP_remain_op (D_remain_op(min_V))
11              critical_ path_csteps  = ASAP(TP_power_op, TP_remain_op)
12              if Ls >= critical_path_csteps do
13                      (Ls, TP_power_op, TP_remain_op, V) to l_b_struct
14                      upper_bound (Ls, min_V, u_b_struct)
15                      add_to_list (l_b_struct, u_b_struct, clock_throughputs)
16              end if
17      end if
18      increase Ls
19      calculate Tclk
20 end while
```

From Listing 5.1, it can be seen that the first steps (lines 1 to 2) are the same that in the original algorithm described in Listing 4.1. These steps lead to the computation of the lower bound of the operations throughput and the critical path. After validating

the critical path (line 3), the schedule length *Ls*, operating voltage *V* and throughputs TP_{power_op} and TP_{remain_op} are saved into structure *l_b_struct* (line 4). Then the function *upper_bound* (line 5) is called to calculate the upper bound of the operations throughput, which is saved into structure *u_b_struct*. Finally, the function *add_to_list* (line 6) apply pruning techniques to decide whether the structures *l_b_struct* and *u_b_struct* will be added to the list *clock_throughputs* or not. To facilitate the explanation of the algorithm from Listing 5.1, the functions *add_to_list* and *upper_bound* are described later in Listing 5.2 and Listing 5.3 respectively. From Listing 5.1 it can be seen that in case the increased delay D_{power_op} is greater than the maximum delay $maxD_{power\_op}$, the throughputs TP_{power_op} and TP_{remain_op} are calculated (lines 9 and 10) at minimum voltage *min_V* = 1.08V. This voltage was selected according to the library specified in Table 5.9. Then the critical path is calculated (line 11) and a similar process (line 12 to 16) to the one described in lines 3 to 7 is followed.

Listing 5.2 shows the pseudocode of the function *upper_bound* used in Listing 5.1. This function allows exploring solutions that were not explored by the original algorithm developed in Chapter 4 with the aim of further voltage reduction.

Listing 5.2 Pseudocode of the function *upper_bound*

```
1 upper_bound (Ls, min_V, u_b_struct)
2 {
3       calculate TP_power_op (D_power_op(min_V))
4       calculate TP_remain_op (D_remain_op(min_V))
5       critical_ path_csteps  = ASAP(TP_power_op, TP_remain_op)
6       if Ls >= critical_ path_csteps do
7             (Ls, TP_power_op, TP_remain_op, min_V) to u_b_struct
8       else
9             while (upper_bound not found) do
10                  decrease TP_power_op
11                  calculate D_power_op
12                  calculate V1
13                  decrease TP_remain_op
14                  calculate D_remain_op
15                  calculate V2
16                  if (V1 > max_V && V2 < max_V) do
17                        calculate TP_power_op (D_power_op(V2))
18                        critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
19                        if Ls >= critical_ path_csteps do
20                              (Ls, TP_power_op, TP_remain_op, V2) to u_b_struct
21                        end if
22                  elsif (V1 < max_V && V2 > max_V) do
23                        calculate TP_remain_op (D_remain_op(V1))
24                        critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
25                        if Ls >= critical_ path_csteps do
26                              (Ls, TP_power_op, TP_remain_op, V1) to u_b_struct
```

```
27                          end if
28                    elsif (V1 < max_V && V2 < max_V) do
29                          if (V1 < V2) do
30                                calculate TP_remain_op (D_remain_op(V1))
31                                critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
32                                if Ls >= critical_ path_csteps do
33                                      (Ls, TP_power_op, TP_remain_op, V1) to u_b_struct
34                                else
35                                      calculate TP_remain_op (D_remain_op(V2))
36                                      calculate TP_power_op (D_power_op(V2))
37                                      critical_ path_cs=ASAP(TP_power_op, TP_remain_op)
38                                      if Ls >= critical_ path_csteps do
39                                          (Ls, TP_power_op, TP_remain_op, V2) to u_b_struct
40                                      end if
41                                end if
42                          else
43                                calculate TP_power_op (D_power_op(V2))
44                                critical_ path_csteps = ASAP(TP_power_op, TP_remain_op)
45                                if Ls >= critical_ path_csteps do
46                                      (Ls, TP_power_op, TP_remain_op, V2) to u_b_struct
47                                else
48                                      calculate TP_power_op (D_power_op(V1))
49                                      calculate TP_remain_op (D_remain_op(V1))
50                                      critical_ path_cs=ASAP(TP_power_op, TP_remain_op)
51                                      if Ls >= critical_ path_csteps do
52                                          (Ls, TP_power_op, TP_remain_op, V1) to u_b_struct
53                                      end if
54                                end if
55                          end if
56                    else
57                          break
58                    end if
59              end while
60      end if
61 end if
62 }
```

Lines 3 to 5 of this function calculate the upper bound of the operations throughput and critical path as in Listing 4.1, Chapter 4. After the validation of the critical path (line 6), the schedule length *Ls*, operating voltage *min_V* and throughputs TP_{power_op} and TP_{remain_op} are saved into the structure *u_b_struct* (line 7). However, if the critical path is larger than the schedule length, the upper bound of the operations throughput will be determined by decreasing the throughput of the operations, and then calculating their delay and operating voltage (lines 10 to 15). For example, consider a schedule length *Ls* = 9 csteps for the DIFFEQ benchmark, where TP_{power_op} and TP_{remain_op} represent the operations throughputs of the multiplications and additions respectively. Then, at *min_V* = 1.08V, the operations throughput are $TP_{power\_op} = 3$ and $TP_{remain\_op} = 2$. With these throughputs, the critical path of DIFFEQ is executed in 10 csteps, which violates the schedule length *Ls*. Then the throughputs need to be

decreased as explained in the following. Firstly, TP_{power_op} is reduced to 2 csteps, which results in a multiplication delay of 10.6ns using a voltage $V1$ = 1.13V, as shown in Figure 5.1a. Secondly, TP_{remain_op} is decreased to 1 cstep, resulting in an addition delay of 5.3ns using a voltage $V2$ = 1.11V (see Figure 5.1b). The algorithm then evaluates the feasibility of $V1$ and $V2$ by comparing them with the maximum allowed voltage, max_V = 1.32V according to the library from Table 5.9. There are three possible cases: $V1$ is unfeasible and $V2$ feasible, $V1$ is feasible and $V2$ is unfeasible, and $V1$ and $V2$ are feasible.

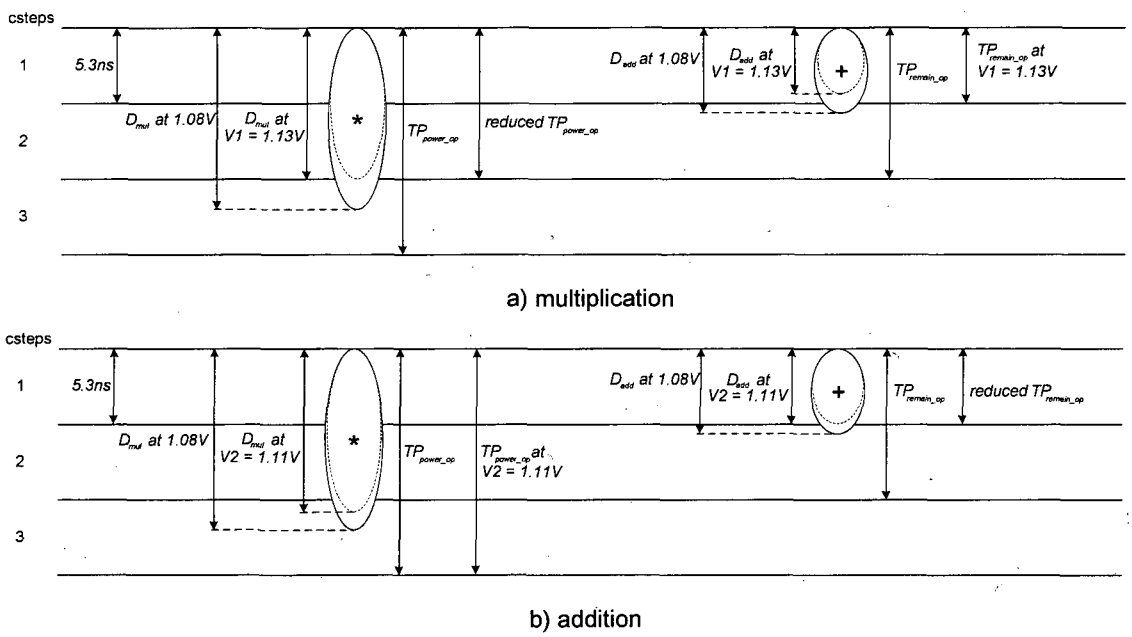


Figure 5.1 Operations throughput decrease

In case $V1$ is unfeasible and $V2$ is feasible, TP_{power_op} is calculated at $V2$ (line 17) and later used to calculate the critical path (line 18), which is validated (line19) before saving the parameters Ls, TP_{power_op}, TP_{remain_op}, and $V2$, into the structure *u_b_struct* (line 20). In case $V1$ is feasible and $V2$ is unfeasible, TP_{remain_op} is calculated at $V1$ (line 23) and later used to calculate the critical path (line 24), which is validated (line 25) before saving the parameters Ls, TP_{power_op}, TP_{remain_op}, and $V1$, into the structure *u_b_struct* (line 26). In case both voltages $V1$ and $V2$ are feasible, they need to be compared to determine the lowest voltage (line 29). If $V1$ is lower than $V2$, TP_{remain_op} is calculated at $V1$ (line 30) and later used to calculate the critical path (line 31), which is validated (line 32) before saving the parameters Ls, TP_{power_op}, TP_{remain_op}, and $V1$, into the structure *u_b_struct* (line 33). In the case of a schedule

length violation, the algorithm attempts to use *V2* to compute TP_{remain_op} and TP_{power_op} (line 35 and 36), which are used to compute the critical path (line 37). This is then validated (line 39) before saving the parameters *Ls*, TP_{power_op}, TP_{remain_op}, and *V2*, into the structure *u_b_struct* (line 40). A similar process (line 43 to 54) to the one explained (line 30 to 40) is followed if *V1* is not lower than *V2*, but now considering first *V2*. For example, consider again the voltages *V1* = 1.13V and *V2* = 1.11V previously obtained for DIFFEQ and the schedule length of 9 csteps. Since *V2* is lower than *V1*, the delay of the multiplication D_{mul} is calculated at *V2* and then used to compute the multiplication throughput TP_{power_op} (line 43). This results for the present case in a multiplication delay D_{mul} of 11.36ns and a multiplication throughput TP_{power_op} of 3 csteps (see Figure 5.1b). Using TP_{power_op} = 3 and the reduced TP_{remain_op} = 1 obtained previously (line 13), a critical path of 8 csteps is obtained. This critical path is lower than the schedule length of 9 csteps, thus *Ls* = 9, TP_{power_op} = 3, TP_{remain_op} = 1 and *V* = 1.11, are saved into *u_b_struct*. In case of violation of the schedule length, the algorithm attempts to use *V1* to find the upper bound of the operations throughput (lines 48 to 53).

As explained in Listing 5.1, once the structures *l_b_struct* and *u_b_struct* have been found, the function *add_to_list* is called to determine which structures should be pruned. The pseudocode of the function *add_to_list* is presented in Listing 5.3.

Listing 5.3 Pseudocode of the function *add_to_list*

```
1 add_to_list (l_b_struct, u_b_struct)
2 {
3       if (l_b_struct == u_b_struct) do
4               if (!check_redundancy(l_b_struct)) do
5                       add l_b_struct to list clock_throughputs
6                       prev_low_b = l_b_struct
7               end if
8       elsif (prev_low_b != l_b_struct || prev_upp_b != u_b_struct) do
9               if (!check_redundancy(l_b_struct)) do
10                      add l_b_struct to list clock_throughputs
11                      prev_low_b = l_b_struct
12              end if
13              if (!check_redundancy(u_b_struct)) do
14                      add u_b_struct to list clock_throughputs
15                      prev_upp_b = u_b_struct
16              end if
17      end if
18 }
```

It can be seen that in case the structures *l_b_struct* and *u_b_struct* are the same (line 3), the redundancy of the contents of only *l_b_struct* is verified with the function

check_redundancy (line 4). This function checks that the contents of the structure, i.e. Ls, TP_{power_op} and TP_{remain_op}, are not all multiples of any Ls, TP_{power_op} and TP_{remain_op} included already in the list *clock_throughputs*. For example, consider that a structure containing $Ls = 4$, $TP_{power\_op} = 1$ and $TP_{remain\_op} = 1$, is already included in the list *clock_throughputs*. Then, a structure with $Ls = 8$, $TP_{power\_op} = 2$ and $TP_{remain\_op} = 2$ will not be included in the list *clock_throughputs* since these values of schedule length and operations throughputs are all multiples of $Ls = 4$, $TP_{power\_op} = 1$ and $TP_{remain\_op} = 1$. However, a structure with $Ls = 8$, $TP_{power\_op} = 2$ and $TP_{remain\_op} = 1$, may be added to the list (line 5). Then the previous lower bound *prev_low_b* is updated with the throughputs contained in the structure *l_b_struct* (line 6).

Note that in case the throughputs of *prev_low_b* are different from the ones of *l_b_struct* or the throughputs of *prev_upp_b* are different from the ones of *u_b_struct*, the redundancy of the structures *l_b_struct* and *u_b_struct* is verified (line 9 and 13). If there is no redundancy the structures *l_b_struct* and *u_b_struct* are included in the list *clock_throughputs* (line 10 and 14). Then, the throughputs of *prev_low_b* and *prev_upp_b* are updated with the throughputs of *l_b_struct* or *u_b_struct* respectively (line 11 and 15). Note that the function *add_to_list* included in Listing 5.3 allows decreasing the number of elements of the list *clock_throughputs*, hence reducing the number of possible operating voltages when searching for a low power solution. Consequently, the computational time of the algorithm is also reduced.

An example of the resultant list *clock_throughputs* containing n clock periods is shown in Figure 5.2. This list provides the schedule length Ls and operations throughput TP_{power_op} and TP_{remain_op} necessary to generate the schedules of a list of solutions that will be used in the algorithm presented in Section 5.4. The list *clock_throughputs* also provides the operating voltage V, which is used to estimate the power dissipated by the datapath.

$Tclk_1$	Ls_1	$TP_{power\ op\ 1}$	$TP_{remain\ op\ 1}$	V_1
$Tclk_2$	Ls_2	$TP_{power\ op\ 2}$	$TP_{remain\ op\ 2}$	V_2
$Tclk_3$	Ls_3	$TP_{power\ op\ 3}$	$TP_{remain\ op\ 3}$	V_3
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
$Tclk_n$	Ls_n	$TP_{power\ op\ n}$	$TP_{remain\ op\ n}$	V_n

Figure 5.2 List *clock_throughputs*

5.3 Power reduction in multiplexer-based interconnections

This section illustrates by means of examples the techniques used to reduce the complexity of multiplexer-based interconnections in the datapath and thus achieve power savings. These techniques were previously used in [57] during a simulated annealing process to obtain an area optimised datapath, however their impact on the power consumption was not considered. The four techniques used comprise:

1) Scheduling a selected operation into a new cstep

A selected operation is scheduled into a cstep randomly chosen from the interval defined by the ASAP and ALAP values of the operation. For example, consider that operation N10 in Figure 5.3a has been chosen to be scheduled into a new cstep selected randomly between the ASAP value, i.e. 1, and the ALAP value, i.e. 5. Figure 5.3b shows the resultant schedule assuming that operation N10 has been moved from cstep 1 to the randomly selected cstep 5.

Assigning an operation to a new cstep may lead to a different module or register binding. The module binding before and after moving operation N10 is shown in Figure 5.4. It can be seen that module M5 in Figure 5.4a is unused in cstep 5 and hence the module binding can be maintained as shown in Figure 5.4b. However, the register binding after moving operation N10 can not be preserved, as shown in Figure 5.5. Moving N10 from cstep 1 to cstep 5 reduces the lifetime of value N10 starting now in cstep 6 and extends the lifetime of value *x* until cstep 5. From Figure 5.5a it can be seen that register R2 is not available in cstep 5 and a new register binding needs to be obtained using the left edge algorithm [38]. Note that this new solution can be later optimised during the simulated annealing process. After register binding, value *x* is moved to register R1 where the lifetime reduction of value N10 has resulted in a sufficiently large gap as shown in Figure 5.5b.

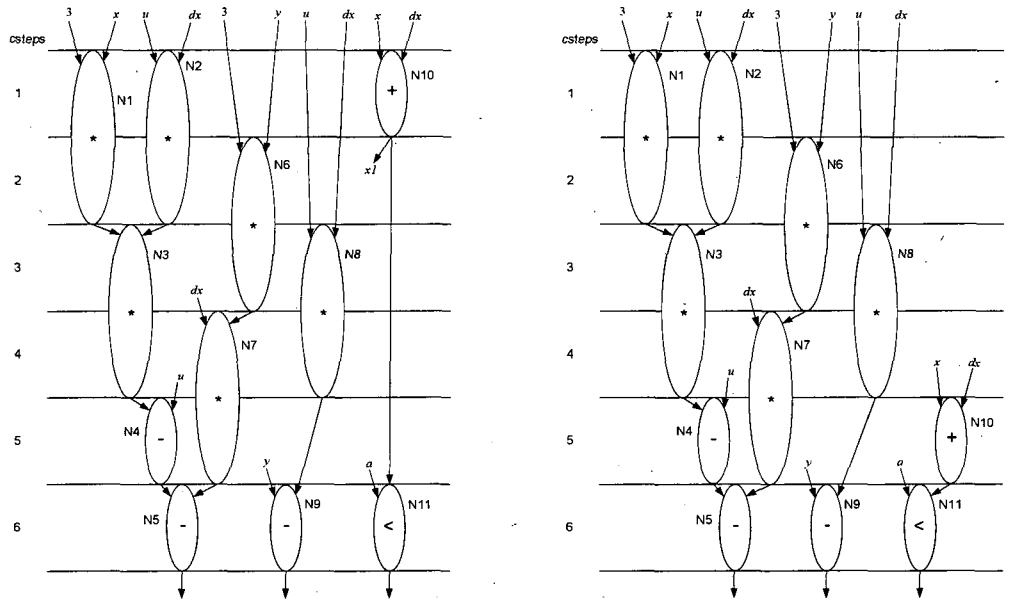
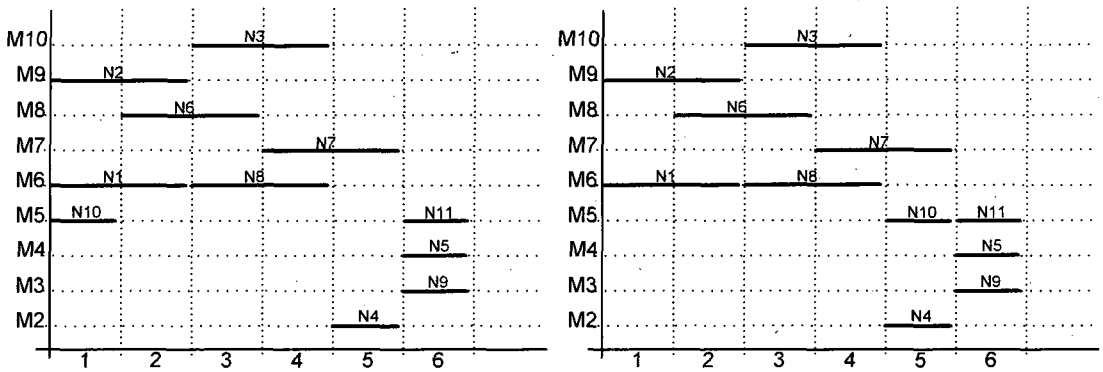
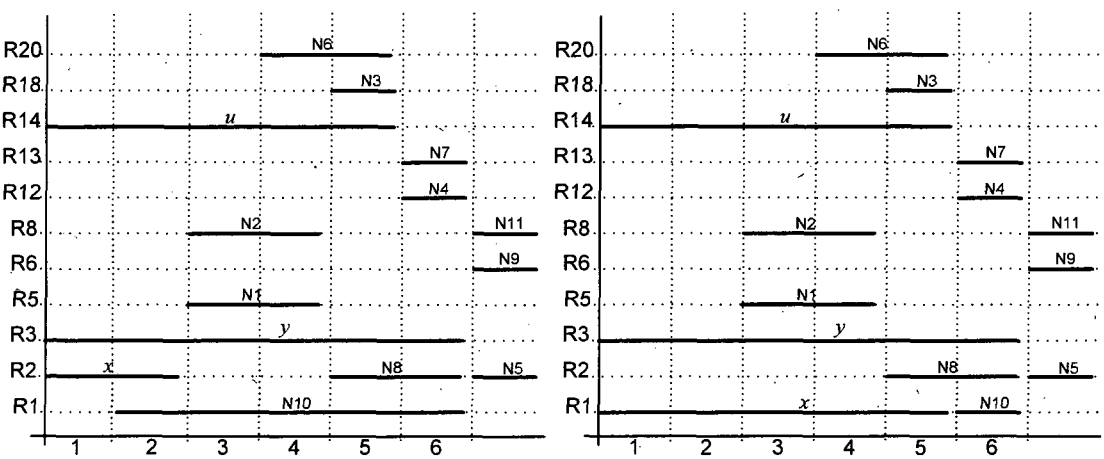


Figure 5.3 Scheduling a selected operation into a new cstep



a) before scheduling N10 into cstep 5 b) after scheduling N10 into cstep 5

Figure 5.4 Module binding



a) before scheduling N10 into cstep 5 b) after scheduling N10 into cstep 5

Figure 5.5 Register binding

From Figure 5.4 and Figure 5.5 it can be seen that the number of modules and registers remain the same after scheduling N10 into a new cstep. However, the complexity of the interconnection changes as illustrated in Table 5.1, which shows the multiplexers connected to the inputs of modules and registers. Such inputs have been named as input_a and input_b. For sake of explanation, each multiplexer has been named according to the module or register is connected to. For example, the multiplexer connected to input_a of module M5 is named as M5a, whereas the multiplexer connected to register R2 is named as R2a.

Table 5.1 Multiplexers requirement

	Scheduling N10 into cstep 1		Scheduling N10 into cstep 5	
Module or register	input_a	input_b	input_a	input_b
M5	mux2-1 (M5a)	-	-	-
M6	mux2-1 (M6a)	-	mux2-1 (M6a)	-
R2	mux3-1 (R2a)	-	mux2-1 (R2a)	-
R8	mux2-1 (R8a)	-	mux2-1 (R8a)	-

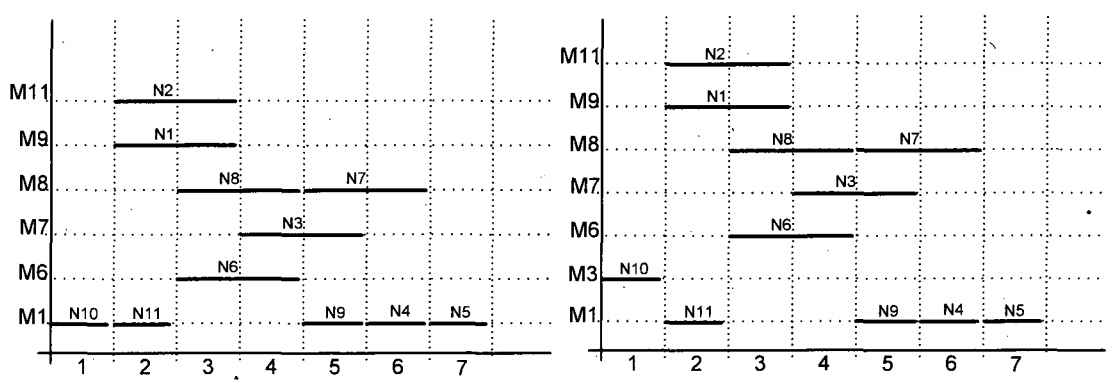
From Table 5.1, it can be seen that after scheduling N10 into cstep 5, the multiplexer M5a is no longer required and the R2a has changed from a 3-input multiplexer to a 2-input multiplexer. These two changes in the interconnection do not only affect the area requirements, but also the power consumption. This is shown in Table 5.2, which presents the number of times each multiplexer is used and its power consumption. Power reported in Table 5.2 was calculated using equation (5.5), the library from Table 5.9, and considering a time constraint of 16ns and an operating voltage of 1.32V. Note that the power reduction in R2a is due not only to the fact that the multiplexer is used less number of times, but also because a lower power multiplexer is being used, i.e. mux2-1 instead of mux3-1.

Table 5.2 Multiplexers power

	Scheduling N10 into cstep 1		Scheduling N10 into cstep 5	
multiplexer	# times used	Power (μW)	# times used	Power (μW)
M5a	2	43.0	-	-
M6a	2	43.0	2	43.0
R2a	3	95.6	2	43.0
R8a	2	43.0	2	43.0

2) Binding a selected operation to a new functional module

A selected operation is executed by a new module that is chosen randomly from a set of modules specified by the user. For example, consider that operation N10 in Figure 5.6a may be moved from module M1 to module M3 in Figure 5.6b. This new module binding results in a less complex interconnection as shown in Table 5.3. It can be seen that M1a has changed from a 5-input multiplexer to a 4-input multiplexer while the rest of the multiplexers remain the same. This affects the power consumption of the multiplexers. Consider for example Table 5.4, which reports the multiplexers power for the bindings shown in Figure 5.6. Power reported in Table 5.4 was calculated considering a time constraint of 21ns and an operating voltage of 1.32V. Note that the power dissipated by the multiplexer M1a has been reduced from 215.2μW to 97.1μW. Again, this is due not only to the less number of times M1a is used after the new binding, but also to the reduction of the number of inputs of M1a, i.e. from 5 to 4 (see Table 5.3). Binding operation N10 to module M3 was possible because the module was unused during all the csteps. However, in case the module is not available an exchange of operations between the old module and new module is attempted.



a) before binding N10 into M3 b) after binding N10 into M3

Figure 5.6 Module binding

Table 5.3 Multiplexers requirement

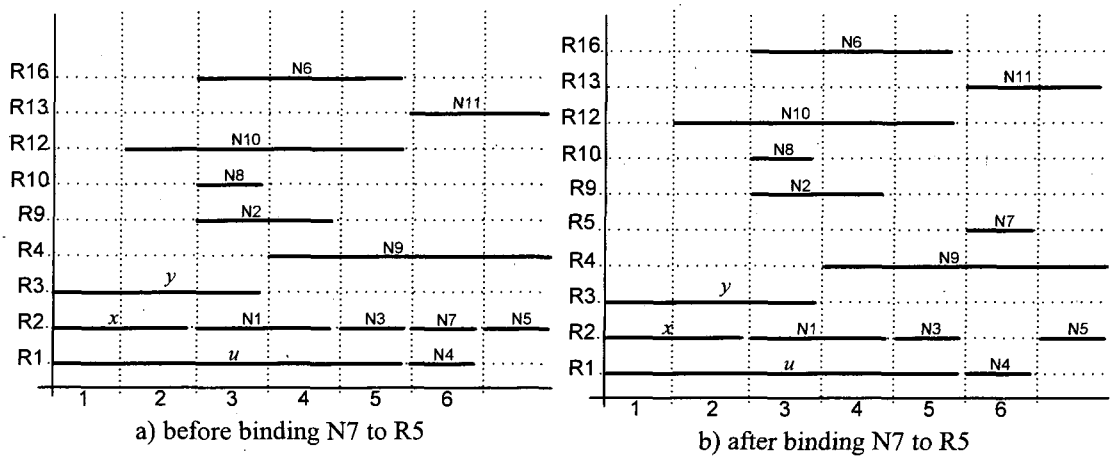
	Binding N10 to module M1		Binding N10 to module M3	
Module or register	input_a	input_b	input_a	input_b
M1	mux5-1 (M1a)	mux3-1 (M1b)	mux4-1 (M1a)	mux3-1 (M1b)
M8	mux2-1 (M8a)	-	mux2-1 (M8a)	-
R1	mux2-1 (R1a)	-	mux2-1 (R1a)	-
R2	mux2-1 (R2a)	-	mux2-1 (R2a)	-
R6	mux2-1 (R6a)	-	mux2-1 (R6a)	-

Table 5.4 Multiplexers power

	Binding N10 to module M1		Binding N10 to module M3	
multiplexer	# times used	Power (μW)	# times used	Power (μW)
M1a	5	215.2	4	97.1
M1b	3	72.8	3	72.8
M8a	2	32.7	2	32.7
R1a	2	32.7	2	32.7
R2a	2	32.7	2	32.7
R6a	2	32.7	2	32.7

3) Binding a selected operation result into a new register

A selected operation value is saved in a new register that is chosen randomly from a set of registers given by the user. For example, consider that operation value N7 in Figure 5.7a may be moved from register R2 to register R5 in Figure 5.7b. This new register binding results in a less complex interconnection as shown in Table 5.5. It can be seen that R2a has changed from a 5-input multiplexer to a 4-input multiplexer while the rest of the multiplexers remain the same. This affects the power consumption of the multiplexers as shown in Table 5.6, where the reported power dissipation was calculated considering a time constraint of 16ns and an operating voltage of 1.32V. Note that the power dissipated by the multiplexer R2a has been reduced from 282.4μW to 127.5μW. This is due not only to the less number of times R2a is used after the new binding, but also to the reduction of the number of inputs of R2a, i.e. from 5 to 4 (see Table 5.5). Binding operation value N7 to register R5 was possible because the register was unused during all the csteps. However, in case the register is not available an exchange of operations values between the old register and new register is attempted.



a) before binding N7 to R5

b) after binding N7 to R5

Figure 5.7 Register binding

Table 5.5 Multiplexers requirement

	Binding N7 to register R2		Binding N7 to register R5	
Module or register	input_a	input_b	input_a	input_b
M2	mux2-1 (M2a)	mux2-1 (M2b)	mux2-1 (M2a)	mux2-1 (M2b)
M9	mux2-1 (M9a)	mux2-1 (M9b)	mux2-1 (M9a)	mux2-1 (M9b)
R1	mux2-1 (R1a)	-	mux2-1 (R1a)	-
R2	mux5-1 (R2a)	-	mux4-1 (R2a)	-

Table 5.6 Multiplexers power

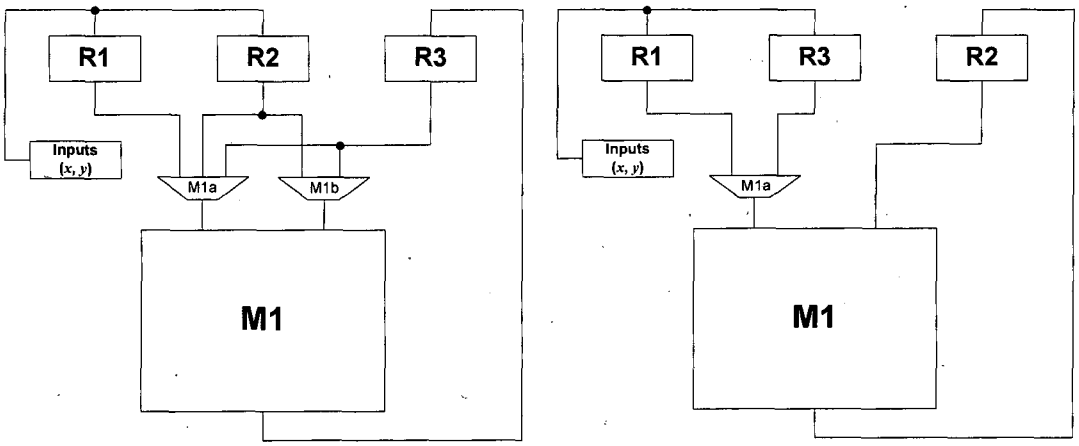
	Binding N7 to register R2		Binding N7 to register R5	
multiplexer	# times used	Power (μW)	# times used	Power (μW)
M2a	2	43.0	2	43.0
M2b	2	43.0	2	43.0
M9a	2	43.0	2	43.0
M9b	2	43.0	2	43.0
R1a	2	43.0	2	43.0
R2a	5	282.4	4	127.5

4) Swapping the inputs of a selected operation

The inputs of a randomly selected operation are swapped if the operation is commutative. This is better illustrated with the help of Table 5.7, which contains the commutative operations N1, N2 and N3, with their respective inputs and the registers where these inputs are saved. Consider that the commutative operations are executed by module M1 in the datapath shown in Figure 5.8a. It can be seen that two multiplexers are required to share M1, i.e. a 3-input multiplexer M1a and a 2-input multiplexer M1b. Figure 5.8b shows the generated datapath after swapping the inputs of operation N3. It can be seen that M1a has become a 2-input multiplexer and M1b is no longer required. Hence, the complexity of the interconnection has been reduced by swapping the inputs of N3, affecting the power consumption as illustrated in Table 5.8. Power reported in Table 5.8 was calculated considering a time constraint of 16ns and an operating voltage of 1.32V. It can be inferred from the table that the power consumption due to multiplexers has been reduced from 138.6µW to 43.0µW after swapping the inputs of operation N3.

Table 5.7 Commutative operations and their inputs

operation	input_a (register)	Input_b (register)
N1	x (R1)	y (R2)
N2	N1 (R3)	y (R2)
N3	y (R2)	N2 (R3)



a) before swapping inputs of N3 b) after swapping inputs of N3

Figure 5.8 Datapath

Table 5.8 Multiplexers power

	Before swapping N3 inputs		After swapping N3 inputs	
multiplexer	# times used	Power (μW)	# times used	Power (μW)
M1a	3	95.6	2	43.0
M1b	2	43.0	-	-

5.4 Power-aware datapath optimisation

The power reduction techniques presented in Section 5.2 and Section 5.3 have been integrated into a power-aware behavioural compiler (PABCOM) as shown in Figure 5.9. For a given time constraint, PABCOM considers concurrently the interrelation between scheduling, binding, clock and operations throughput selection while searching for solutions that meet the optimisation objective: power, area or a combination thereof. The input to PABCOM is a standard text file that contains a DFG, a set of modules and registers, annealing parameters (δ, χ_0, ϵ_s) and user defined parameters (time constraint T, maximum frequency max_f and power optimisation weight α). The output consists of a standard text-file that contains details about the datapath structure and timing information required for the synthesis of the control path. This output file is then used to write RTL Verilog for the datapath and synthesisable VHDL for the controller. Examples of the input/output file to/from PABCOM are given in Appendix 3.

From Figure 5.9 it can be seen that the input file provides the information necessary i.e. DFG, time constraint and maximum frequency, to apply the improved algorithm for clock and operations throughput selection developed in Section 5.2. This

algorithm generates the list *clock_throughputs* that contains combinations of possible clock periods and operations throughputs, as shown previously in Figure 5.2. For each of these combinations the scheduling task is performed using a modified version of [58] (as in Chapter 4, Section 4.4). After scheduling, module and register binding are generated using the left-edge algorithm [38] to produce a complete solution. Then, all the complete solutions derived from list *clock_throughputs* are grouped in the list *implementations*. The list *implementations* is then used by a power-aware datapath optimisation based on simulated annealing.

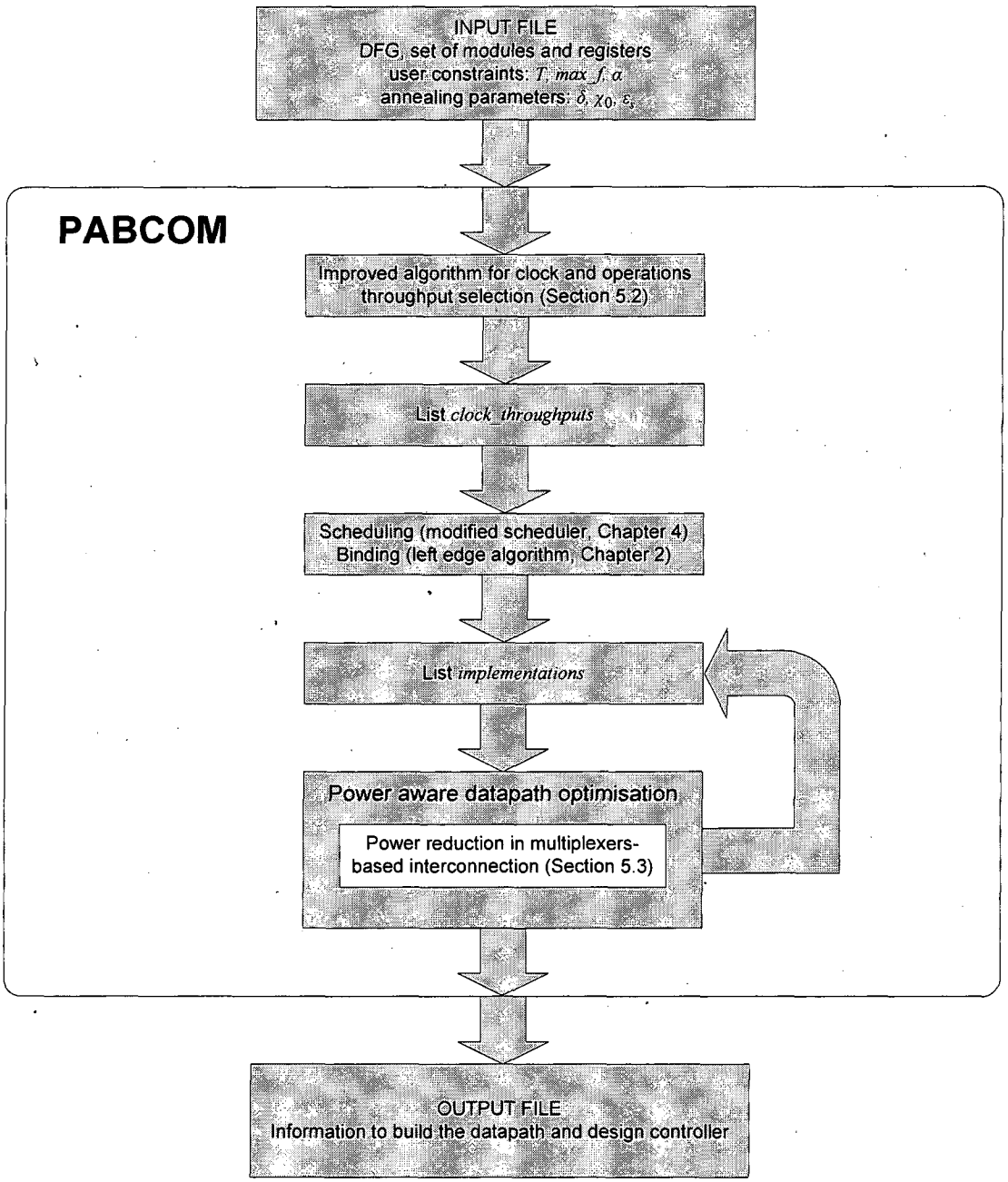


Figure 5.9 Overview of PABCOM

In the power-aware datapath optimisation, the first solution from list *implementations* is taken and new designs are examined in two nested loops as shown in Listing 5.4. The inner loop generates a number of solutions at a constant control parameter value by applying five different moves in turns. Moves 1 to 4 correspond to the power reduction techniques presented in Section 5.3. Move 5 consists on taking a new solution from the list *implementations*. This new solution includes a new clock period, operations throughput and operating voltage, resulting in different power and area requirements. After applying a move, the generated solution can be either accepted or rejected depending on the acceptance criterion defined in the simulated annealing algorithm (see Chapter 2, Section 2.4.1). The probability of accepting solutions with increasing cost depends on a control parameter, which is gradually lowered with the outer loop. The annealing process stops when the variation of the solution quality falls below a certain value [64].

Listing 5.4 Power-aware datapath synthesis algorithm

```
1 while system is not frozen do
2       while valid solutions < solutions to generate at this control parameter do
3               generate a new solution applying one of the following moves:
4                       1) schedule a randomly selected operation into a new cstep
5                       2) bind a randomly selected operation to a new functional module
6                       3) bind a randomly selected operation value to a new register
7                       4) swap the module inputs of a randomly selected operation
8                       5) clock and operations throughput selection
9               evaluate the cost of the new solution
10              accept or reject the new solution
11      end while
12      decrease control parameter
13 end while
```

5.4.1 Cost function

PABCOM considers the minimisation of not only the resource usage but also the power consumption in a given time constraint. Since two parameters, power and area, need to be optimised, the following compound cost function is used:

$$\text{cost} = \sqrt{\alpha\left(\frac{P_i}{P_0}\right)^2 + (1-\alpha)\left(\frac{Q_i}{Q_0}\right)^2} \qquad (5.1)$$

where α is the power weight defined by the user, (1-α) is the area weight, P_i and Q_i are respectively the power and area cost of a new solution, P_0 and Q_0 are respectively the maximum estimated power and maximum estimated area of the design.

The power cost P_i is the same that the estimated power consumption in the datapath of the design. To allow a quick and simple comparison among different design alternatives, the power of the datapath can be expressed as:

$$P_{DP} = P_{FU} + P_{REG} + P_{MUX} \tag{5.2}$$

where P_{DP} is the power consumption of the datapath, P_{FU} is the power dissipated by the functional units, P_{REG} is the power dissipated by the registers and P_{MUX} is the power consumption due to the multiplexers. Note that equation (5.2) does not consider the power consumed by wires used to transfer data between datapath components. Power dissipation in wires mainly depends on the switching activity and capacitance of the wire [150]. The switching activity in turn depends on the behavioural synthesis tasks scheduling and binding, whereas the wire capacitance is directly dependent on the wire length, which is determined by floorplanning. The integration of a floorplanner and PABCOM would allow considering the power dissipated by wires, which may be a significant part of the total circuit power [67]. At the moment power minimisation in the wires is out of the scope of this thesis.

The power values P_{FU}, P_{REG} and P_{MUX} from equation (5.2) are calculated assuming that the inputs of the datapath components are static when they are being used and clocks are switched off when they are idle. Hence, the power consumption of the FUs is given as [122]:

$$P_{FU} = \frac{\sum_{All\ FUs} N_F P_F D_F}{T} \tag{5.3}$$

where N_F is the number of times each type of FU is used, D_F is the delay of the FU, P_F is its average power and T is the time constraint. The power consumption of the registers and multiplexers is calculated respectively in the same way that in equation (5.3):

$$P_{REG} = \frac{\sum_{All\ registers} N_R P_R D_R}{T} \tag{5.4}$$

$$P_{MUX} = \frac{\sum_{All\ multiplexers} N_X P_X D_X}{T} \tag{5.5}$$

where N_R is the number of times each register is used, P_R is the power of a register, D_R is the delay of a register, N_X is the number of times each multiplexer is used, P_X is the power of a multiplexer and D_X is the delay of a multiplexer.

The area cost Q_i corresponds to the estimated area of the datapath in the design. To allow a quick and simple comparison among different design alternatives, the area of the datapath can be expressed as:

$$Q_{DP} = \sum_{All\ FUs\ used} F a_F + R a_R + X a_X \tag{5.6}$$

where Q_{DP} is the area of the datapath, F is the number of FUs used of each type, a_f is the area of each type of FU, R is the number of registers used, a_R is the area of a register, X is the number of multiplexers and a_X is the area of a multiplexer.

5.4.2 Cooling schedule

The parameters that determine the cooling schedule of the simulated annealing process used in PABCOM are calculated as follows:

a) Initial control parameter value and decrement rule [2]

The initial control parameter value c_0 is determined in such a way that nearly all new generated solutions (for example 95% [1]) are accepted at the beginning of the annealing process. The first step to calculate the initial control parameter is setting c_0 = 0 and then generate a sequence of m_0 solutions. After each generated solution, a new c_0 is calculated using:

$$c = \overline{\Delta f^+} \left(\ln \frac{m_2}{m_2 \chi_0 - m_1 (1 - \chi_0)} \right)^{-1} \tag{5.7}$$

where χ_0 is the initial acceptance ratio, m_2 and m_1 is the number of cost increasing and cost decreasing solutions, and $\overline{\Delta f^+}$ is the average difference in cost over m_2. The final value that equation (5.7) converges to is the initial control parameter c_0.

To achieve small decrements in the control parameter c, the stationary distributions at the end of the Markov chains need to be close to each other [64]. Consequently, information about the cost distribution within a Markov chain is included in the following decrement rule:

$$c' = c \cdot \left(1 + \frac{c \cdot \ln(1+\delta)}{3\sigma(c)} \right)^{-1} \tag{5.8}$$

where $\sigma(c)$ is the standard deviation of the cost function values at the control parameter value c and δ is the distance parameter that determines the speed at which

the control parameter is lowered. Small values of δ result in small decrements in c, whereas large values of δ result in large decrements in c.

b) Algorithm stop criterion

The annealing process is terminated after the standard deviation $\sigma(m)$ falls below the given value for the stop parameter ϵ_s:

$$\sigma(m) < \epsilon_s \tag{5.9}$$

$$\sigma(m) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\overline{f_i} - \overline{f}\right)^2} \tag{5.10}$$

where m is a number of Markov chains, $\overline{f_i}$ is the average cost value of the i-th Markov chain and $\overline{f}$ is the average value of all $\overline{f_i}$ over the m Markov chains.

c) Length of the Markov chains

The Markov chain length needs to be chosen such that the algorithm has a sufficiently large probability of exploring at least a major part of the solution neighbourhood [1]. A straightforward choice is given with the following equation:

$$L = 2 \cdot N \cdot (M + R + 1) \tag{5.11}$$

where N is the number of operations in the behavioural description, M is the number of available functional modules and R the number of registers. Hence, the Markov chain length is constant for a given synthesis problem.

5.4.3 Choice of annealing parameters

Although the performance of the used simulated annealing algorithm is determined with the parameters δ, χ_0 and ϵ_s, not all of them have the same impact on the quality of the solution. The quality of the solution is highly dependent on the distance parameter δ and almost independent from the initial acceptance ratio χ_0 and the stop parameter ϵ_s when both are chosen with acceptable degree of accuracy [1]. Therefore, to achieve low cost solutions the distance parameter δ needs to be carefully selected. To obtain a simple relation between δ and the solution quality, the proportional differences to the optimum solutions have been averaged. Figure 5.10 shows this dependence for the benchmarks: autoregressive filter (AR), elliptical wave filter (EWF) and discrete cosine transform (DCT). It can be seen that the expected design quality (average difference from the best solution) decreases continuously for EWF

and DCT with values $\delta \leq 0.7$, and for AR with values $\delta \leq 0.5$. This shows that the appropriate selection of the value δ depends on the design problem. To ensure the highest possible solution quality, it is advisable to synthesise the design with a set of different values. During the synthesis of the motion vector reconstructor in Chapter 6 it has been found empirically that $0.1 \leq \delta \leq 0.2$ represents a good initial choice.

While the initial acceptance ratio χ_0 is set to 0.95 [1], the stop parameter ϵ_s requires further analysis. Figure 5.11 shows the percentage of cost increase as a function of the stop parameter ϵ_s for AR, EWF and DCT. It can be seen that using small values of the stop parameter, i.e. $\epsilon_s \leq 0.0001$, lead to solutions with insignificant or null cost increase, whereas using values of $\epsilon_s > 0.0001$ result in solutions with a large cost increase. During the synthesis of the motion vector reconstructor in Chapter 6 it has been found empirically that $\epsilon_s \leq 0.0001$ represents a good initial choice.

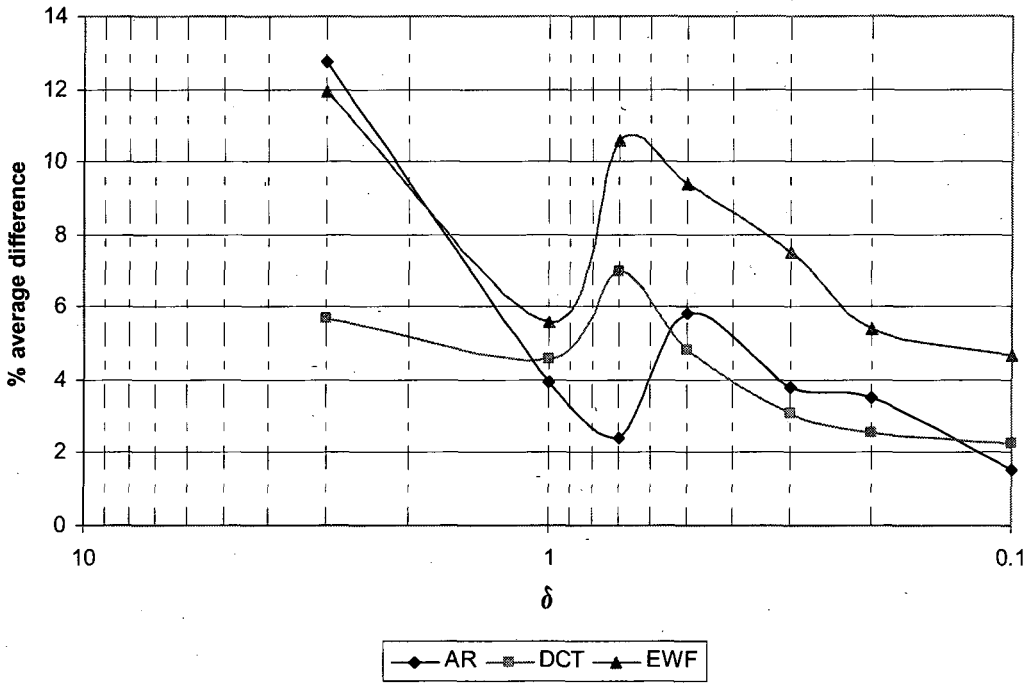


Figure 5.10 Dependence of design cost on the parameter δ

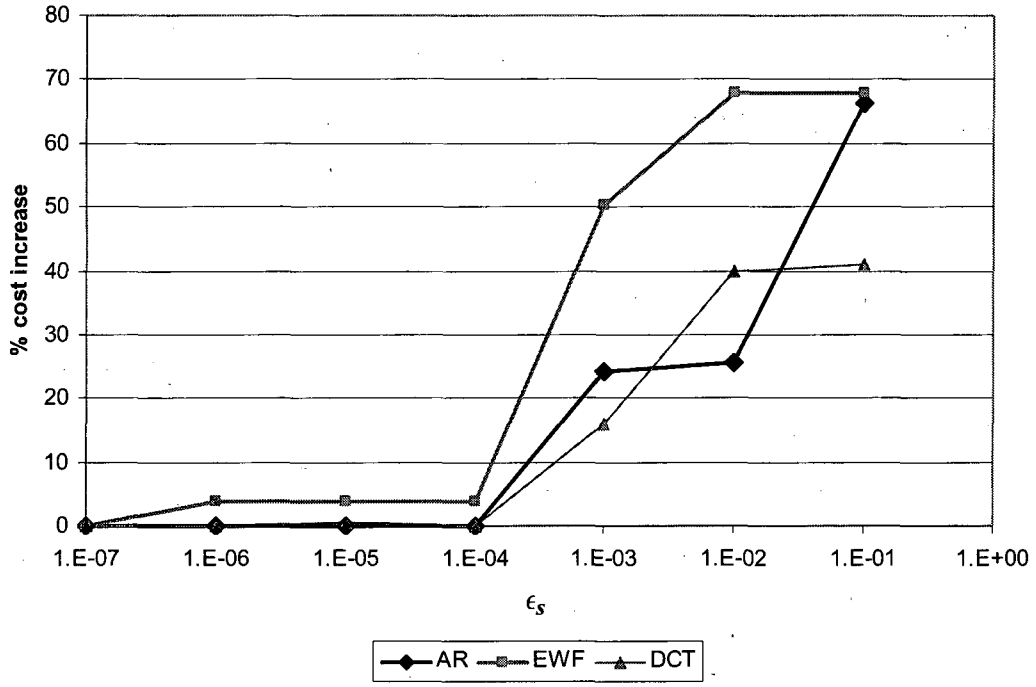


Figure 5.11 Cost increase in function of the parameter ϵ_s

5.4.4 Performance of the simulated annealing algorithm

This section investigates the performance of the algorithm by examining characteristic metrics such as the control parameter c and cost function during the simulated annealing process. Figure 5.12 shows the variation of the control parameter value when using the cooling schedule described in Section 5.4.2 during the synthesis of the autoregressive filter (AR) with $\alpha = 0.9$ and $\delta = 0.2$. Initially, at large values of the control parameter c, high increases in cost will be accepted. As c decreases, only small increases in cost will be accepted and finally, as c approaches to 0, only improvements in cost will be accepted. Consequently, the average values of the cost function calculated using equation (5.1) and the standard deviation *stdev* from this value are relatively high at large values of c, as shown in Figure 5.13. Note that both values decrease until the cost function becomes steady and the standard deviation approximates to zero. The variation of the standard deviation provides information about the annealing process and is used to calculate the next control parameter value using equation (5.8).

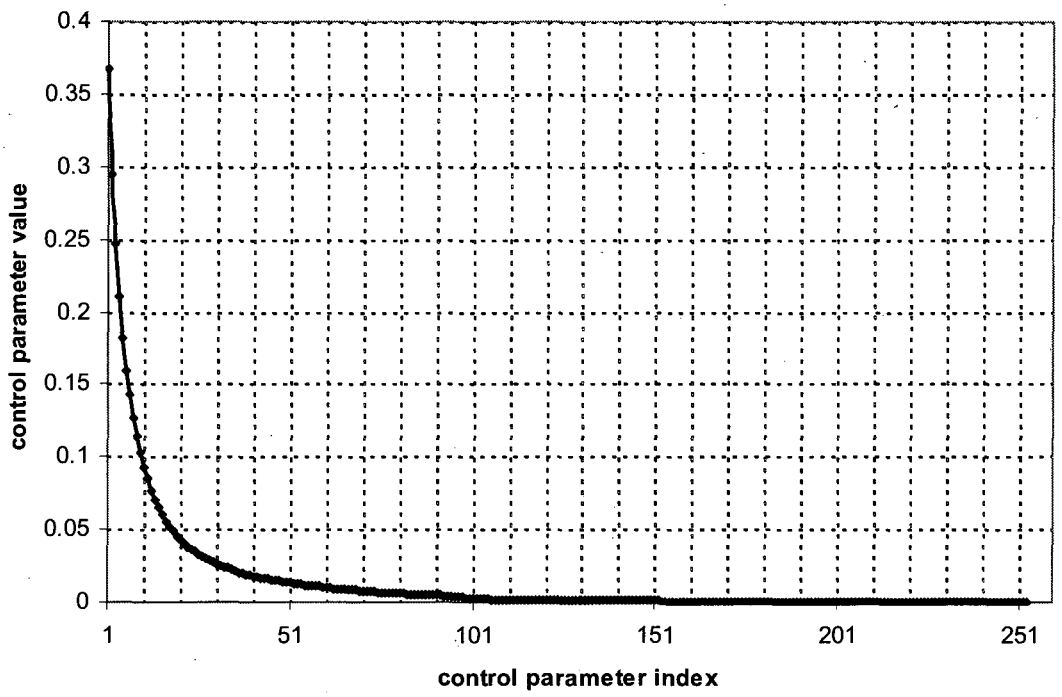


Figure 5.12 Variation of the control parameter during an annealing cycle

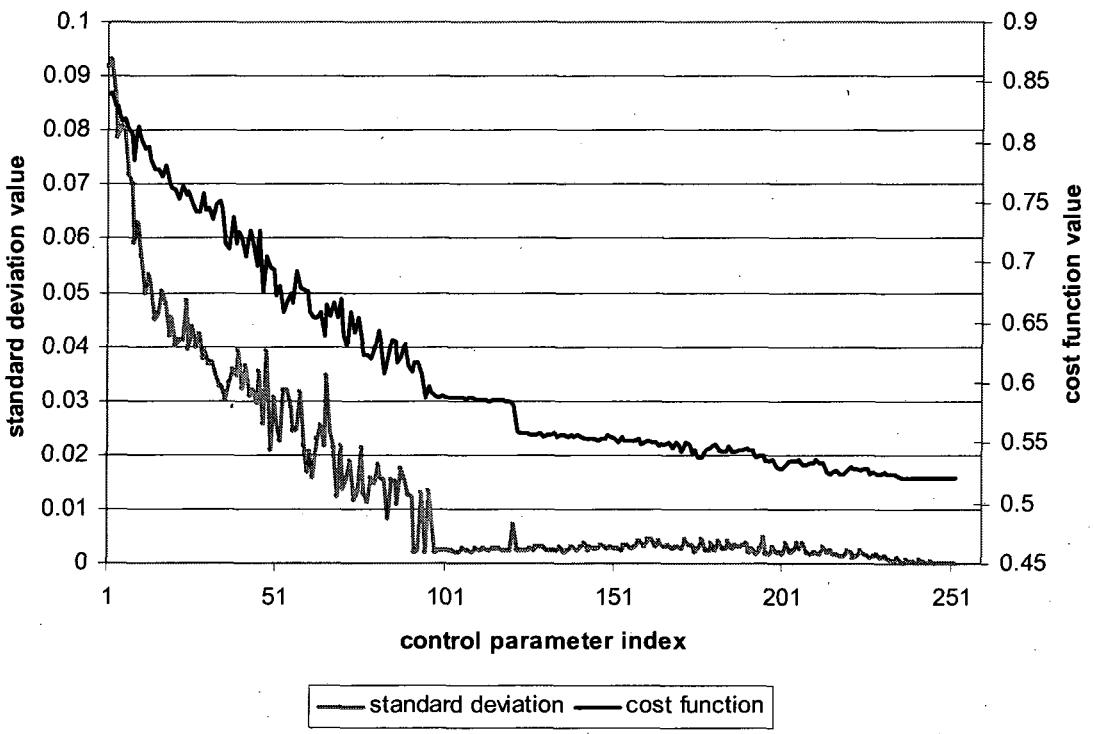


Figure 5.13 Variation of the standard deviation and cost function during an annealing cycle

To give a better insight into the behaviour of the cost function in Figure 5.13, consider Figure 5.14 and Figure 5.15, which show respectively the power and area cost of the datapath components during subsequent control parameter values. From

Figure 5.14 it can be seen that the modules power cost is rapidly reduced and becomes steady below 2.2mW. Similar behaviour can be seen for the registers power cost, which becomes constant around a value of 220μW. The module and register power cost become rapidly steady because the algorithm found a low voltage that may satisfy the optimisation goal. Unlike the modules and registers, the power cost of the multiplexers increases during the first half and decreases later until becoming steady towards the end of the annealing cycle. This is because although a low voltage has been found, the number and type of multiplexers change due to module and register sharing, hence changing the power consumption.

From Figure 5.15 it can be seen that during the first 100 control parameter values the modules area is gradually reduced while the multiplexers area is being increased. This may be due to higher multiplexers requirement to increase the module sharing and decrease the modules area. Then the registers and multiplexers area start decreasing slightly while the modules area remains constant until the 125th control parameter, where a significant reduction, i.e. 1 multiplier less, can be noticed. After this, the modules area continues decreasing due to the reduction of adders until it becomes constant after the 230th control parameter value. Meanwhile, the registers and multiplexers area experience a gradually decrease as of the 125th control parameter value, becoming steady at the end of the annealing cycle.

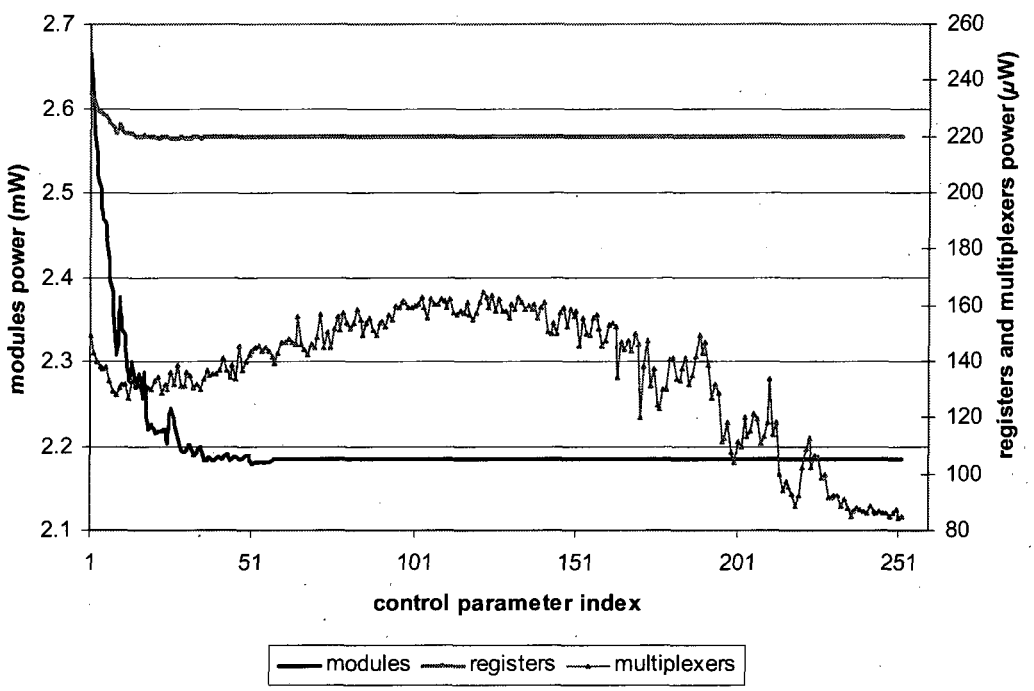


Figure 5.14 Variation of the power cost of the datapath components during an annealing cycle

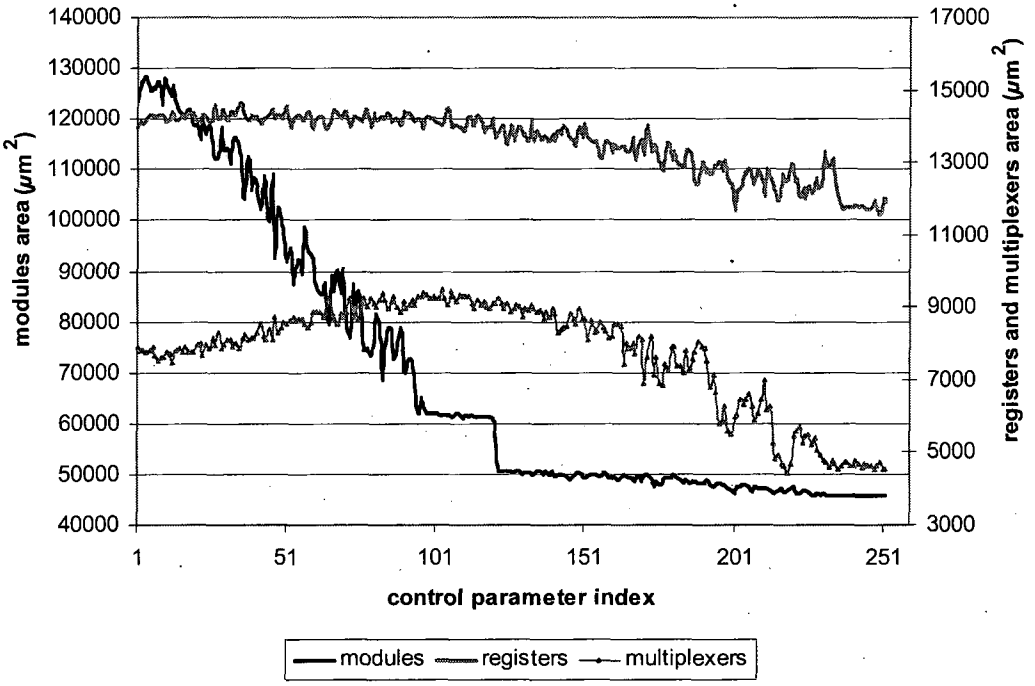


Figure 5.15 Variation of the area cost of the datapath components during an annealing cycle

5.5 Experimental results

To verify the efficiency of the power-aware datapath synthesis algorithm, three benchmarks have been used: autoregressive filter (AR), elliptical wave filter (EWF) and discrete cosine transform (DCT). Experiments with these benchmarks have been conducted on a Pentium 4, 2.2GHz, 1 GB RAM under different time constraints corresponding to 1.5, 2, 2.5, 3 and 3.5 times the critical path (cp). The annealing parameters χ_0, δ and ϵ_s, have been set respectively to 0.95, 0.2 and 0.0001 according to Section 5.4.3. The maximum allowed frequency *max_f* has been set arbitrarily to 500MHz and the library components have been characterized for power P, area A and delay D as shown in Table 5.9. More details about the library characterisation can be found in Appendix 2. In addition to the data presented in Table 5.9, the library also contains information about the power variation when changing the throughput of the multiplier and adder as shown in Table 5.10 and Table 5.11 respectively. In the header of these tables, P_1 represents the power of the component with throughput of 1 cstep, P_3 represents the power of the component with throughput of 3 csteps and P_5 represents the power of the component with throughput of 5 csteps. The 16-bit library used during these experiments consists of: Wallace multiplier, CLA adder, register and diverse multiplexers. The area count and delay of the library components

were obtained after logic synthesis using Synplify ASIC from Synplicity and ST 0.12μm technology. Power values at 90MHz were obtained using PrimePower from Synopsys and experimentally averaged over a number of pseudo-random input vectors obtained with a Linear Shift Feedback Register (LSFR).

Table 5.9 0.12μm library components

MULTIPLIER, $A = 10661\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	1237	11.05
1.2	1870	7.14
1.32	2556	4.65
ADDER, $A = 1107\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	40	4.29
1.2	49	2.86
1.32	66	1.98
REGISTER, $A = 549\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	47	0.55
1.2	55	0.34
1.32	77	0.23
MUX2, $A = 258\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	19	0.14
1.2	23	0.09
1.32	31	0.06
MUX3, $A = 549\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	28	0.37
1.2	34	0.23
1.32	46	0.15
MUX4, $A = 549\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	28	0.37
1.2	34	0.23
1.32	46	0.15
MUX5, $A = 807\mu m^2$		
V(V)	P_{dyn}(μW)	D(ns)
1.08	50	0.55

1.2	60	0.35
1.32	82	0.23
MUX6, $A = 1065\mu m^2$		
V(V)	$P_{dyn}(\mu W)$	D(ns)
1.08	72	0.55
1.2	86	0.35
1.32	117	0.23
MUX7, $A = 1356\mu m^2$		
V(V)	$P_{dyn}(\mu W)$	D(ns)
1.08	81	0.55
1.2	98	0.35
1.32	133	0.23
MUX8, $A = 1356\mu m^2$		
V(V)	$P_{dyn}(\mu W)$	D(ns)
1.08	81	0.55
1.2	98	0.35
1.32	133	0.23

Table 5.10 Power consumption of the multiplier with different supply voltages and throughputs

	P_1 (μW)	P_3 (μW)	P_5 (μW)
1.08V	1237	1239	1242
1.2V	1870	1874	1876
1.32V	2556	2560	2564

Table 5.11 Power consumption of the adder with different supply voltages and throughputs

	P_1 (μW)	P_3 (μW)	P_5 (μW)
1.08V	40	43	45
1.2V	49	52	55
1.32V	66	70	75

5.5.1 Power-area tradeoffs

The aim of this experiment is to demonstrate that given a time constraint, PABCOM is capable of obtaining good quality solutions in terms of power and area according to the power optimisation weight α specified by the designer.

EWF

The solutions obtained for EWF with different time constraints and optimisation goals are shown in Figure 5.16.

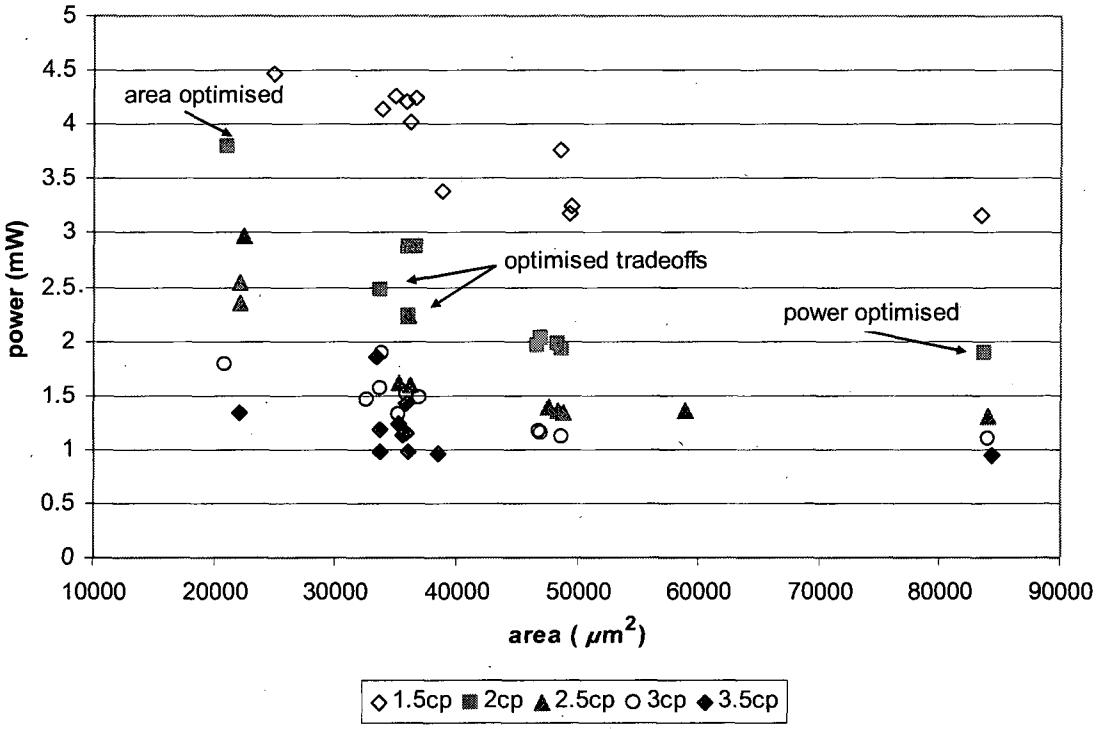


Figure 5.16 Solutions for EWF using different optimisation goals

It can be seen that for each time constraint, PABCOM is capable of obtaining solutions with different area and power requirements. This range of solutions is possible due to the compound cost function of PABCOM (see Section 5.4.1), which carefully explores the solution space through the specification of the power weight α. For example, Table 5.12 shows the solutions obtained for a time constraint of two times the critical path (2cp) when varying α from 0 to 1 in steps of 0.1. It can be seen that setting $\alpha = 0$ results in an area optimised solution whereas setting $\alpha = 1$ results in a power optimised solution. The area optimised and power optimised solutions are shown in Figure 5.16 to better illustrate the different power and area requirements.

The area optimised and power optimised solutions have been shaded in Table 5.12. Note that these solutions use different schedule length Ls, multiplication throughput TP_m and addition throughput TP_a, resulting in different operating voltages. For example, the clock period and operations throughput selected for $\alpha = 0$ allow scaling the voltage to 1.29V, leading to a power consumption of 3.8mW, while for $\alpha = 1$ the operating voltage is 1.13V with a power consumption of 1.9mW. It can also be noted that different values of clock and operations throughput lead to different area implementations. For example, for $\alpha = 0$, $Ls = 31$, $TP_m = 2$ and $TP_a = 1$, the datapath requires 1 multiplier (*), 2 adders (+), 13 registers (r) and 4 multiplexers (x), leading

to an area of 21042μm^2. For $\alpha = 1$, $Ls = 17$, $TP_m = 2$ and $TP_a = 1$, the datapath needs 6*, 5+, 19r and 11x, leading to the highest area implementation, 83639μm^2. This is because the optimisation goal targets only power since the area weight (1 - α) becomes zero, hence relaxing the area parameter. However, PABCOM is capable of obtaining optimised power-area tradeoffs with lower area than a power optimised solution and lower power than an area optimised solution, i.e. solutions with $\alpha = 0.3$ and $\alpha = 0.5$ in Table 5.12.

Table 5.12 Solutions for EWF with 2cp

α	*Tclk* (ns)	*Ls* (cs)	TP_m (cs)	TP_a (cs)	*	+	*r*	*x*	*V* (V)	*A* (μm^2)	*P* (mW)
0	2.9	31	2	1	1	2	13	4	1.29	21042	3.8
0.1	6.5	14	1	1	2	3	13	11	1.26	36651	2.9
0.2	6.5	14	1	1	2	3	12	13	1.26	36038	2.9
0.3	4.8	19	2	1	2	2	13	8	1.16	33607	2.5
0.4	6.5	14	1	1	2	3	13	10	1.26	36102	2.9
0.5	4.6	20	2	1	2	3	12	12	1.17	36070	2.2
0.6	2.6	35	4	2	3	3	13	11	1.14	47021	2.0
0.7	5.4	17	2	1	3	3	13	11	1.13	46731	2.0
0.8	2.6	35	4	2	3	4	14	12	1.14	48355	2.0
0.9	5.4	17	2	1	3	4	14	13	1.13	48613	1.9
1	5.4	17	2	1	6	5	19	11	1.13	83639	1.9

cs: abbreviation of csteps

Optimised power-area tradeoffs for other time constraints are shown in Figure 5.17, where power and area ratios are normalised with respect to power and area values of $\alpha = 1$. It should be noted that a power or area ratio greater than 1 means an increase whereas a ratio lower than 1 means a decrease with respect to $\alpha = 1$. From Figure 5.17 it can be seen that using $\alpha = 0.9$ for time constraints ranging from 1.5 to 3 times the critical path (1.5cp to 3cp) reduces the area of the datapath in 40% and increases its power consumption in 2% approximately. For a time constraint 3.5cp and $\alpha = 0.9$, the area reduction is of 55% with a power increase of 2%. It can also be seen that further area reductions are possible but at the expense of a larger power increase. For example, for 2.5cp with $\alpha = 0.5$, an area reduction of 60% is possible with a power overhead of 20%. For the same time constraint with $\alpha = 0.2$, power is increased in 80% with an area reduction of 75%.

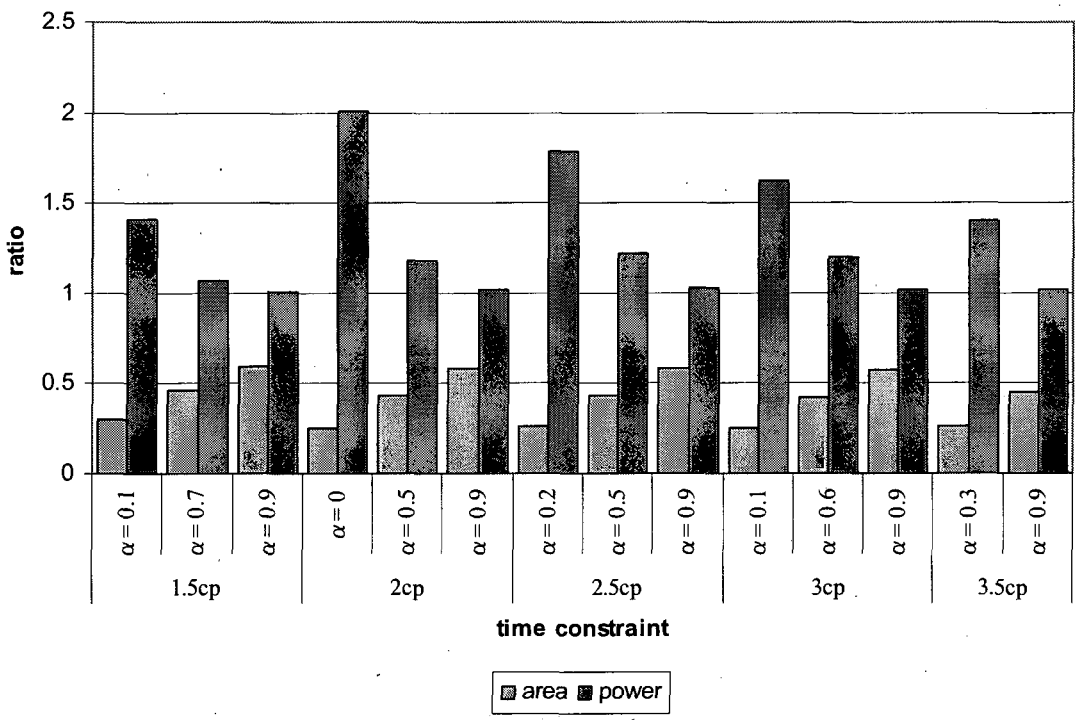


Figure 5.17 Power area tradeoffs for EWF

DCT

Figure 5.18 illustrate the range of solutions obtained for DCT under different time constraints. It can be seen that as previously shown for EWF, solutions with different area and power requirements can be obtained for the same time constraint. For example, for a time constraint 2.5cp, an area optimised solution has 53787μm^2 with a power consumption of 11mW, whereas a power optimised solution dissipates 5.8mW with an area of 182500μm^2. The optimised power-area tradeoffs present lower area than the optimised power solution and lower power than the area optimised solution, i.e. an area of 86028μm^2 with a power consumption of 6.9mW. The different area and power requirements of the solutions are due to the use of different power weight α which leads to a different scheduling and binding. For example, Figure 5.19 and Figure 5.20 show respectively the schedules obtained for $\alpha = 0.1$ and $\alpha = 0.4$ with a time constraint of 2.5cp. It can be seen that in both schedules the operations are single cycled, i.e. $TP_m = 1$ and $TP_a = 1$. Note that Figure 5.19 has a schedule length Ls of 8 csteps whereas Figure 5.20 has a schedule length of 6 csteps. The combination of these schedule lengths and operations throughput allowed scaling the voltage to 1.29V in the case of $\alpha = 0.1$ and to 1.21V in the case of $\alpha = 0.4$.

Consequently, a datapath power consumption of 10.8W and 8.7mW is obtained for α = 0.1 and α = 0.4 respectively, as shown later in Table 5.13.

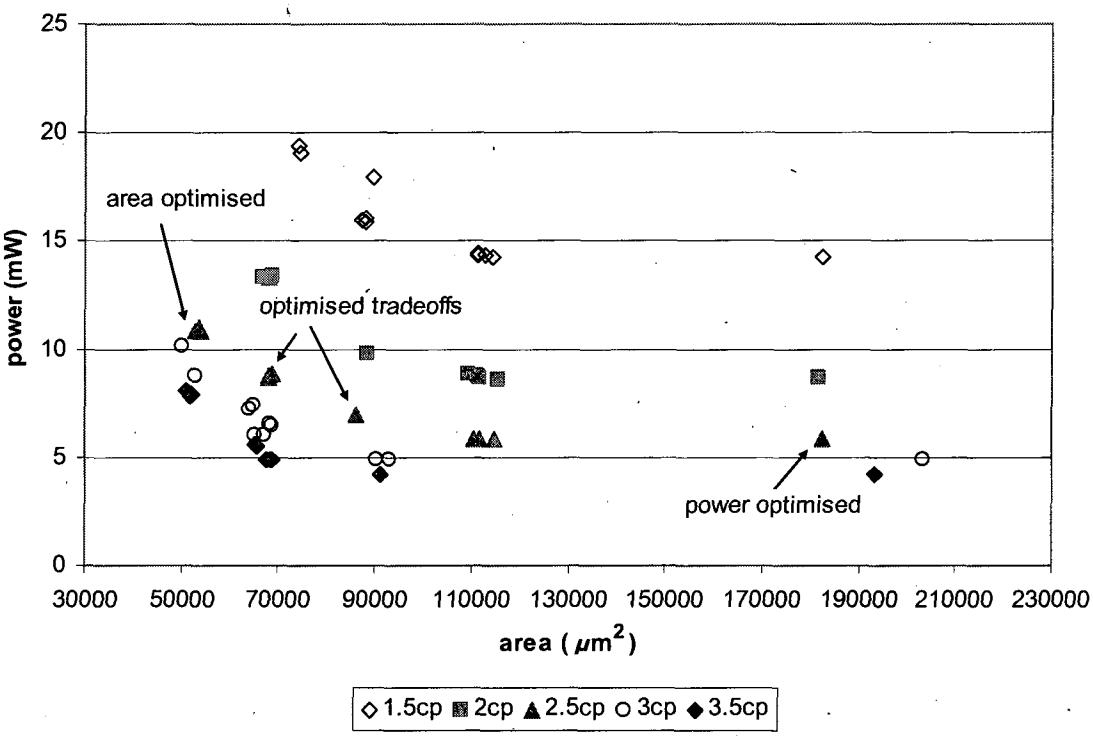


Figure 5.18 Solutions for DCT using different optimisation goals

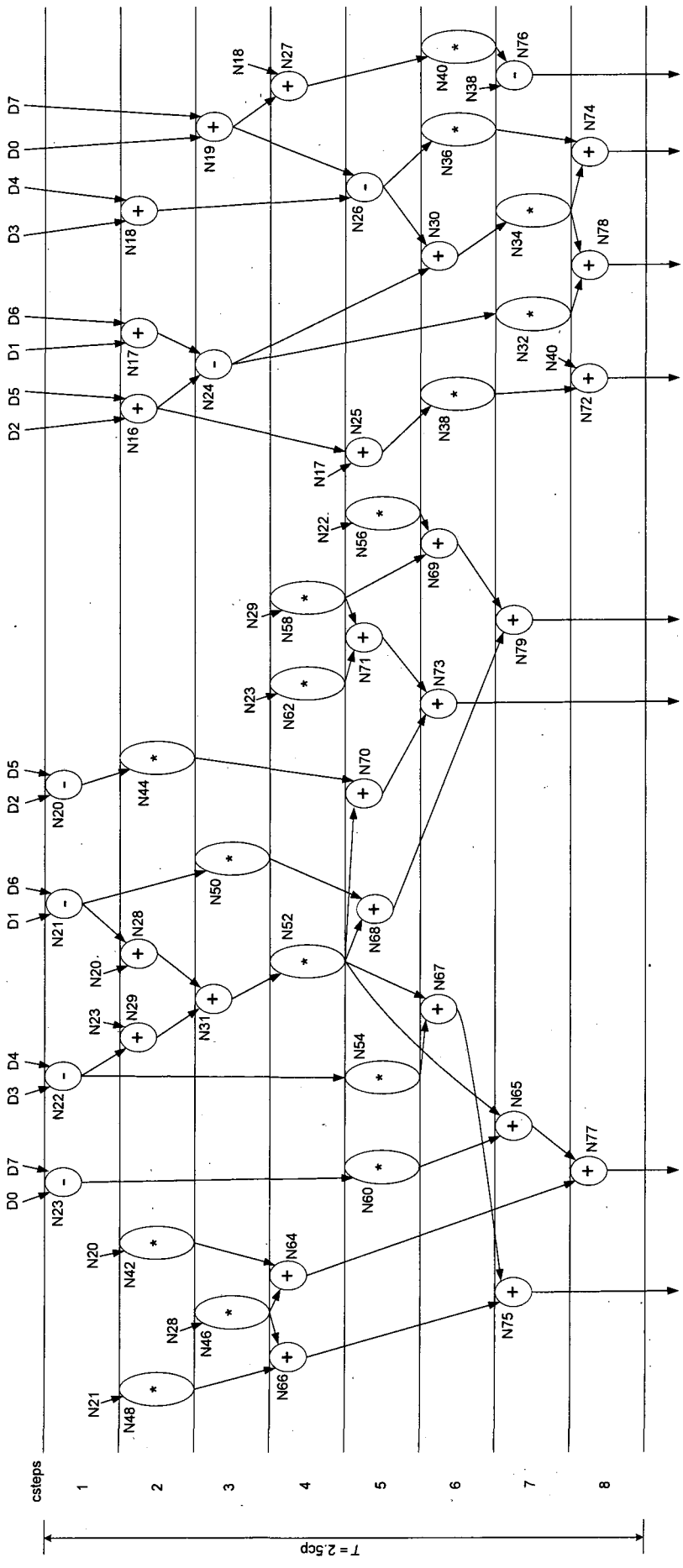


Figure 5.19 DCT schedule for $\alpha' = 0.1$

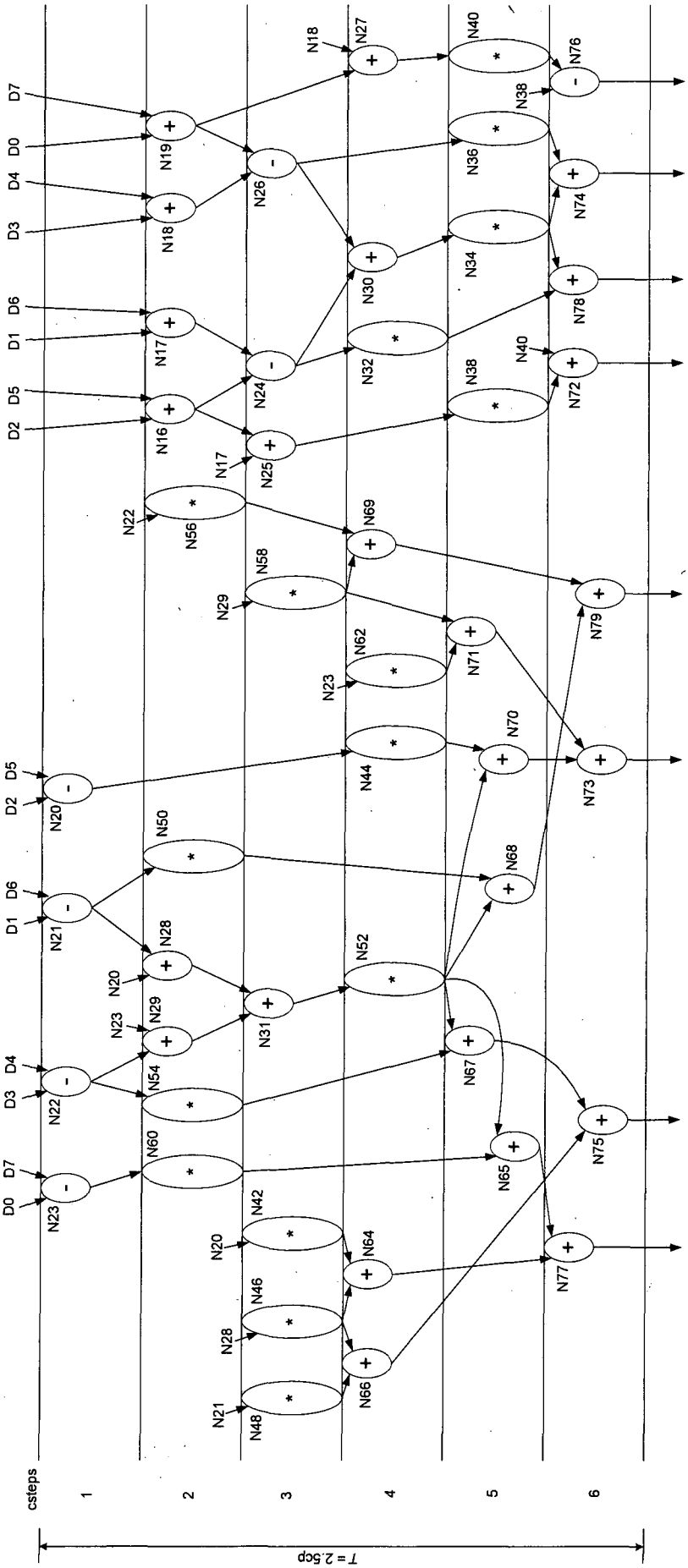


Figure 5.20 DCT schedule for $\alpha = 0.4$

The module bindings for the scheduled operations of Figure 5.19 and Figure 5.20 are given respectively in Figure 5.21 and Figure 5.22. Note that the binding for $\alpha = 0.1$ requires 8 modules whereas the binding for $\alpha = 0.4$ requires 12 modules. In Figure 5.21, the additions are bound to adders M1, M2, M3, M7 and M10, and the multiplications to multipliers M11, M13 and M15. In Figure 5.22, the additions are bound to adders M1, M2, M4, M5, M7, M8, M9 and M10, and the multiplications to multipliers M11, M12, M13 and M15.

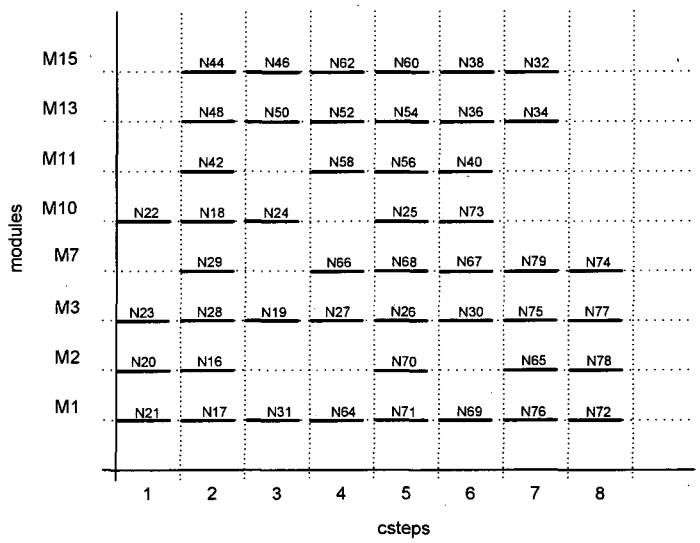


Figure 5.21 DCT module binding for $\alpha = 0.1$

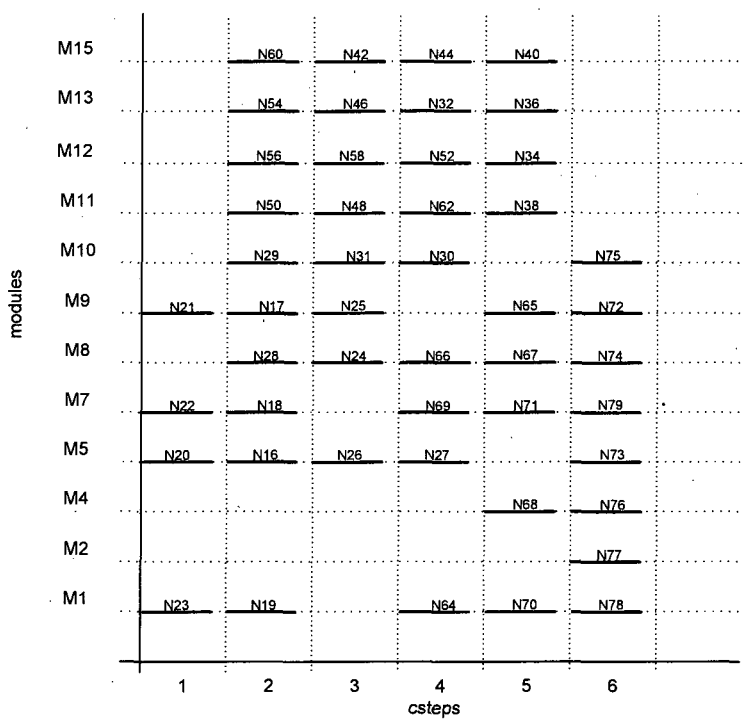


Figure 5.22 DCT module binding for $\alpha = 0.4$

The register bindings for the operations values of Figure 5.19 and Figure 5.20 are given respectively in Figure 5.23 and Figure 5.24. Note that the binding for $\alpha = 0.1$ (Figure 5.23) requires 15 registers whereas the binding for $\alpha = 0.4$ (Figure 5.24) requires 17 registers. Having finished the register and module binding, the complete datapath structures for $\alpha = 0.1$ and $\alpha = 0.4$ are shown in Figure 5.25 and Figure 5.26 respectively. In Figure 5.25, it can be seen that the datapath consist of 5 adders, 3 multipliers, 15 registers and 18 multiplexers. The datapath shown in Figure 5.26 consist of 8 adders, 4 multipliers, 17 registers and 22 multiplexers. Note that both datapaths use two groups of multiplexers. One group is connected to the inputs of the functional modules while other group is connected to the inputs of the registers. These multiplexers allow respectively the sharing of modules and registers.

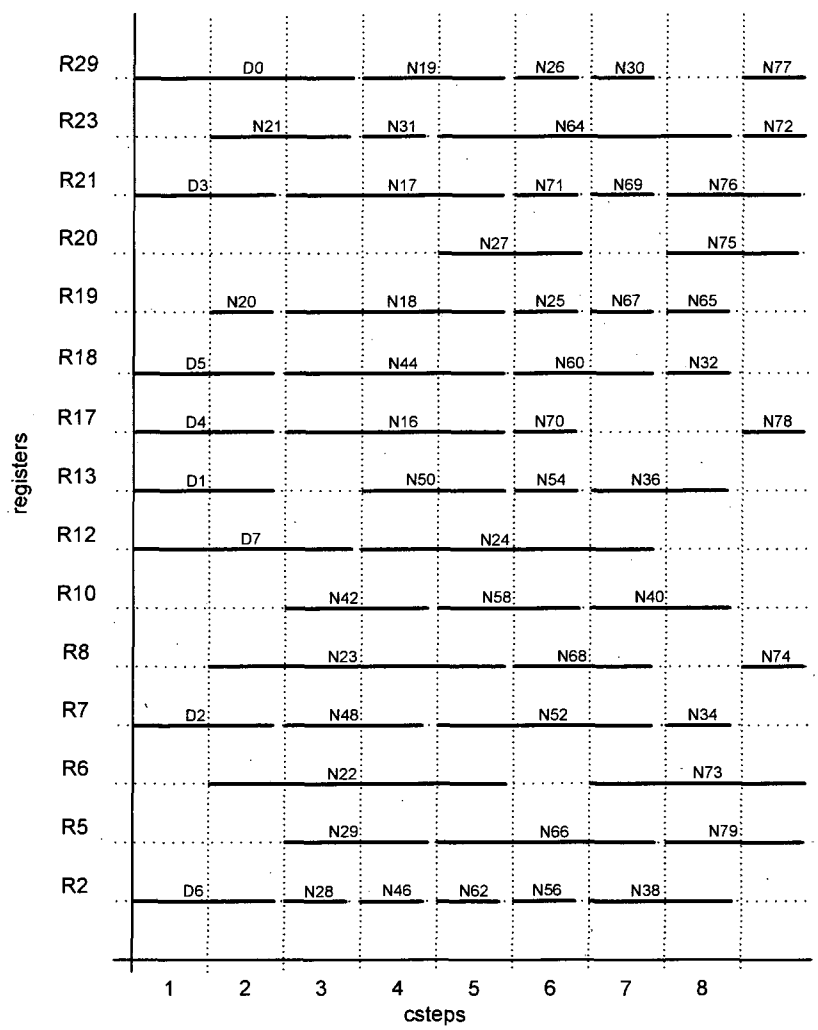


Figure 5.23 DCT register binding for $\alpha = 0.1$

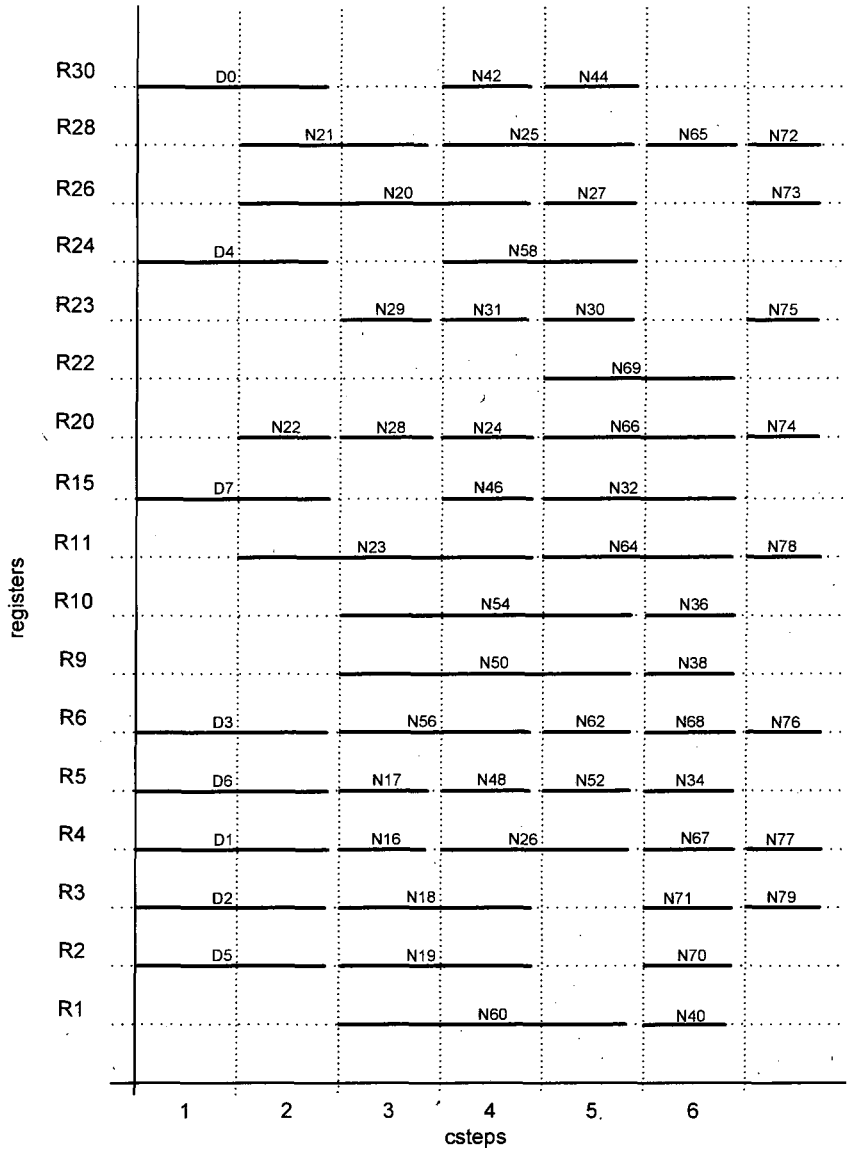


Figure 5.24 DCT register binding for $\alpha = 0.4$

The schedules (Figure 5.19 and 5.20), bindings (Figures 5.21, 5.22, 5.23 and 5.24) and datapath structures (Figures 5.25 and 5.26) shown for DCT with $\alpha = 0.1$ and $\alpha = 0.4$ were derived from the output file obtained by PABCOM. The output files generated by PABCOM for $\alpha = 0.1$ and $\alpha = 0.4$ are presented in Appendix 3.

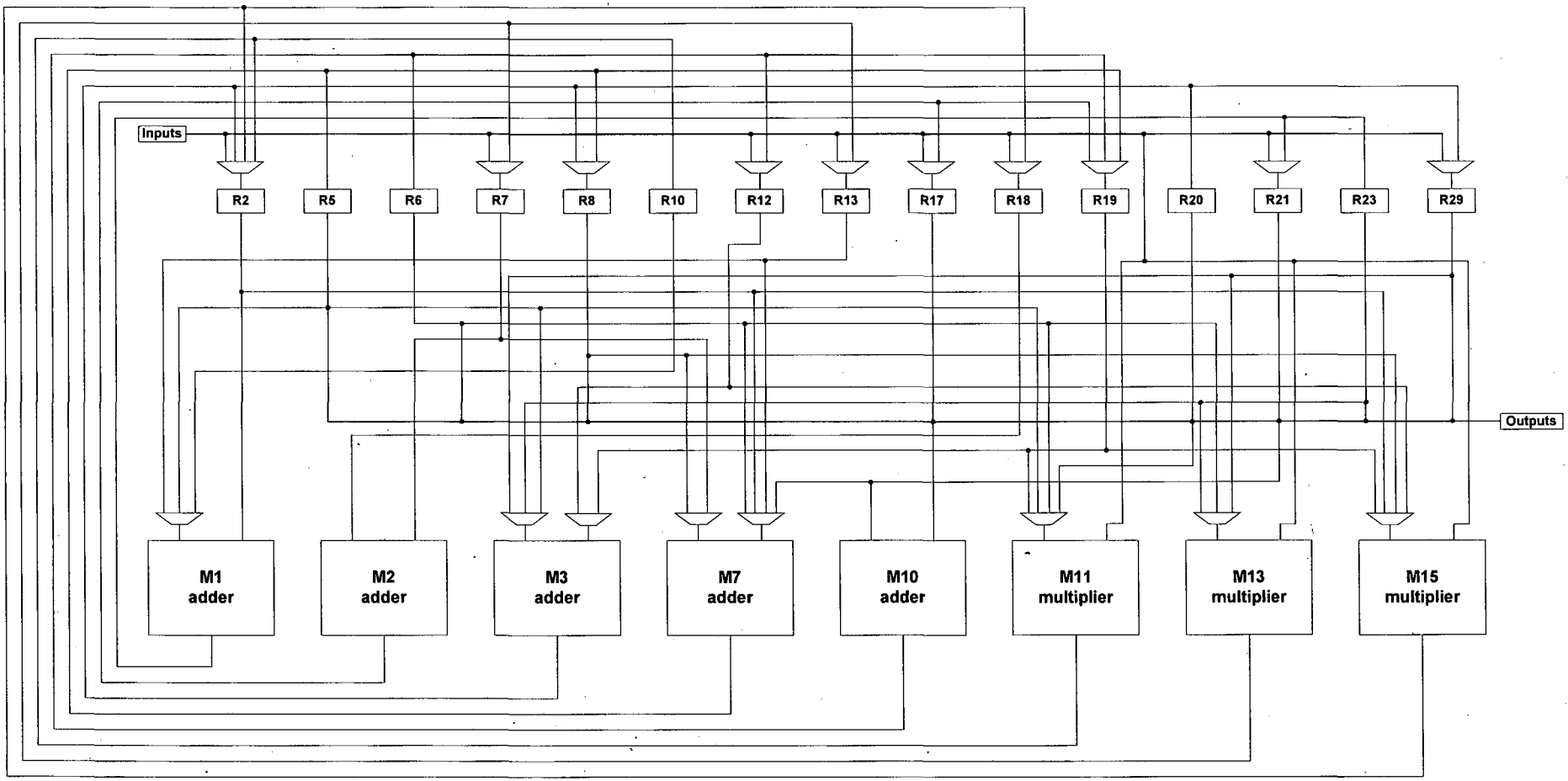


Figure 5.25 DCT datapath for $\alpha = 0.1$

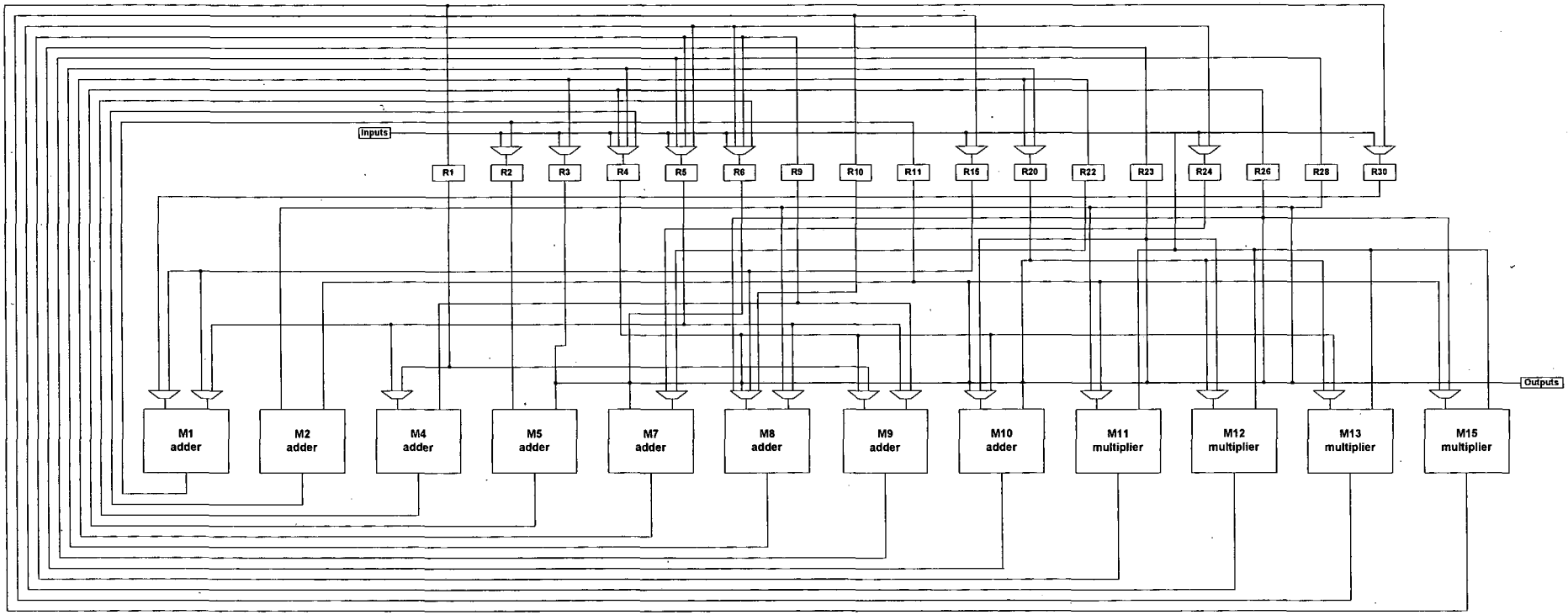


Figure 5.26 DCT datapath for $\alpha = 0.4$

The main parameters of the solutions for $\alpha = 0.1$ and $\alpha = 0.4$ are summarised in Table 5.13. Note that for $\alpha = 0.1$ the datapath (Figure 5.25) has an area of 52722μm^2 with a power dissipation of 10.8mW, whereas for $\alpha = 0.4$ the datapath (Figure 5.26) presents an area of 67964μm^2 with a power consumption of 8.7mW. From Table 5.13, it can also be seen that $\alpha = 1$ provides the solution with minimum power, i.e. 5.8mW, and highest area implementation, i.e. 182500μm^2. However, as in the EWF case, PABCOM obtains optimised power-area tradeoffs with smaller area and slightly higher power values when compared to a power optimised solution. This is illustrated in Figure 5.27, where it can be seen that using $\alpha = 0.9$ for time constraints ranging from 2.5cp to 3.5cp reduces the area of the datapath in 52% average and increases its power consumption in 1% approximately. For time constraints 1.5cp and 2cp with $\alpha = 0.6$ and $\alpha = 0.8$ respectively, an area reduction of 3% is obtained with a power increase of 1%. It can also be seen that further area reductions are possible but at the expense of a larger power increase. For example, for 1.5cp with $\alpha = 0.5$, an area reduction of 24% is possible with a power overhead of 12%. For the same time constraint with $\alpha = 0.1$, the power is increased in 35% with an area reduction of 35%.

Table 5.13 Solutions for DCT with 2.5cp

α	*Tclk* (ns)	*Ls* (cs)	TP_m (cs)	Tp_a (cs)	*	+	*r*	*x*	*V* (V)	*A* (μm^2)	*P* (mW)
0	5.9	8	1	1	3	5	15	21	1.29	53787	11.0
0.1	5.9	8	1	1	3	5	15	18	1.29	52722	10.8
0.2	5.9	8	1	1	3	5	17	22	1.29	53981	10.8
0.3	7.8	6	1	1	4	8	16	20	1.21	68642	8.8
0.4	7.8	6	1	1	4	8	17	22	1.21	67964	8.7
0.5	7.8	6	1	1	4	8	17	20	1.21	67738	8.7
0.6	5.2	9	2	1	6	5	18	19	1.13	86028	6.9
0.7	6.7	7	2	1	8	8	18	20	1.08	110349	5.8
0.8	6.7	7	2	1	8	8	20	21	1.08	111705	5.8
0.9	6.7	7	2	1	8	10	21	24	1.08	114662	5.8
1	6.7	7	2	1	14	10	29	22	1.08	182500	5.8

cs: abbreviation of csteps

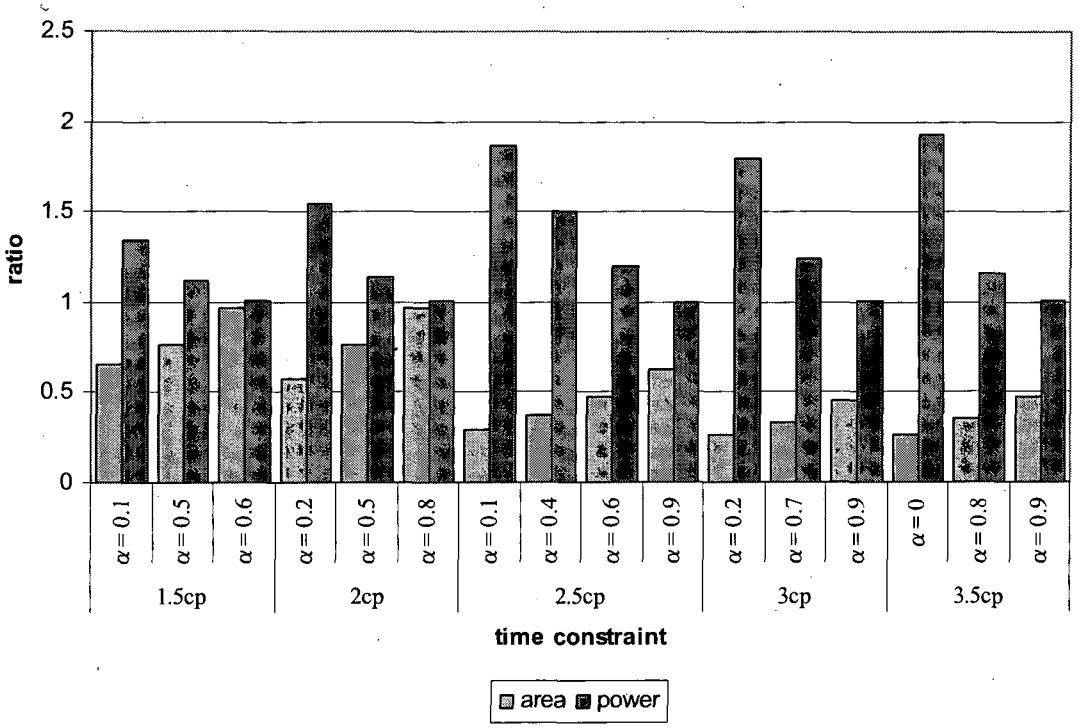


Figure 5.27 Power area tradeoffs for DCT

5.5.2 Comparison with a power-aware base case

From the previous work on low power behavioural synthesis, the most relevant to our work is [110]. The behavioural system developed in [110] performs scheduling, allocation and binding, and includes supply voltage and clock period pruning techniques to eliminate inferior design points when searching for the minimum power solution. Although [110] has performed effective power reduction, the algorithm focuses only on obtaining the minimum power solution and has some shortcomings. For example, the optimum power solution is chosen after synthesising a datapath for each combination of supply voltage and clock period that could not be pruned, leading to high computational times, as explained later in Table 5.14. Moreover, the optimization cost function ignores the area cost, which may lead to unnecessary big area implementations, as shown later in Table 5.18.

This section demonstrates the benefits of PABCOM when compared with a power-aware base case. The base case is a modified version of PABCOM that includes the clock period and supply voltage pruning techniques from [110] instead of the algorithm developed in Section 5.2 as part of this research. To find the minimum power solution, the base case follows the same methodology that [110], which enters

the synthesis phase for all the values of clock period and supply voltage that could not be pruned. As the single objective algorithm developed in [110], the base case targets only power as the cost function and the metric for evaluating moves. Consequently, the power weight α is set to 1 in PABCOM and the base case to provide a fair comparison. Table 5.14 shows the improvement percentages in terms of power P and computational time Ct, for AR, EWF and DCT, with a time constraint T ranging from 1.5cp to 3cp. It should be noted that a % saving greater than 0 means a decrease whereas a % saving lower than 0 means an increase with respect to the base case.

Table 5.14 Power and time savings when comparing PABCOM and base case

	AR		EWF		DCT	
T	*%P*	*%Ct*	*%P*	*%Ct*	*%P*	*%Ct*
1.5cp	5.2	84.4	4.2	85.2	6.1	84.1
2cp	4.8	91.7	4.2	91.4	5.8	85.6
2.5cp	0.3	93.9	0.0	94.1	0.0	90.2
3cp	0.1	95.9	0.1	95.5	0.3	94.4

From Table 5.14, it can be seen that for most of the time constraints, PABCOM is able to further reduce the power consumption for the three benchmarks. For example, for T = 1.5cp, power was decreased by 5.2% for AR, whereas for EWF and DCT there was a power reduction of 4.2% and 6.1% respectively. These power savings are due to a lower operating voltage as shown later in Tables 5.15, 5.16 and 5.17. From Table 5.14 it can also be seen that PABCOM obtained optimised power solutions in much less time than the base case, with computational time savings greater than 80% in general. For example, for AR at 1.5cp, PABCOM obtains a solution in 1.5min whereas the base case takes 9.3min. The computational time savings are due to the inclusion of clock period and operations throughput selection algorithm that determines the scaled supply voltage into the datapath synthesis. This is unlike the base case, where the synthesis phase is performed for all the combinations of clock period and voltage that could not be pruned.

To give a better insight into the achieved power savings from Table 5.14, the solutions obtained by PABCOM and the base case for AR, EWF and DCT with different time constraints are shown in Table 5.15, Table 5.16 and Table 5.17 respectively. A general observation from these three tables is that for the first two time constraints, 1.5cp and 2cp, PABCOM obtains lower supply voltages, thus

reducing the power when compared to the base case. For example, for AR with time constraint 2cp in Table 5.15, PABCOM decreases the supply voltage from 1.15V to 1.13V, reducing the power consumption from 5.3mW to 5.0mW. In the case of EWF, PABCOM obtains not only a lower voltage but also longer clock periods and smaller throughputs than the base case, resulting in further power reduction. For example, for time constraint 1.5cp in Table 5.16, *Tclk* has been increased from 2ns to 4ns, TP_m has decreased from 4 to 2 and TP_a has decreased from 2 to 1. Consequently, V has decreased from 1.22V to 1.21V, leading to a power reduction from 3.3mW to 3.1mW. Another general observation from Table 5.15, Table 5.16 and Table 5.17, is that for the time constraints 2.5cp and 3cp, PABCOM and the base case obtain the minimum voltage allowed, i.e. 1.08V. This results in similar power consumption, as shown in Table 5.17 for DCT with time constraint 2.5cp, where power consumption of 5.8mW has been obtained.

Unlike the single objective base case that only targets power minimisation, PABCOM uses a compound cost function that allows investigating explicitly the power-area tradeoffs of the design and constraining the area when searching for a low power solution. This may result in substantial area reductions with very few or null power overhead, as demonstrated in Table 5.18, which shows the comparison between PABCOM with $\alpha = 0.9$ and the single objective base case with $\alpha = 1$. It can be seen that for AR, EWF and DCT, PABCOM still obtains power savings around 4% for the time constraints 1.5cp and 2cp. For the remaining time constraints the power has an approximated increase of 2% for AR and EWF, and 1% for DCT. However, the datapath area has been reduced significantly for all the cases. For example, average area savings of 58%, 42% and 43% have been obtained for AR, EWF and DCT respectively.

Table 5.15 AR solutions

	Base case					PABCOM				
T	*Tclk* (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (mW)	*Tclk* (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (mW)
1.5cp	4	2	1	1.23	9.0	4	2	1	1.21	8.4
2cp	5.4	2	1	1.15	5.3	5.4	2	1	1.13	5.0
2.5cp	6.6	2	1	1.08	3.4	6.6	2	1	1.08	3.4
3cp	6.8	2	1	1.08	2.8	8	2	1	1.08	2.8

cs: abbreviation of csteps

Table 5.16 EWF solutions

	Base case					PABCOM				
T	$Tclk$ (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (μW)	$Tclk$ (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (μW)
1.5cp	2	4	2	1.22	3.3	4	2	1	1.21	3.1
2cp	2.4	5	2	1.14	2.0	5.4	2	1	1.13	1.9
2.5cp	6.7	2	1	1.08	1.3	6.7	2	1	1.08	1.3
3cp	6.2	3	1	1.08	1.1	8	2	1	1.08	1.1

cs: abbreviation of csteps

Table 5.17 DCT solutions

	Base case					PABCOM				
T	$Tclk$ (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (mW)	$Tclk$ (ns)	TP_m (cs)	Tp_a (cs)	V (V)	P (mW)
1.5cp	4	2	1	1.22	1.5	4	2	1	1.21	1.4
2cp	5.3	2	1	1.15	9.0	5.3	2	1	1.13	8.7
2.5cp	6.7	2	1	1.08	5.8	6.7	2	1	1.08	5.8
3cp	3.3	4	2	1.08	4.8	6.2	3	1	1.08	4.8

cs: abbreviation of csteps

Table 5.18 Power and area percentage savings

	AR		EWF		DCT	
T	$\%A$	$\%P$	$\%A$	$\%P$	$\%A$	$\%P$
1.5cp	60.2	4.4	41.2	4.8	40.7	4.5
2cp	58.4	3.9	41.9	3.3	36.6	4.6
2.5cp	56.2	-2.3	42.0	-2.1	43.9	-0.3
3cp	58.8	-2.5	42.1	-2.2	48.8	-1.9

5.6 Concluding Remarks

This chapter has presented a power-aware behavioural compiler that considers the close interrelation among scheduling, binding, clock and operations throughput selection. PABCOM with its compound cost function provides solutions not only optimised for low power or low area, but also facilitates the automatic exploration of power-area tradeoffs. For example, for DCT with a time constraint equal to 2.5 times the critical path, a tradeoff with 35% lower power than an area optimised solution and 53% less area than a power optimised solution was obtained. It has also been shown that PABCOM obtains solutions in lower computational time and with lower power than a base case algorithm that uses the clock and supply voltage pruning techniques from [110] whilst meeting the same time constraint. For example, for a time constraint equal to 2 times the critical path, a power saving of 5% with computational time saving of 89% averaged over AR, EWF and DCT were obtained.

Moreover, area savings can be achieved by PABCOM at the expense of a slight or null increase in power consumption when compared to the base case. For example, a power saving of 4.6% with area saving of 47% averaged over AR, EWF and DCT were obtained for a time constraint of 1.5 times the critical path. In general, power reductions were due to the use of lower voltages, or a combination of lower voltages and lower frequencies obtained after an appropriate selection of clock period and operations throughput. It has also been shown how the clock and operations throughput selection affects the schedule, binding and voltage applied leading to solutions with different area and power consumption. Extensive experimental results for typical behavioural synthesis benchmarks with different time constraints have shown that PABCOM is capable of achieving power and area savings.

Chapter 6

Case study: MPEG-1 Motion Vector Reconstructor

6.1 Introduction

Chapters 4 and 5 have presented algorithms for the generation of low power RTL structures from behavioural descriptions. To validate the practical applicability of these techniques, their application during the design of a motion vector reconstructor from the Berkeley MPEG-1 player [40] is considered in this chapter. A synthesisable RTL VHDL implementation of the motion vector reconstructor was presented in [34], where different parallelizing transformations were applied during the behavioural synthesis process. In this chapter, the possible power-area tradeoffs due to an adequate selection of clock period and operations throughput will be presented. The motion vector reconstructor was selected as a case study because it is more complex than the benchmarks used in Chapter 4 and 5. The motion vector reconstructor consists of 12 conditional statements, 8 multiplications, 10 additions, 9 subtractions, 13 comparisons and 2 shifts. The presence of conditional statements in the motion vector reconstructor results in a DFG with conditional branches, which lead to a more complex scheduling, allocation and binding problem. Other reason why the motion vector reconstructor was selected as a case study is that it belongs to an application domain (data dominated) targeted by the algorithms developed in this research. Furthermore, the motion vector reconstructor forms part of a real-life multimedia application, which can be used for example in TV quality video communications [133].

The principles of an MPEG-1 video decoder are outlined in Section 6.2. Section 6.3 describes the motion vector reconstructor DFG, library components, selection of optimisation parameters and the results obtained after low power behavioural

synthesis. Section 6.4 discusses the implementation in 0.12μm technology of two solutions of the motion vector reconstructor, including their Verilog code, functional validation, and area and power values. These values are based on the reports obtained after logic synthesis with Synplify ASIC from Synplicity (see Figure 6.15) and power analysis with PrimePower from Synopsys (see Figure 6.21). Functional validation is performed using ModelSim in combination with a C++ program (see Figure 6.16). Section 6.5 presents the conclusions of this chapter.

6.2 MPEG-1 background

The Moving Picture Experts Group or MPEG [42] developed video and audio encoding standards such as MPEG-1, MPEG-2, MPEG-3 and MPEG-4 [41]. MPEG-1 standard is intended for the storage of VHS-quality audio and video on CDROM [36] and consists of five parts [15]: system, video, audio, compliance testing and reference software. For compression purposes, video data can be represented as a sequence of digitised images or frames [100]. In MPEG-1, the frames can be encoded in three types: intra-frames (I-frames), predicted frames (P-frames) and bidirectional frames (B-frames). Figure 6.1 shows an example of the frame sequence for MPEG-1, where the edges indicate that the pointed frame has been constructed from previous or subsequent frames, called reference frames. For example, I-frames are encoded without reference to any other frames whereas P-frames are approximated using the preceding I- or P-frame as reference. B-frames are constructed using the nearest I- and P-, P- and P-, or P- and I-frame as references. Each frame can be represented as an array of macroblocks, where each macroblock corresponds to a 16 by 16 pixels area of the original frame.

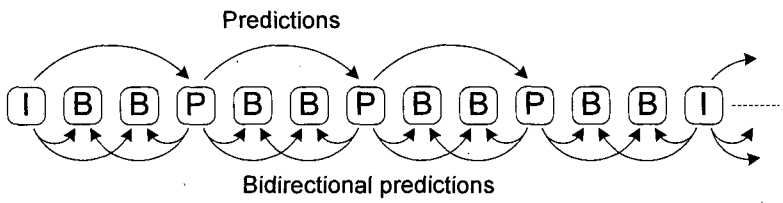


Figure 6.1 MPEG-1 example frame sequence [36]

MPEG-1 compresses video data applying motion estimation and motion compensation techniques to each macroblock in P- and B- frames. Decoding the

frames is simpler and faster than the encoding operation since it only requires the motion compensation technique. Motion compensation produces a predicted macroblock using the motion vector obtained during the compression process and the reference frame(s). The decoding scheme for the three types of frames is illustrated in Figure 6.2. The received bitstream of the I-frame is decoded employing the variable length and Huffman decoders. Then the obtained DCT coefficients are inverse quantised and IDCT is applied to generate the referenced frame, which is stored in memory. This referenced frame is used for motion compensation for both the P- and B- frames. To decode a P-frame it is necessary to generate the motion compensated P-frame by employing the referenced frame together with the reconstructed motion vectors. The quantised DCT coefficients obtained after variable length and Huffman coding are inverse quantised, and then IDCT is applied. The result is then added to the motion compensated P-frame to produce the referenced P-frame, which is also the decoded P-frame sent to display. To decode a B-frame it is necessary to generate the motion compensated B-frame by using the pair of nearest referenced frames and the reconstructed bidirectional motion vectors. The quantised DCT coefficients obtained after variable length and Huffman decoding are inverse quantised, and then IDCT is applied. The result is then added to the motion compensated B-frame to produce the decoded B-frame. Once the frame type (I-, P- or B-) has been reconstructed, a process called dithering is applied, which converts the frame to a representation appropriate for display.

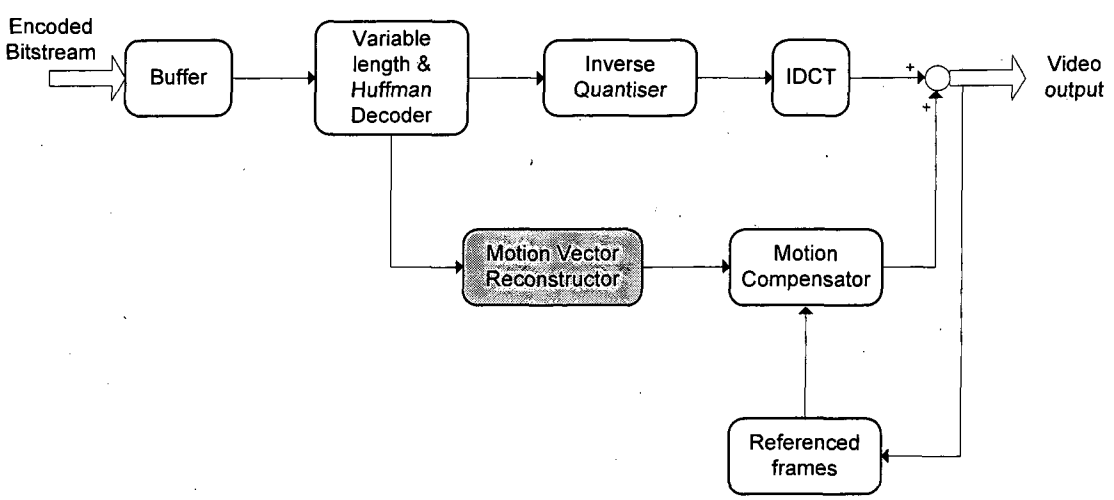


Figure 6.2 Block diagram of MPEG decoding [142]

6.3 Low power behavioural synthesis of a motion vector reconstructor

This section describes the steps carried out to obtain a low power datapath structure for the MPEG-1 motion vector reconstructor. Firstly the motion vector reconstructor is converted into a more suitable representation for behavioural synthesis, i.e. DFG. Then the synthesis constraints and library components are defined. To complete the set of inputs from PABCOM it is necessary to specify the optimisation parameters. Once all the inputs have been defined, PABCOM is able of generating low power solutions according to the designer requirements.

6.3.1 Motion Vector Reconstructor DFG

A publicly available C implementation of the MPEG-1 decoder was developed by the Berkeley Multimedia Research Centre [40]. The MPEG-1 player includes the C specification of the motion vector reconstructor according to the instructions given in the MPEG December 1991 standard draft. This C specification is not suitable for the power-aware synthesis tool developed in Chapter 5 and then manual conversion to a DFG was done. The resultant DFG of the motion vector reconstructor is shown in Figure 6.3. The black circles do not belong to the original C specification, but they were added to represent all operations as 2-input operations, hence, allowing the use of PABCOM. For example, N41 has rrprev_i as one of its inputs, but the other input can be either 0 or N25 or N26. By inserting NOP3 after the joint of branch 3, and NOP4 after the joint of branch 2, N41 can be represented as an operation with 2 inputs: rrprev_i and NOP4. Notice that the inputs for NOP3 are N25 and N26, and for NOP4 are 0 and NOP3, consequently the data dependency between the involved DFG operations is not affected. Since all the input data to PABCOM is through a text file, a DFG in a text form that exactly corresponds to the DFG shown in Figure 6.3 needs to be generated. This text representation of the DFG is shown in Listing 6.1, where three sections can be identified: inputs, operations and outputs. Each of these sections contains information about their components such as name, type, inputs, branch, and how they interrelate with forks and joints of the conditional branches. For example, from Listing 6.1 it can be seen that operation N12 is a comparison with inputs N7 (inp_a) and zero (inp_b), and a nested branch generated by the condition

N16 (nstcnd). Operation N16 is a comparison with inputs N7 (inp_a) and zero (inp_b), where value N7 comes from fork 7 (fk_i) when the condition corresponding to N12 (fk_own) is evaluated to false, i.e. logic zero. In the branch generated by N16, there is no nested branch (nstcnd). The output of operation N16 is not used as input to any fork (fk_o) or joint (jnt). Notice that operations NOP1, NOP2, ..., NOP16 are defined as type auxiliar, and as mentioned previously, do not affect the original data dependency of all the operations in the DFG.

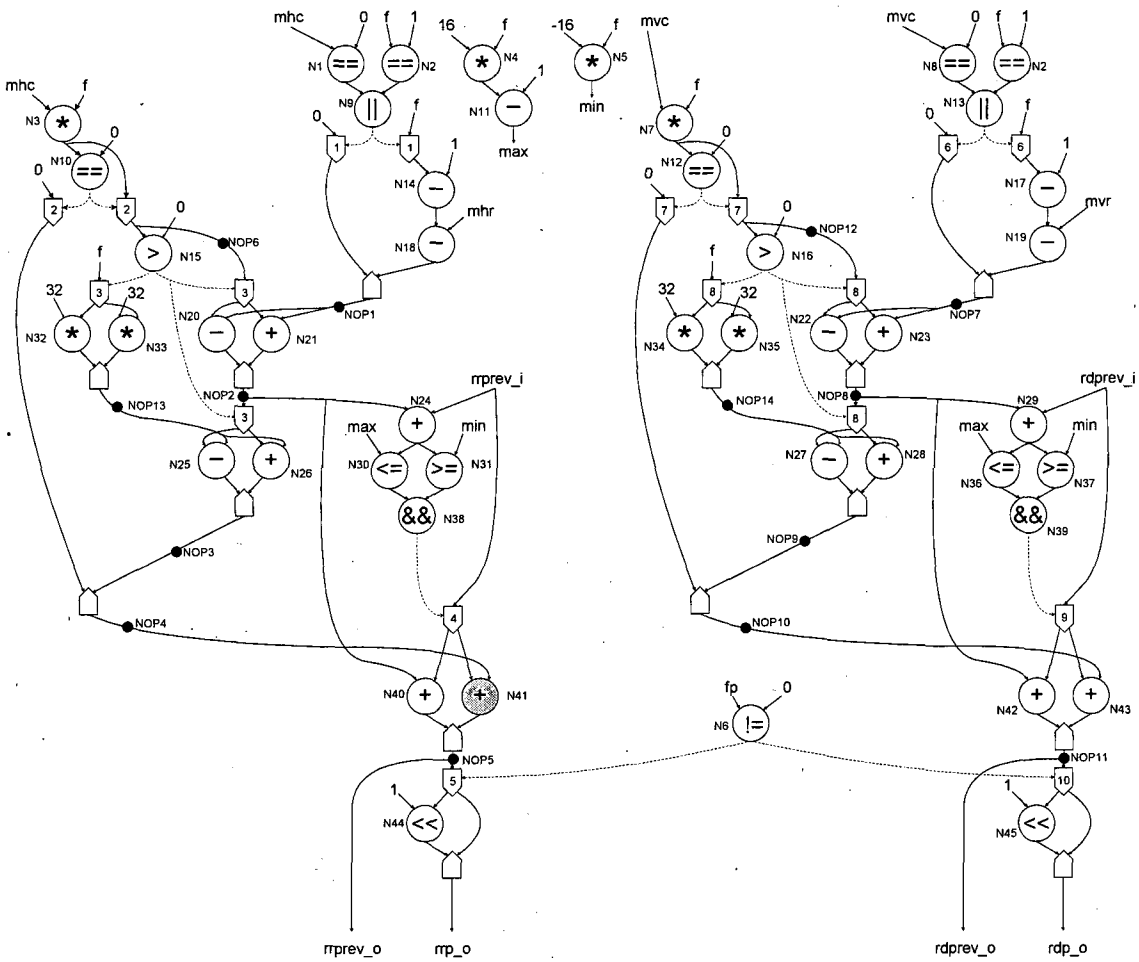


Figure 6.3 Motion vector reconstructor DFG

Listing 6.1 DFG of the motion vector reconstructor in text form

#name	type	inp_a	inp_b	fk_i	jnt	branch	fk_own	nstcnd	fk_o
#// inputs //									
mhc	input	-	-	0	0	-	-	-	-
mhr	input	-	-	0	0	-	-	-	-
mvc	input	-	-	0	0	-	-	-	-
mvr	input	-	-	0	0	-	-	-	-
f	input	-	-	0	0	-	-	-	1,6
fp	input	-	-	0	0	-	-	-	-
rrprev_i	input	-	-	0	0	-	-	-	4
rdprev_i	input	-	-	0	0	-	-	-	9
zero	input	-	-	0	0	-	-	-	1,2,6,7
1	input	-	-	0	0	-	-	-	-
16	input	-	-	0	0	-	-	-	-
(-16)	input	-	-	0	0	-	-	-	-
32	input	-	-	0	0	-	-	-	-
#// operations //									
N1	compare	mhc	zero	0	0	-	-	-	-
N2	compare	f	1	0	0	-	-	-	-
N3	multiply	mhc	f	0	0	-	-	-	2
N4	multiply	f	16	0	0	-	-	-	-
N5	multiply	f	(-16)	0	0	-	-	-	-
N6	compare	fp	zero	0	0	-	-	-	-
N7	multiply	mvc	f	0	0	-	-	-	7
N8	compare	mvc	zero	0	0	-	-	-	-
N9	or	N1	N2	0	0	-	-	-	-
N10	compare	N3	zero	0	0	-	-	N15	-
N11	subtract	N4	1	0	0	-	-	-	-
N12	compare	N7	zero	0	0	-	-	N16	-
N13	or	N8	N2	0	0	-	-	-	-
N14	subtract	f	1	1	0	1F	N9	-	-
N15	compare	N3	zero	2	0	2F	N10	-	-
NOP6	auxiliar	N3	-	2	0	2F	N10	-	3
N16	compare	N7	zero	7	0	7F	N12	-	-
NOP12	auxiliar	N7	-	7	0	7F	N12	-	8
N17	subtract	f	1	6	0	6F	N13	-	-
N18	subtract	N14	mhr	0	0	1F	-	-	-
NOP1	auxiliar	zero	N18	0	1	-	-	-	-
N19	subtract	N17	mvr	0	0	6F	-	-	-
NOP7	auxiliar	zero	N19	0	6	-	-	-	-
N20	subtract	NOP6	NOP1	3	0	2F3T	N15	-	-
N21	add	NOP6	NOP1	3	0	2F3F	N15	-	-
NOP2	auxiliar	N20	N21	0	3	2F	-	-	3
N32	multiply	32	f	3	0	2F3T	N15	-	-
N33	multiply	32	f	3	0	2F3F	N15	-	-
NOP13	auxiliar	N32	N33	0	3	2F	-	-	-
N22	subtract	NOP12	NOP7	8	0	7F8T	N16	-	-
N23	add	NOP12	NOP7	8	0	7F8F	N16	-	-
NOP8	auxiliar	N22	N23	0	8	7F	-	-	8
N34	multiply	32	f	8	0	7F8T	N16	-	-
N35	multiply	32	f	8	0	7F8F	N16	-	-
NOP14	auxiliar	N34	N35	0	8	7F	-	-	-
N24	add	rrprev	NOP2	0	0	-	-	-	-
N25	subtract	NOP2	NOP13	3	0	2F3T	N15	-	-

N26	add	NOP2	NOP13	3	0	2F3F	N15	-	-
NOP3	auxiliar	N25	N26	0	3	2F	-	-	-
N29	add	rdprev	NOP8	0	0	-	-	-	-
N27	subtract	NOP8	NOP14	8	0	7F8T	N16	-	-
N28	add	NOP8	NOP14	8	0	7F8F	N16	-	-
NOP9	auxiliar	N27	N28	0	8	7F	-	-	-
N30	compare	N24	N11	0	0	-	-	-	-
N31	compare	N24	N5	0	0	-	-	-	-
N36	compare	N29	N11	0	0	-	-	-	-
N37	compare	N29	N5	0	0	-	-	-	-
N38	and	N30	N31	0	0	-	-	-	-
N39	and	N36	N37	0	0	-	-	-	-
NOP4	auxiliar	zero	NOP3	0	2	-	-	-	-
NOP10	auxiliar	zero	NOP9	0	7	-	-	-	-
N40	add	rrprev	NOP2	4	0	4T	N38	-	-
N41	add	rrprev	NOP4	4	0	4F	N38	-	-
NOP5	auxiliar	N40	N41	0	4	-	-	-	5
N42	add	rdprev	NOP8	9	0	9T	N39	-	-
N43	add	rdprev	NOP10	9	0	9F	N39	-	-
NOP11	auxiliar	N42	N43	0	9	-	-	-	10
N44	shift	NOP5	-	5	0	5T	N6	-	-
N45	shift	NOP11	-	10	0	10T	N6	-	-
// outputs //									
rrp_o	output	NOP5	N44	0	5	-	-	-	-
rdp_o	output	NOP11	N45	0	10	-	-	-	-
rrprev_o	output	NOP5	-	0	0	-	-	-	-
rdprev_o	output	NOP11	-	0	0	-	-	-	-

6.3.2 Synthesis constraints and library components

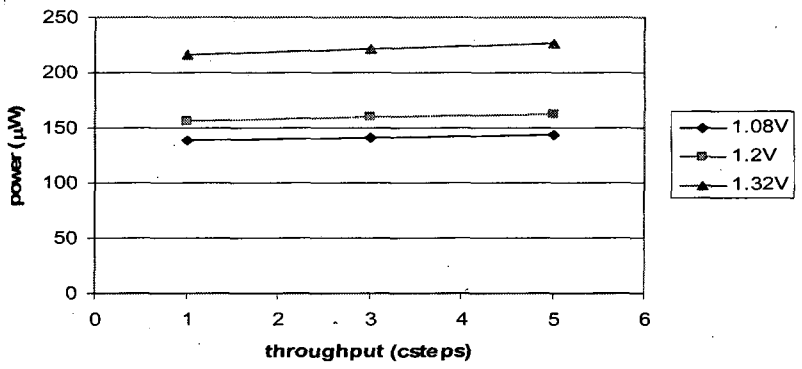
Section 6.3.1 has presented a suitable DFG of the motion vector reconstructor to be used as an input for the proposed algorithm developed in Chapter 5. In addition to the DFG from Listing 6.1, the proposed algorithm requires the specification of the time constraint T, the maximum allowed frequency f_{max} and a library component characterised for power, area and delay. The time constraint and maximum allowed frequency for the motion vector reconstructor design are set arbitrarily to 50ns and 500MHz respectively. The 16-bit library component consists of: multiplier, adder, subtractor, comparator, shifter, logic operations (and, or), register and multiplexers. The library was characterised for power and delay at three different voltages, 1.08V, 1.2V and 1.32V. These voltages are henceforth referred to as characterization voltages. Table 6.1 shows the library used, where A is the area, V is the characterisation voltage, $\dot{P}_{dyn}$ is the dynamic power and D is the delay. The area and delay of the library components were obtained after synthesising their VHDL descriptions, which are included in Appendix 1. The logic synthesis process was

performed using Synplify ASIC from Synplicity with ST 0.12μm technology library. Power values at 100MHz were obtained using PrimePower from Synopsys and experimentally averaged over a number of input vectors. Details about the power, area and delay characterization process of each library component can be found in Appendix 2.

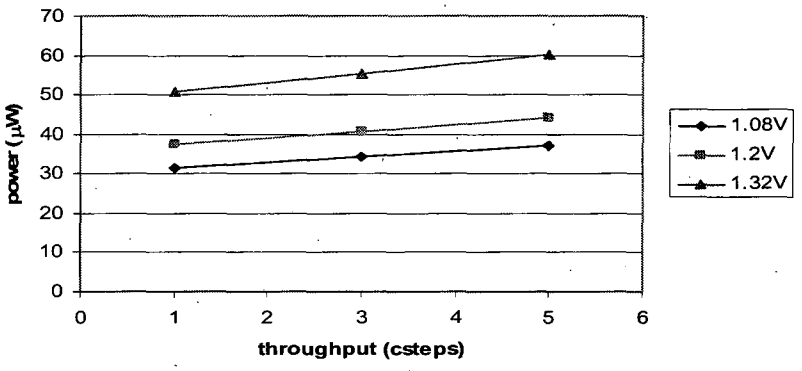
Table 6.1 0.12μm library components

MULTIPLIER, A = 4813 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	130.700	5.544
1.2	154.300	3.697
1.32	216.700	2.636
ADDER, A = 1107.4 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	30.700	4.050
1.2	37.370	2.706
1.32	50.830	1.920
SUBTRACTOR, A = 1176 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	31.590	4.260
1.2	42.600	2.833
1.32	57.780	1.973
COMPARATOR, A = 1585.5 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	54.850	4.978
1.2	69.960	3.197
1.32	90.950	2.174
SHIFTER, A = 306.6 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	7.264	0.231
1.2	8.675	0.139
1.32	11.840	0.095
AND, A = 74.6 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	1.421	0.377
1.2	1.693	0.247
1.32	2.338	0.174
OR, A = 74.6 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	1.970	0.405
1.2	2.353	0.264

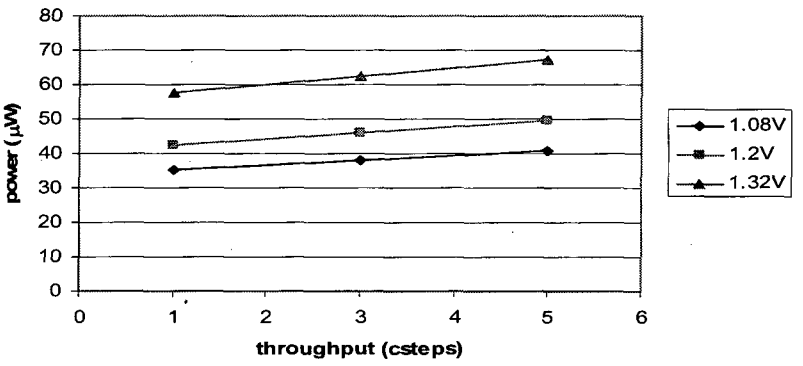
1.32	3.234	0.184
REGISTER, A = 581 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	44.150	0.529
1.2	52.000	0.334
1.32	69.000	0.223
MUX2, A = 258.2 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	6.713	0.142
1.2	8.031	0.091
1.32	10.980	0.061
MUX3, A = 548.7 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	11.060	0.366
1.2	13.200	0.233
1.32	18.030	0.150
MUX4, A = 548.7 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	11.060	0.366
1.2	13.200	0.233
1.32	18.030	0.150
MUX5, A = 806.9 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	18.690	0.551
1.2	22.550	0.351
1.32	30.710	0.231
MUX6, A = 1065.1 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	30.800	0.551
1.2	37.060	0.351
1.32	50.400	0.231
MUX7, A = 1355.6 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	35.000	0.553
1.2	42.260	0.352
1.32	57.590	0.232
MUX8, A = 1355.6 μm^2		
V (V)	P_{dyn}(μW)	D(ns)
1.08	35.000	0.553
1.2	42.260	0.352
1.32	57.590	0.232



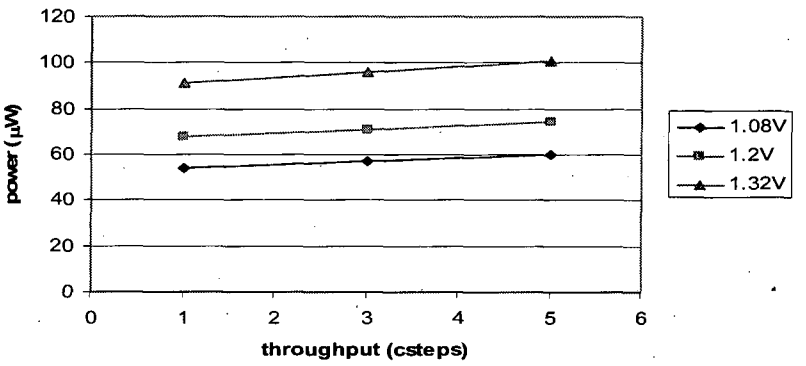
a) multiplier



b) adder



c) subtractor



d) comparator

Figure 6.4 Power consumption of multicycled components

It should be noted that similar library characterisation was done in Chapter 5, Table 5.9. The main difference is that the library from Table 5.9 was characterised using input vectors obtained with a LFSR (Linear Feedback Shift Register) for demonstrations purposes and library from Table 6.1 was characterised using real data. In addition to the data presented in Table 6.1, the library also contains information about the power variation when changing the throughput of some components, as shown in Figure 6.4. It can be seen that for each characterisation voltage, i.e. 1.08V, 1.2V or 1.32V, there is a linear increase of the power dissipated when increasing the number of cycles used by the component.

6.3.3 Optimisation parameter selection

To use PABCOM it is also necessary to define the simulated annealing parameters related to the cooling schedule. As previously explained in Chapter 5, the initial acceptance ratio χ_0 is set to 0.95 and the stop parameter ϵ_s is set to 0.0001. This section discusses the selection of a suitable distance parameter δ for the power-aware synthesis process. Figure 6.5 shows the percentage of difference between achieved cost and lowest cost as a function of the distance parameter δ. Notice that for values of $10 \leq \delta < 0.4$ the cost difference presents an irregular behaviour and it is difficult to establish its relation with the distance parameter δ. However, for values of $\delta \leq 0.4$ there is a clear trend of the cost difference to decrease. An acceptable cost difference of 2% approximately is obtained for a distance parameter $\delta <= 0.2$, with the solution obtained in the lowest run-time for $\delta = 0.2$. Hence, the distance parameter δ is set to 0.2, which is used to obtain the solutions presented in next section in Table 6.2. Due to the strong dependence between the design quality and the distance parameter δ, it is recommended to perform the power-aware synthesis process for a number of different δ values.

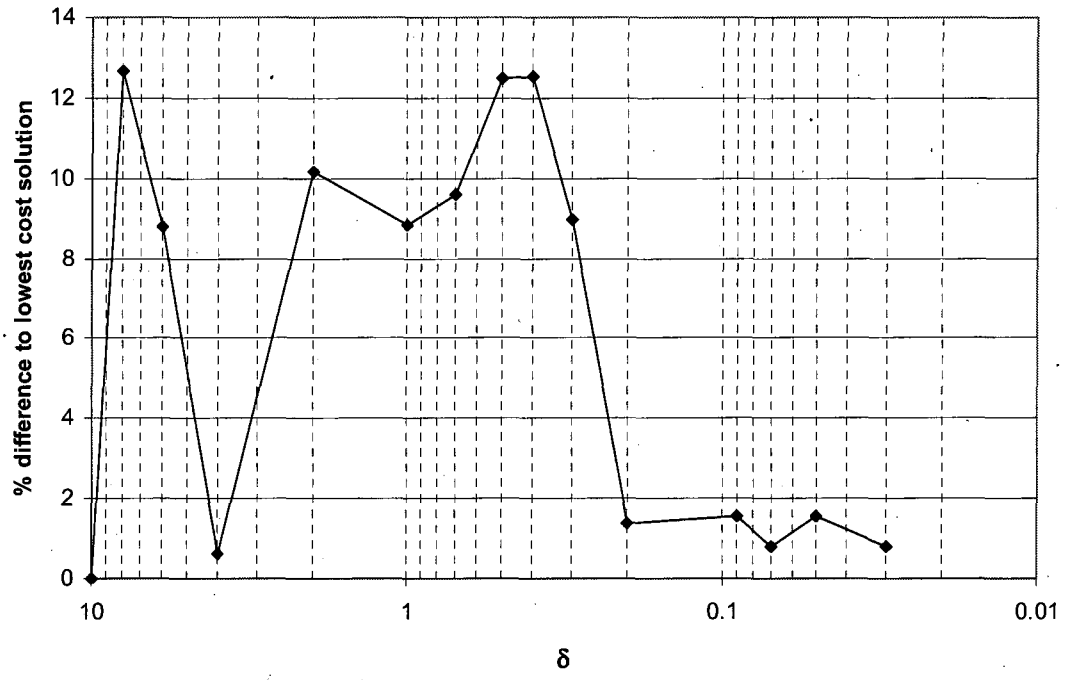


Figure 6.5 Dependence of design cost on the parameter δ

6.3.4 Results of the synthesis process

PABCOM explores efficiently the design space and investigates explicitly power-area tradeoffs that may be of interest to the designer. These optimised power-area tradeoffs are obtained according to the optimisation goal specified by the user: power, area or a combination thereof. The optimisation goal is defined in the cost function by α, which is the optimisation weight for the power cost, and $(1-\alpha)$, which represents the optimisation weight for the area cost.

By varying the values of α from 1 to 0 in steps of 0.1, PABCOM obtains solutions with different area and power costs, as shown in Table 6.2. These power and area costs were estimated respectively using equations (5.2) and (5.6) from Chapter 5, in combination with the library component from Table 6.1. Note that equations (5.2) and (5.6) ignore respectively the contribution of the controller to the power and area of the design. This is a valid approximation for data dominated designs as demonstrated later in Section 6.4.2 and Section 6.4.3. Power and area values from Table 6.2 will be henceforth referred as approximated values.

Table 6.2 Solutions for the motion vector reconstructor obtained with different α

α	Area (μm^2)	Power (μW)
1	42872	835
0.9	41179	833
0.8	42493	848
0.7	40266	832
0.6	41333	866
0.5	34469	1000
0.4	34368	999
0.3	31693	1015
0.2	32058	922
0.1	31860	921
0	31818	1017

The best power-area tradeoffs from Table 6.2 are shown in Figure 6.6, where it can be seen that the solution for $\alpha = 0.3$ has similar area to the solution for $\alpha = 0.1$, but higher power consumption. Solution for $\alpha = 0.7$ presents lower power dissipation but occupies more area than solution for $\alpha = 0.1$. Thus, solutions for $\alpha = 0.7$ and $\alpha = 0.1$ are chosen for implementation since they present different power and area requirements. For the remaining of this section, the solution for $\alpha = 0.1$ will be referred as Design 1 and the solution for $\alpha = 0.7$ as Design 2. The datapath and controller of Design 1 and Design 2 are defined using the scheduling and binding information that is included in the synthesis results.

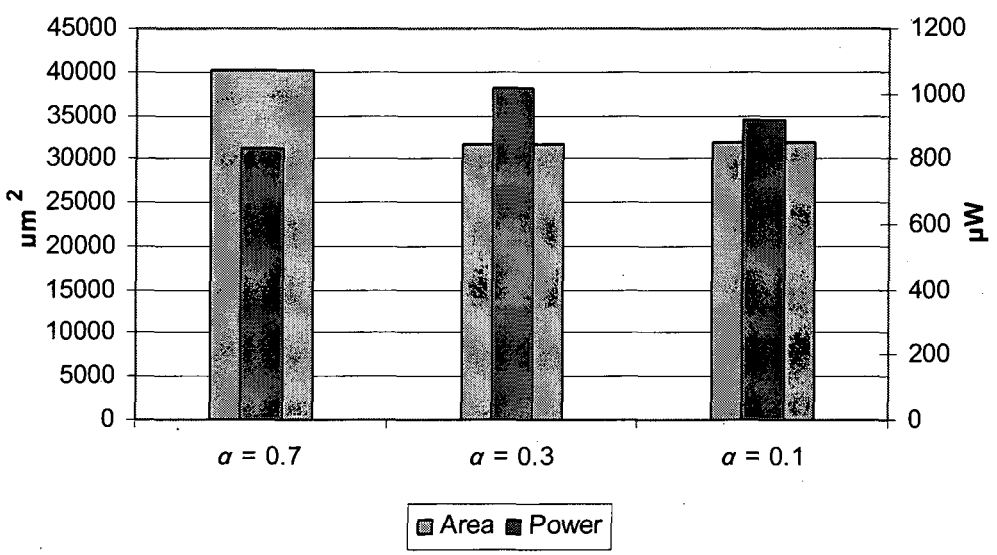


Figure 6.6 Best power-area tradeoffs chosen from Table 6.2.

Scheduling

The schedules of Design 1 and Design 2 are shown in Figure 6.7 and Figure 6.8 respectively. Although both schedules are executed within the same time, i.e. 50ns, they present different schedule length, i.e. 13 csteps for Design 1 and 20 csteps for Design 2. Consequently, the clock period for Design 1 is 3.8ns whereas for Design 2 is 2.5ns. The proposed algorithm determined that all operations in Design 1 are single cycle whereas some operations of Design 2 are multicycled, i.e. multiplications, additions, subtractions and comparisons. This is better illustrated in Table 6.3, which summarises the operations throughput and clock period selected by the proposed algorithm for the schedules of Figure 6.7 and Figure 6.8. In the header table, *Ls* is the schedule length, *Tclk* is the clock period, TP_M, TP_C, TP_A, TP_S, TP_{SFT}, TP_{AND}, and TP_{OR} are the operations throughput for the MULTIPLIER, COMPARATOR, ADDER, SUBTRACTOR, SHIFTER, AND and OR respectively, and *V* is the operating voltage.

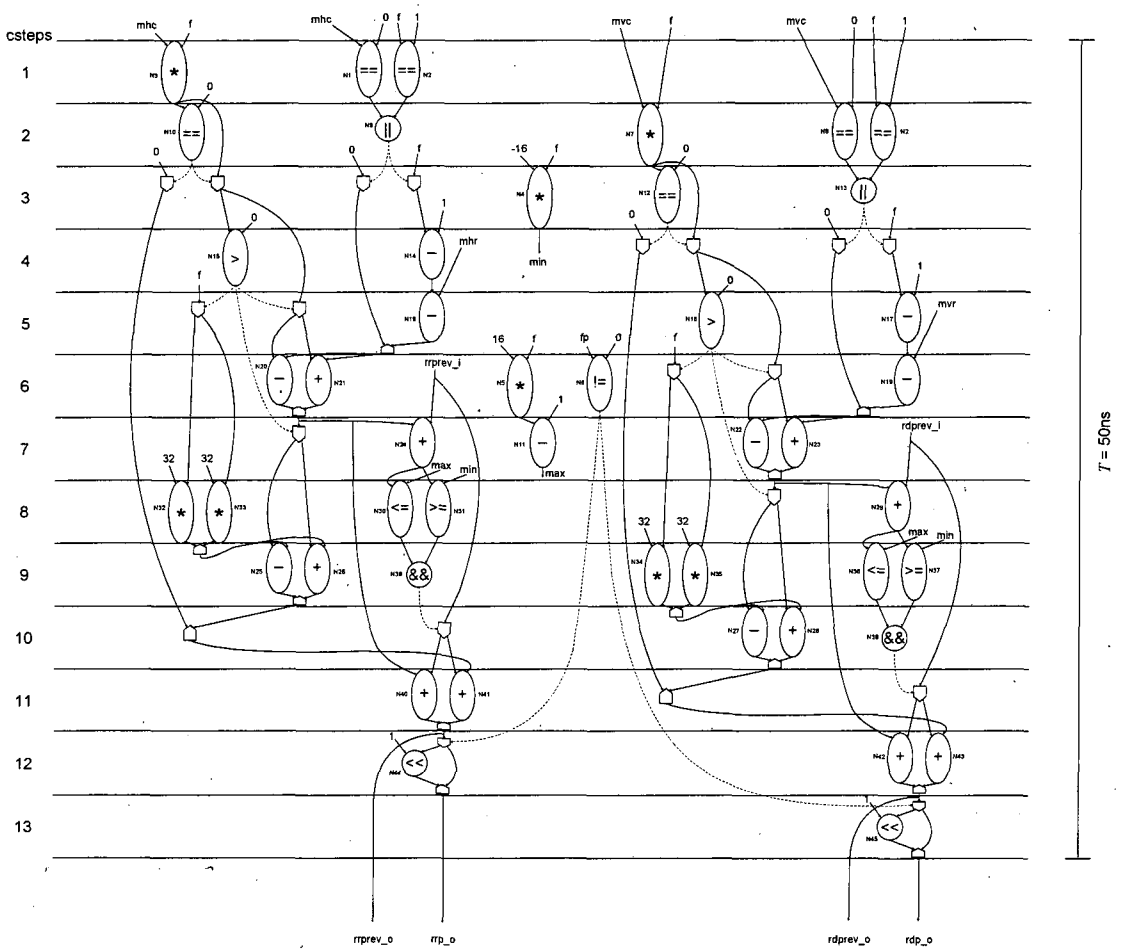


Figure 6.7 Design 1 schedule

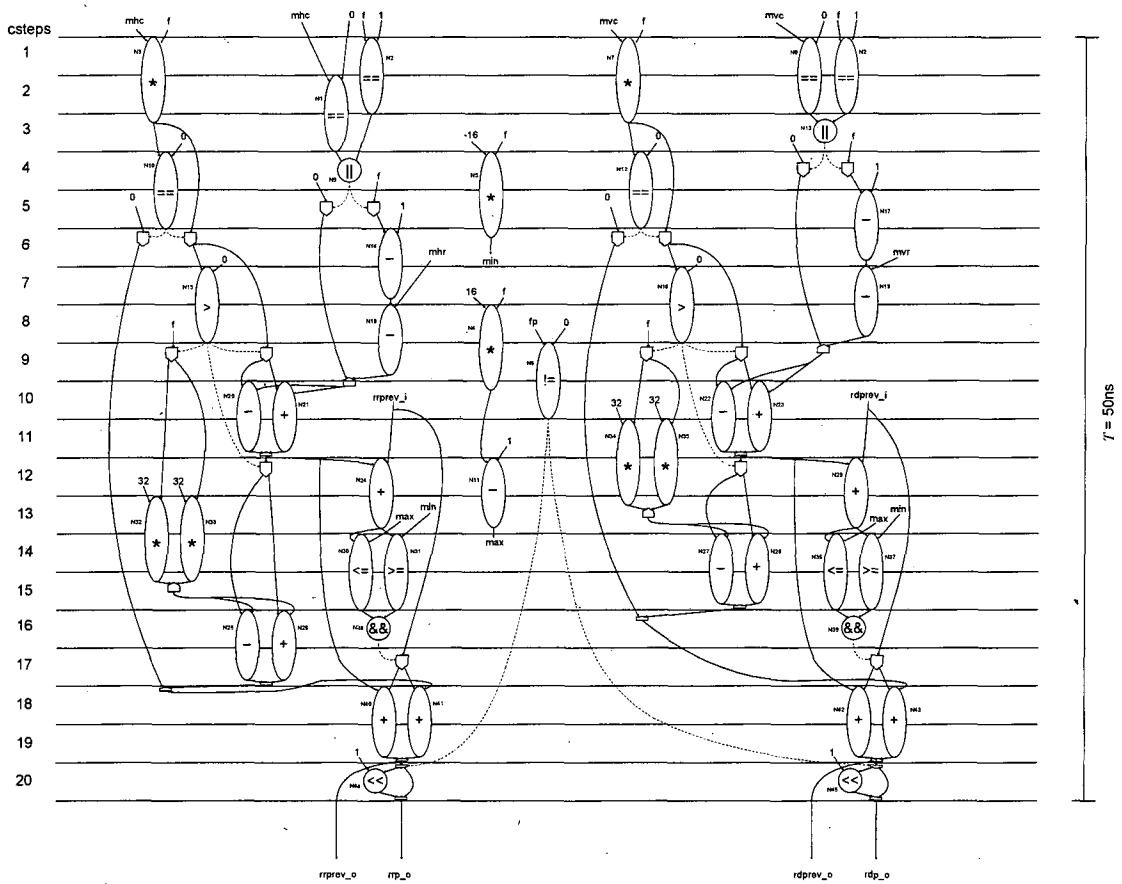


Figure 6.8 Design 2 schedule

Table 6.3 Characteristics of the Design 1 and Design 2

	Ls (cs)	*Tclk* (ns)	TP_M (cs)	TP_C (cs)	TP_A (cs)	TP_S (cs)	TP_{SFT} (cs)	TP_{AND} (cs)	TP_{OR} (cs)	*V* (V)
Design 1	13	3.8	1	1	1	1	1	1	1	1.25
Design 2	20	2.5	3	2	2	2	1	1	1	1.15

cs: abbreviation of csteps

The combination of clock period and operations throughput presented in Table 6.3 allows a minimum operating voltage of 1.25V for Design 1 and 1.15V for Design 2. Using these operating voltages, an estimated power consumption of 921μW and 832μW is obtained for Design 1 and Design 2 respectively, as shown in Table 6.2. Notice that further voltage scaling for these designs is not possible because the time constraint of 50ns would be violated.

Module Binding

The module bindings of Design 1 and Design 2 are shown in Figure 6.9 and Figure 6.10 respectively. It can be seen that Design 2 requires more functional modules than Design 1 due to the greater number of same type operations whose execution times

overlap at the same cstep. For example, in Design 2 (Figure 6.10) the execution times of four comparisons, i.e. N37, N31, N30 and N36, are overlapped along csteps 14 and 15, requiring then four comparators, i.e. M44, M43, M22 and M21. However, in Design 1 (Figure 6.9), the execution times of only two comparisons are overlapped at the same cstep, i.e. N1 and N2 in cstep 1, N10 and N8 in cstep 2, N31 and N30 in cstep 8, N37 and N36 in cstep 9, requiring two comparators, i.e. M22 and M21. In the case of multiplications, they are mapped to multipliers M7 and M6 in Design 2, and to multiplier M7 in Design 1. In Design 2, the additions are bound to adders M2 and M4, and the subtractions to subtractors M13 and M14. In Design 1 the additions are bound to adders M1 and M2, and the subtractions to subtractors M12 and M13. From Figure 6.9 and Figure 6.10 it can also be seen that mutual exclusive operations are bound to the same module to reduce the area of the design. For example, in Design 1 operations N42 and N43 are mapped to module M1 at cstep 12, and in Design 2 operations N33 and N32 are mapped to module M6 along csteps 13, 14 and 15.

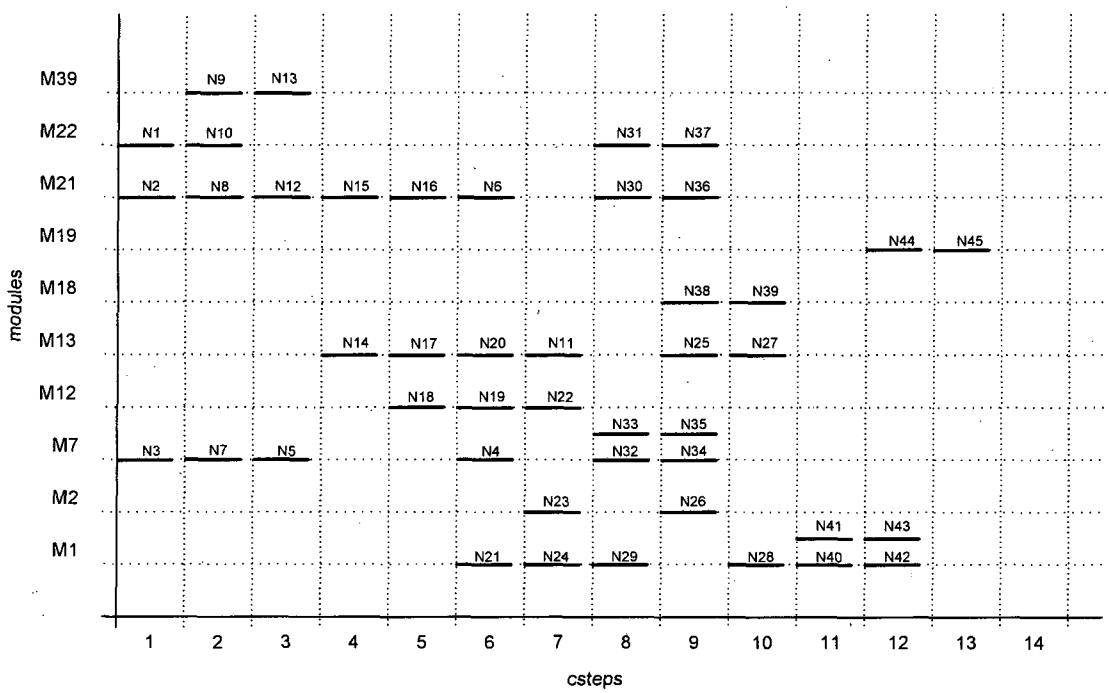


Figure 6.9 Design 1 module binding

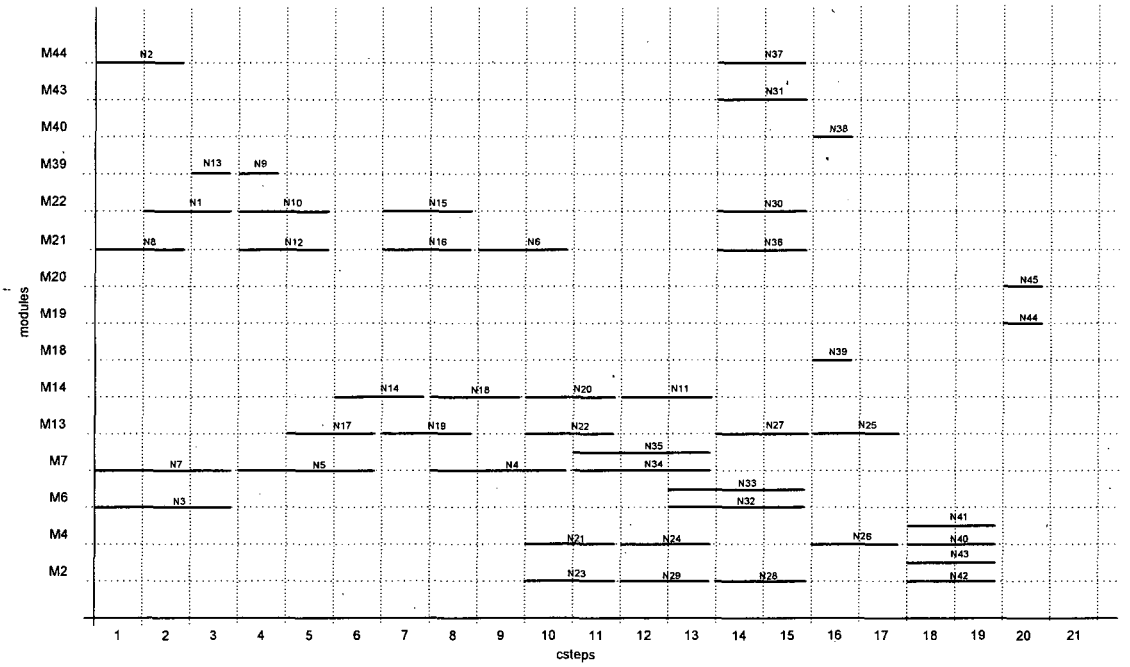


Figure 6.10 Design 2 module binding

Register Binding

The register bindings of Design 1 and Design 2 are shown in Figure 6.11 and Figure 6.12 respectively. Note that Design 2 requires three more registers than Design 1 leading to less register sharing. It can also be seen that the variables in the registers of Design 2 have longer lifetime than in Design 1, mainly because of the use of multicycled functional modules. For example, variable N24 in Design 2 (Figure 6.12) needs to be hold during csteps 14 and 15 so that the multicycled operations N30 and N31 can be executed, as shown in the schedule of Figure 6.8. Variable N24 in Design 1 (Figure 6.11) needs to be hold only during cstep 8 to execute operations N30 and N31, as shown in the schedule of Figure 6.7. From Figure 6.11 and Figure 6.12 it can be seen that values of mutual exclusive operations are bound to the same register to reduce the area of the design. For example, in Design 1 variables N20 and N21 are mapped to register R4 holding the value from cstep 7 to cstep 11. There are some cases when more than two values are bound to the same register, for example, R10, N25 and N26 in Design 2 are mapped to register R1. This occurs when two consecutive joints are present in the DFG, for example in Figure 6.8, N25 and N26 are the inputs to a joint, which in turn is the input to another joint.

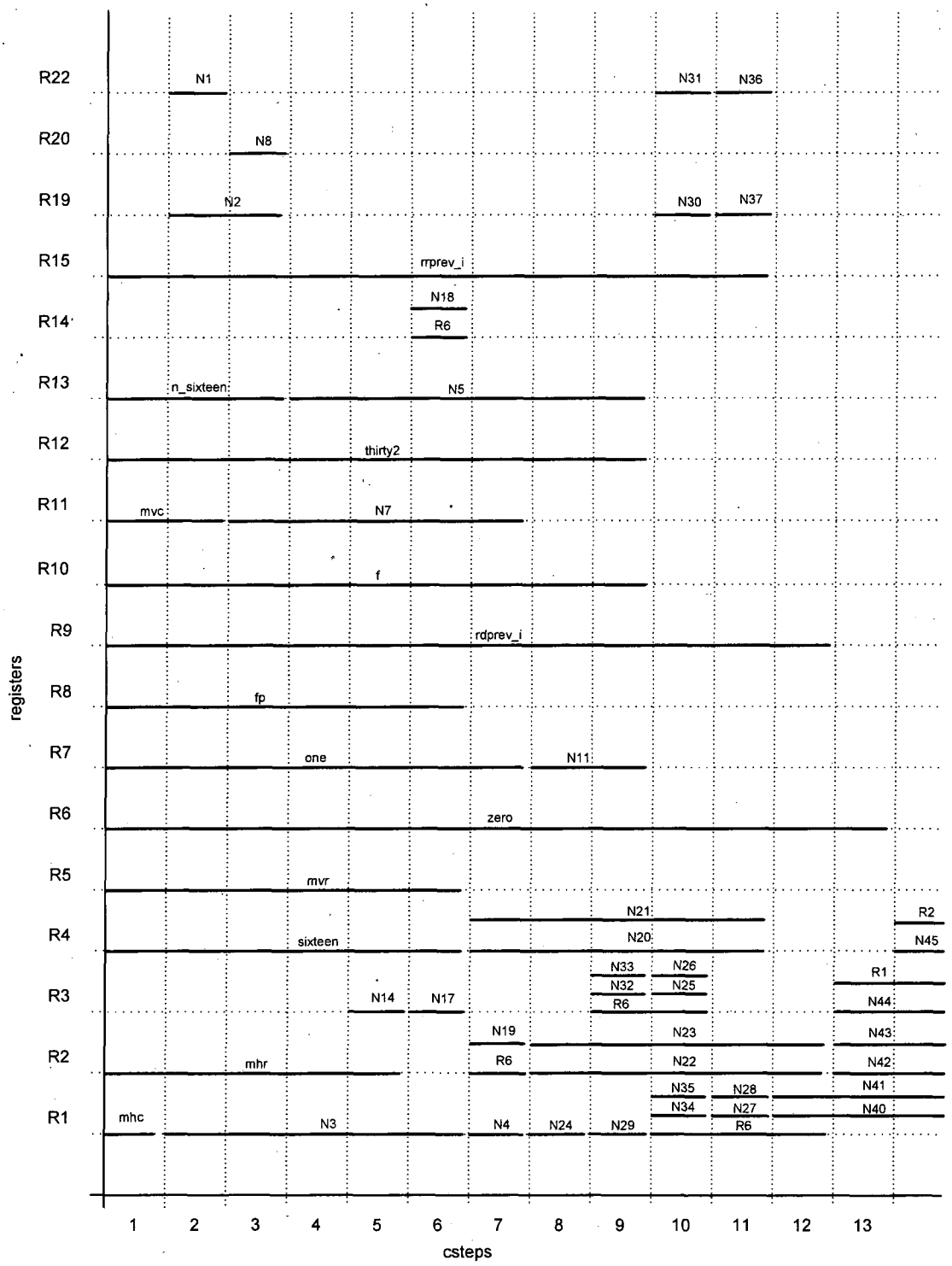


Figure 6.11 Design 1 register binding

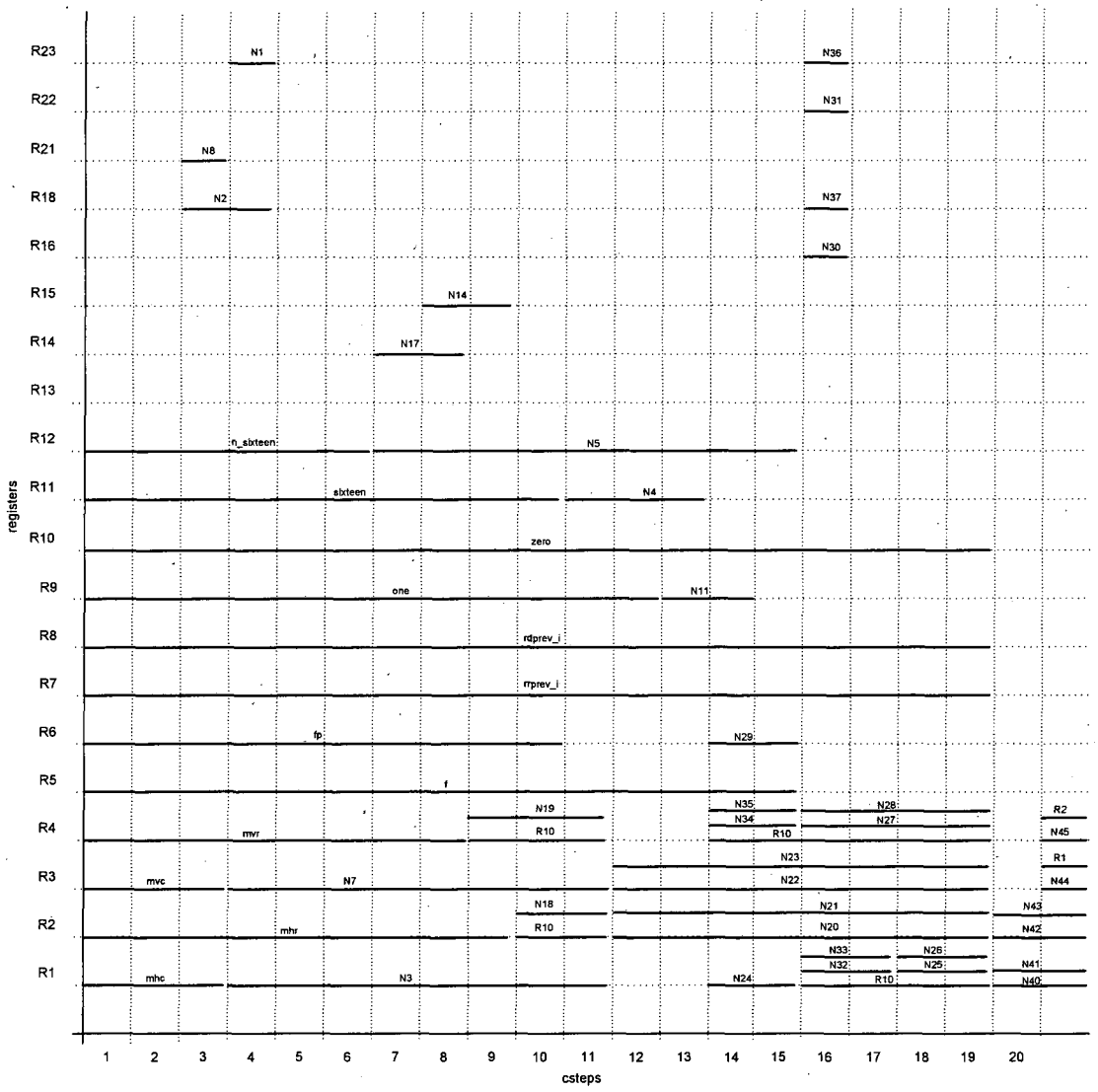


Figure 6.12 Design 2 register binding

Generation of datapath structures

With the information provided by the module and register binding it is possible to define the datapath of Design 1 and Design 2 as shown in Figure 6.13 and Figure 6.14 respectively. All the functional units in Figure 6.13 execute the single cycle operations of the schedule shown in Figure 6.7. The functional units used in Figure 6.14 execute both, the single cycle and multicycled operations of the schedule shown in Figure 6.8. The datapaths in Figure 6.13 and Figure 6.14 use registers with two different word length: 16–bits and 1-bit. Registers of 16-bits (R1, R2, ..., R15), are used to save the resulting values from functional units such as: multiplier, adder, subtractor and shifter. Registers of 1-bit (R19, R20 and R21 in Design 1, and R16, R18, R21, R22 and R23 in Design 2), are used to save the resulting values from the comparator. It can also be seen that both datapaths use status registers (stR1, stR2,

..., stR10) due to the global slicing technique used for state assignment when designing the controller. Status registers are 1-bit registers that store the information about which part of the conditional branch will be executed in the states defined by the controller. Two groups of multiplexers complete the datapaths from Figure 6.13 and Figure 6.14. One group is connected to the inputs of functional modules whereas the other group is connected to the inputs of the registers. These groups of multiplexers allow the functional modules and registers to be shared, reducing the area of the design.

Next section shows the realisation of the datapaths from Figure 6.13 and Figure 6.14 and the interconnection with their respective controller to complete the design. Both complete designs will be functionally validated through logic simulation and subjected to area and power analysis.

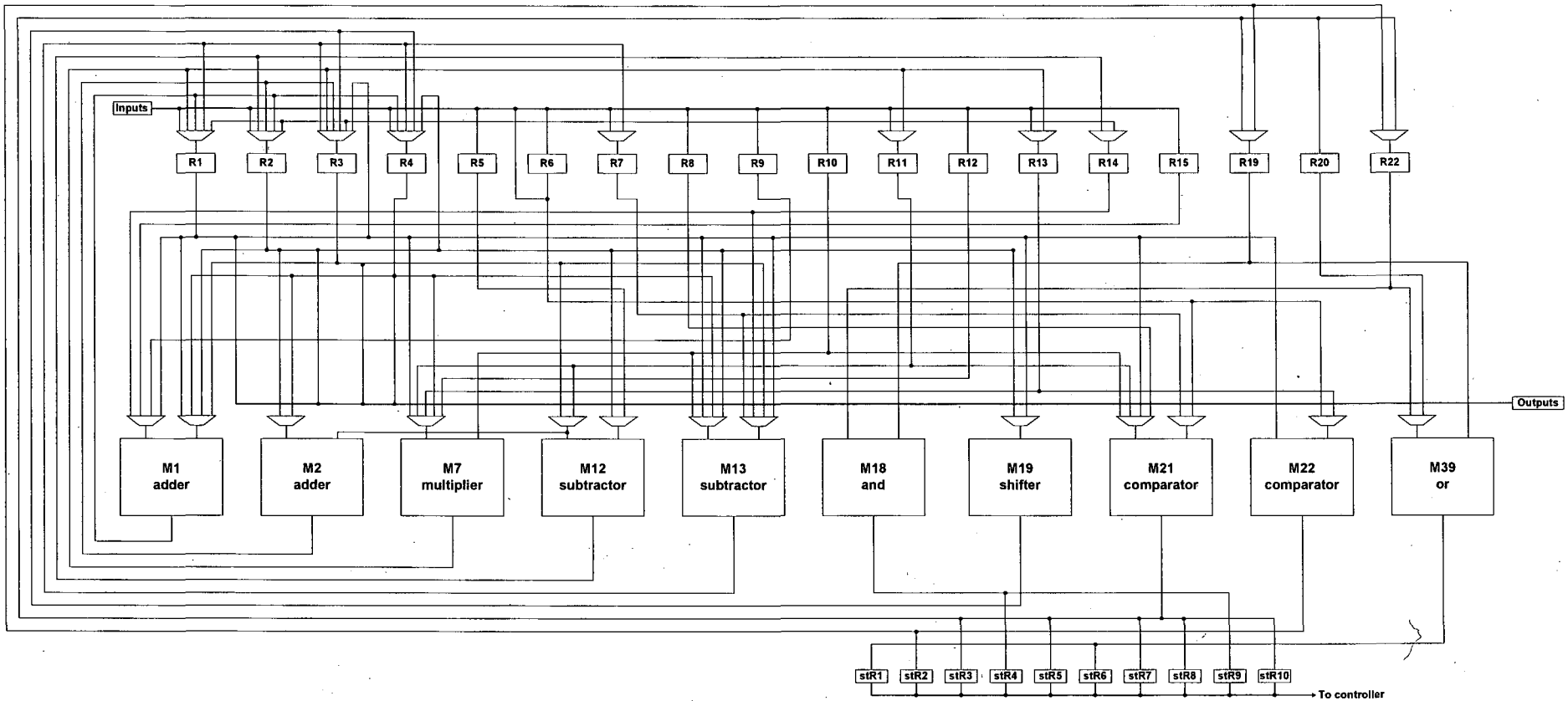


Figure 6.13 Design 1 datapath

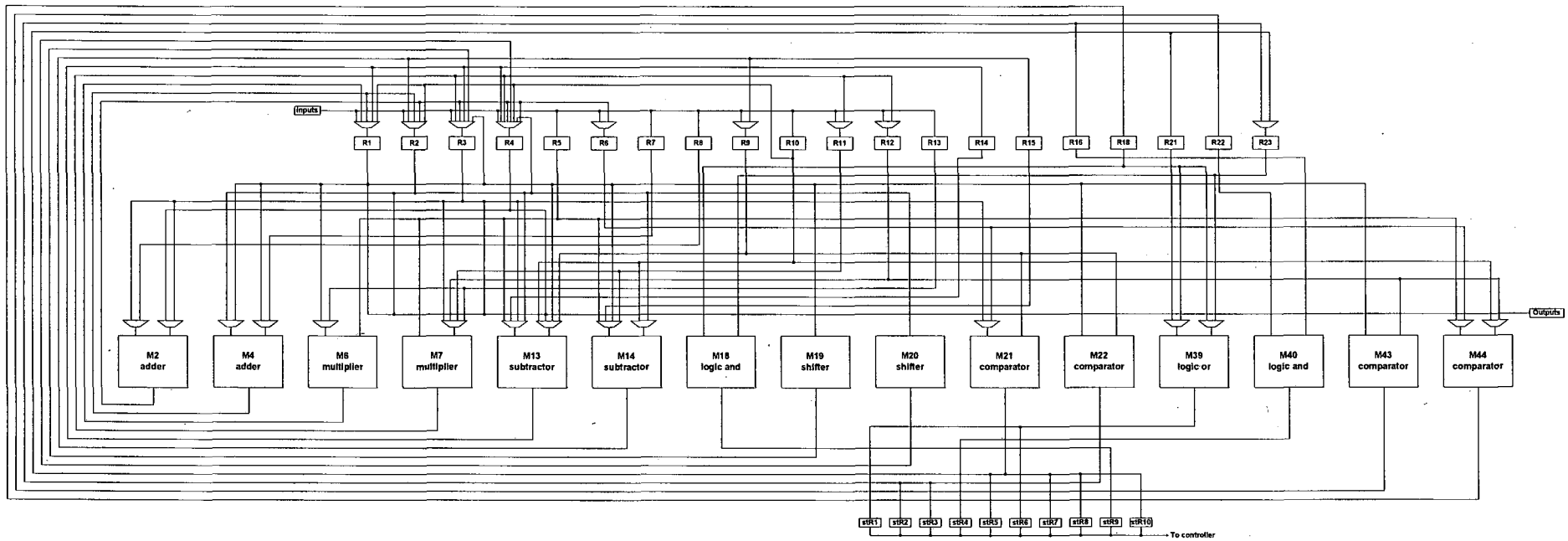


Figure 6.14 Design 2 datapath

6.4 Realisation of the motion vector reconstructor

The previous section has described the schedules, bindings and generation of datapath structures corresponding to two solutions for the motion vector reconstructor. All these results from the power-aware synthesis are used to complete the design as shown in Figure 6.15.

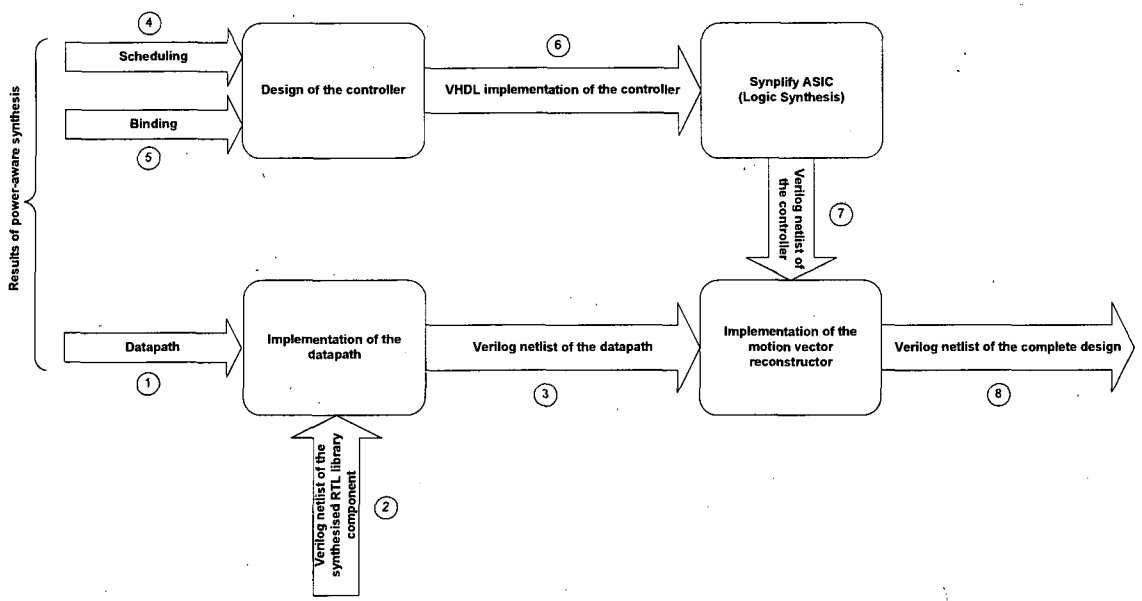


Figure 6.15 Design flow for the complete design

The datapath structures ① contain all the information necessary to generate an equivalent RTL description and the Verilog codes corresponding to Figure 6.13 and Figure 6.14 are included in Appendix 1. The structural Verilog ③ represents the interconnection of the RTL library components ② from Table 6.1, which have been previously synthesised using Synplify ASIC. To complete the design of the motion vector reconstructor, a controller modelled as a finite state machine (FSM) is designed for each datapath. The generation of the FSM starts with the state assignment of the operations from a scheduled design. The state assignment of the FSM was performed using a global slicing technique, which assigns to the same state operations that are mutually exclusive but are executed in the same cstep, as explained previously in Chapter 2 Section 2.2. According to [2], using global slicing with status registers always produces designs with lower area than using local slicing. To complete the FSM, enable signals for both, modules and registers, and select signals for multiplexers, need to be specified according to the schedule ④ and

binding ⑤ presented in Section 6.3.4. Once the controller is completely designed, it is implemented in VHDL ⑥ and later synthesised using Synplify ASIC. Then, the synthesised controller⑦ is interconnected to the synthesised datapath ③ to produce the complete design ⑧. Listing 6.2 and Listing 6.3 show respectively the RTL code of Design 1 and Design 2 after the interconnection of the controller and datapath. The interconnection is carried out using signals declare as "wire". Listing 6.3 requires more signals for interconnection than Listing 6.2 due to the bigger datapath of Design 2, as shown in Section 6.3. Notice that in both listings the top level module "vec_rec" presents the same inputs and outputs. However, the instances "data1" and "control1" in Listing 6.2 are different from the instances "data2" and "control2" in Listing 6.3. Instance "data2" represents the module corresponding to the datapath of Design 2 (Figure 6.14), and the module generated for its respective controller after logic synthesis is "control2". Instance "data1" represents the module corresponding to the datapath of Design 1 (Figure 6.13), and the module generated for its respective controller after logic synthesis is "control1".

Listing 6.2 Complete Design 1 RTL code

```
module vec_rec (
        rst, clk,
        mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2,
        rrprev_i, rdprev_i,
        rrprev_o, rdprev_o, rrp_o, rdp_o,
        done
);
input rst;
input clk;
input [15:0] mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2;
input [15:0] rrprev_i, rdprev_i;
output [15:0] rrprev_o, rdprev_o, rrp_o, rdp_o;
output done;

wire stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10;
wire eM1, eM2, eM7, eM12, eM13, eM18, eM19, eM21, eM22, eM39;
wire ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
     ldR14, ldR15, ldR19, ldR20, ldR22;
wire ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10;
wire [2:0] opM21, opM22;
wire [1:0] sM1a, sM1b, sM13a, sM13b, sM21a;
wire sM2a, sM12a, sM12b, sM19a, sM21b, sM22b, sM39a, sR7a, sR11a, sR13a, sR14a,
     sR19a, sR22a;
wire [2:0] sM7a, sR1a, sR2a, sR3a, sR4a;

    controller control1 (
        clk, rst, stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10,
        eM1, eM2, eM7, eM12, eM13, eM18, eM19, eM21, eM22, eM39,
        ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
```

```
        ldR14, ldR15, ldR19, ldR20, ldR22,
        ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,
        opM21, opM22,
        sM1a, sM1b, sM13a, sM13b, sM21a,
        sM2a, sM12a, sM12b, sM19a, sM21b, sM22b, sM39a, sR7a, sR11a, sR13a, sR14a,
        sR19a, sR22a,
        sM7a, sR1a, sR2a, sR3a, sR4a,
        done
    );

    datapath data1 (
        rst, clk,
        eM1, eM2, eM7, eM12, eM13, eM18, eM19, eM21, eM22, eM39,
        ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
        ldR14, ldR15, ldR19, ldR20, ldR22,
        ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,
        opM21, opM22,
        sM1a, sM1b, sM2a, sM7a, sM12a, sM12b, sM13a, sM13b, sM19a, sM21a, sM21b,
        sM22b, sM39a,
        sR1a, sR2a, sR3a, sR4a, sR7a, sR11a, sR13a, sR14a, sR19a, sR22a,
        mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2,
        rrprev_i, rdprev_i,
        rrprev_o, rdprev_o, rrp_o, rdp_o,
        stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10
    );
endmodule
```

Listing 6.3 Complete Design 2 RTL code

```
module vec_rec (
        rst, clk,
        mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2,
        rrprev_i, rdprev_i,
        rrprev_o, rdprev_o, rrp_o, rdp_o,
        done
);
input rst;
input clk;
input [15:0] mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2;
input [15:0] rrprev_i, rdprev_i;
output [15:0] rrprev_o, rdprev_o, rrp_o, rdp_o;
output done;

wire stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10;
wire eM2, eM4, eM6, eM7, eM13, eM14, eM18, eM19, eM20, eM21, eM22, eM39, eM40,
     eM43, eM44;
wire ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
     ldR14, ldR15, ldR16, ldR18, ldR21, ldR22, ldR23;
wire ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10;
wire [2:0] opM21, opM22, opM43, opM44;
wire [1:0] sM7b, sM13a, sM13b, sM14a;
wire sM2a, sM2b, sM4a, sM4b, sM6a, sM14b, sM21a, sM39a, sM39b, sM44a, sM44b, sR6a,
     sR9a, sR11a, sR12a, sR23a;
wire [2:0] sR1a, sR2a, sR3a, sR4a;

    controller control2 (
        clk, rst, stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10,
        eM2, eM4, eM6, eM7, eM13, eM14, eM18, eM19, eM20, eM21, eM22, eM39, eM40,
        eM43, eM44,
        ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
```

```
        ldR14, ldR15, ldR16, ldR18, ldR21, ldR22, ldR23,
        ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,
        opM21, opM22, opM43, opM44,
        sM7b, sM13a, sM13b, sM14a,
        sM2a, sM2b, sM4a, sM4b, sM6a, sM14b, sM21a, sM39a, sM39b, sM44a, sM44b,
        sR6a, sR9a, sR11a, sR12a, sR23a,
        sR1a, sR2a, sR3a, sR4a,
        done
    );

    datapath data2 (
        rst, clk,
        eM2, eM4, eM6, eM7, eM13, eM14, eM18, eM19, eM20, eM21, eM22, eM39, eM40,
        eM43, eM44,
        ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
        ldR14, ldR15, ldR16, ldR18, ldR21, ldR22, ldR23,
        ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,
        opM21, opM22, opM43, opM44,
        sM2a, sM2b, sM4a, sM4b, sM6a, sM7b, sM13a, sM13b, sM14a, sM14b, sM21a,
        sM39a, sM39b, sM44a, sM44b,
        sR1a, sR2a, sR3a, sR4a, sR6a, sR9a, sR11a, sR12a, sR23a,
        mhc, mhr, mvc, mvr, f, fp, zero, one, sixteen, n_sixteen, thirty2,
        rrprev_i, rdprev_i,
        rrprev_o, rdprev_o, rrp_o, rdp_o,
        stR1, stR2, stR3, stR4, stR5, stR6, stR7, stR8, stR9, stR10
    );
endmodule
```

6.4.1 Functional validation

Once Design 1 and Design 2 have been completed, their functionality is validated following the methodology shown in Figure 6.16.

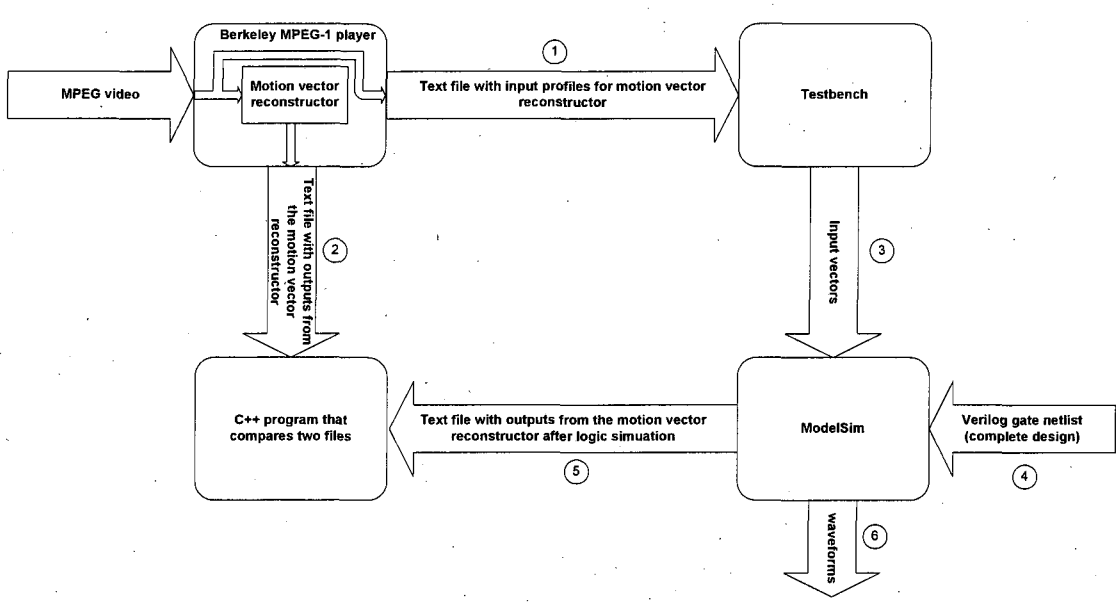


Figure 6.16 Functional validation flow for the complete design

Firstly, a video is played with a modified Berkeley MPEG-1 player that generates text files that contain the inputs ① and outputs ② of the motion vector reconstructor. The text file with the inputs ① is then read by a testbench to provide the input vectors ③ for the timing simulation of the complete design ④ using ModelSim. The complete design is in the form of a Verilog gate netlist as explained at the beginning of Section 6.4. During timing simulation, all the generated output values from the motion vector reconstructor are saved in a text file ⑤. This text file is compared with the text file that contains the outputs obtained by the motion vector reconstructor ② when playing a video with the Berkeley MPEG-1 decoder. The comparison is made using a program written in C++ that finds the differences between two text files and displays the line numbers where these differences have been observed. After performing the comparison, no differences were detected between the text files ② and ⑤, which validates the functionality of the design for all the input vectors ③. After the timing simulation with ModelSim, waveforms ⑥ for Design 1 and Design 2 are obtained. Examples of such waveforms are given in Figure 6.17 and Figure 6.18. These diagrams show the inputs (rst, clk, mhc, ..., rdprev_i) and outputs (rrprev_o, rdprev_o, rrp_o, rdp_o), of the top level module shown in Listing 6.2 and Listing 6.3. The inputs were provided by the testbench after reading the text file ① that contains the input profiles for the motion vector reconstructor.

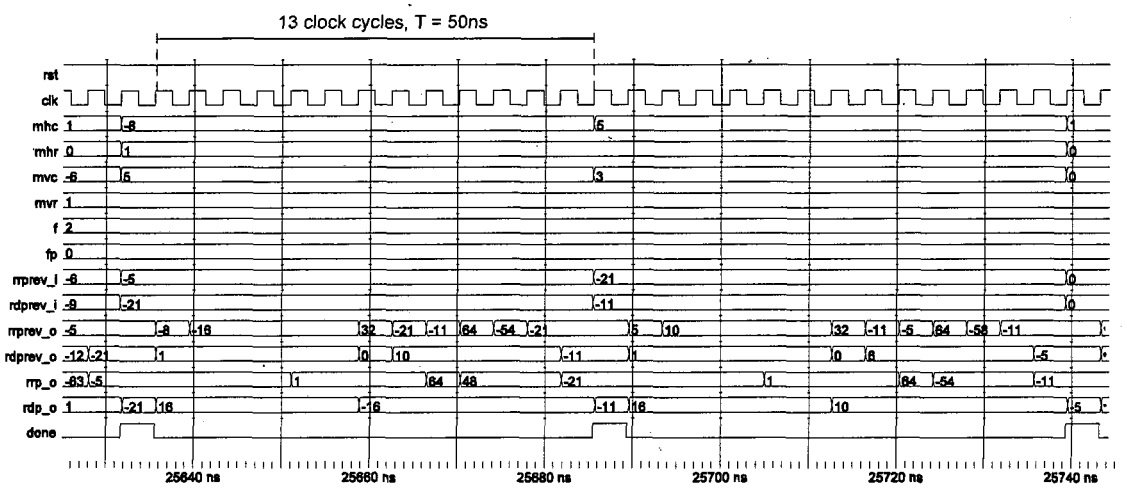


Figure 6.17 Timing simulation of Design 1

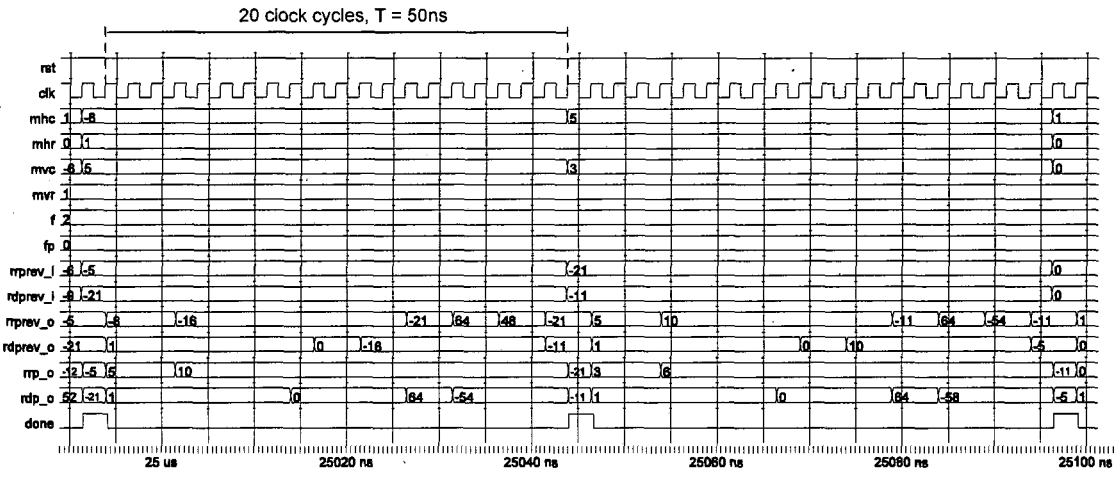


Figure 6.18 Timing simulation of Design 2

In Figure 6.17, outputs are ready every 13 clock cycles, matching with the schedule length specified earlier in the schedule for Design 1 in Figure 6.7. In Figure 6.18, outputs are ready every 20 clock cycles, matching with the schedule length specified earlier in the schedule for Design 2 in Figure 6.8. Note that although the outputs are ready after different number of clock cycles, the sample period remains the same in both figures, i.e. 50 ns.

6.4.2 Area cost

This section presents an analysis of the actual area cost of Design 1 and Design 2, which includes the required area for the datapath and controller after logic synthesis. As described in the previous section, the datapath is constructed by interconnecting the synthesised library components from Table 6.1. Hence, the area of the datapath is equal to the sum of the individual area of each datapath component, i.e. modules, registers and multiplexers. This results in a datapath area of 31860μm^2 for Design 1 and 40266μm^2 for Design 2. The difference in area is due to the use of more functional modules in Design 2 than in Design 1, as can be seen from Figure 6.13 and Figure 6.14. The area values for the controllers of Design 1 and Design 2 were obtained after performing logic synthesis as shown in Figure 6.15. The controller of Design 1 has an area of 1456μm^2 and the controller of Design 2 occupies 1876μm^2. The controller is bigger in Design 2 than in Design 1 since it needs to generate more control signals due to a more complex datapath (Figure 6.14) that executes multicycled operations. Moreover, the controller in Design 2 has more states than in

Design 1 because of a larger schedule length, as shown previously in the schedules of Figures 6.7 and 6.8.

The actual area cost for Design 1 and Design 2 including the datapath and controller is 33316μm^2 and 42142μm^2 respectively. These actual values present a close correlation with the approximated values from Table 6.2, which do not include the area of the controller. This proves that the area cost function (equation (5.6) in Chapter 5) used by PABCOM is reliable for data dominated designs, where the controller contributes very few to the total area of the design, as shown in Figure 6.19.

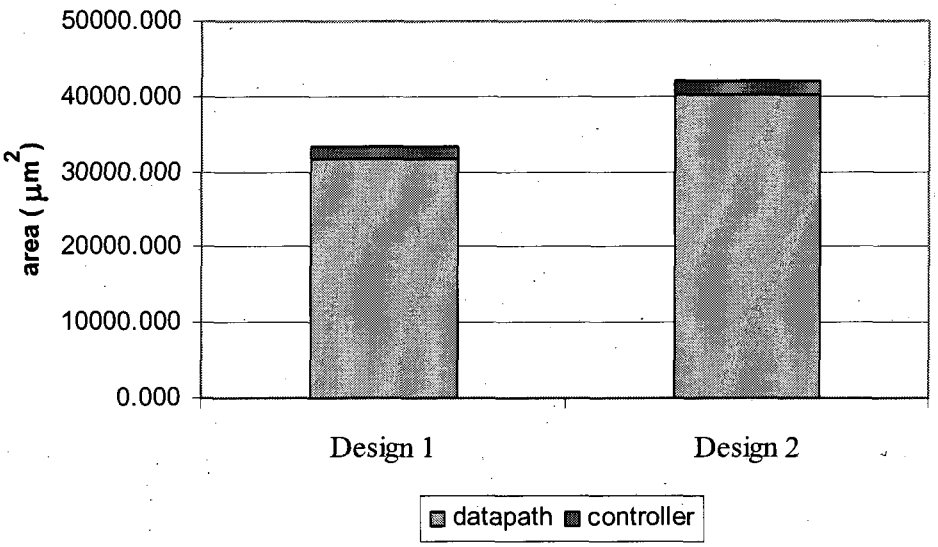


Figure 6.19 Area of two designs for the motion vector reconstructor

Datapath components area

To give an insight into the area contributions of the various components of the datapath for Design 1 and Design 2, consider Figure 6.20. It can be seen that the area of the modules in Design 2 is bigger than in Design 1. This is because Design 2 uses more modules than Design 1, i.e. 1 multiplier, 1 logic-and, 1 shifter and 2 comparators, as shown in the datapaths of Figure 6.13 and Figure 6.14.

Note that the registers area is almost the same for both designs. This is because Design 2 uses only two 1-bit registers more than Design 1. The multiplexers area is also very similar for both designs and the quantity, type and individual area of the multiplexers used in each design is given in Table 6.4.

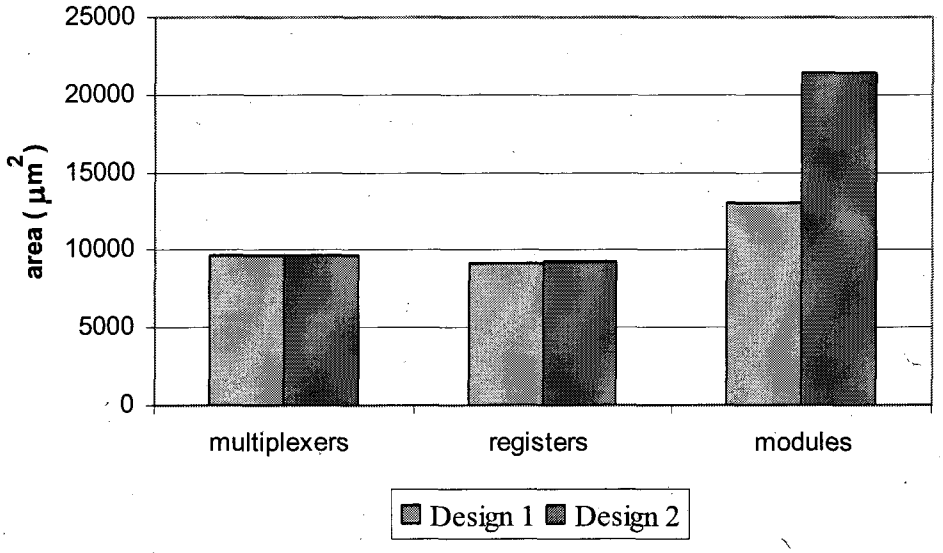


Figure 6.20 Area of the datapath components in Design 1 and Design 2

Table 6.4 Multiplexers requirements for both designs

Design 1			Design 2		
quantity	type	Area (μm^2)	quantity	type	Area (μm^2)
3	Mux 2-1 (1bit)	48.3	3	Mux 2-1 (1-bit)	48.3
10	Mux 2-1	2582	13	Mux 2-1	3356.6
5	Mux 4-1	2743.5	4	Mux 4-1	2194.8
4	Mux 5-1	3227.6	2	Mux 5-1	1613.8
1	Mux 6-1	1065.1	1	Mux 6-1	1065.1
			1	Mux 8-1	1355.6
Total area of muxes = 9666.5 μm^2			Total area of muxes= 9634.2 μm^2		

6.4.3 Power cost

To obtain the power dissipated by Design 1 and Design 2 at their respective operating voltage, i.e. 1.25V and 1.15V (according to PABCOM), the flow shown in Figure 6.21 was adopted. Firstly, gate-level power analysis using PrimePower [105] from Synopsys was carried out at each characterization voltage of the 0.12μm ST library, i.e. 1.08V, 1.2V and 1.32V, (Figure 6.21a). To estimate the power consumption at the selected operating voltage for the design, the quadratic dependency of power on voltage is modelled using the three pairs (voltage, power) previously obtained and a 2nd order Lagrange interpolation polynomial (Figure 6.21b).

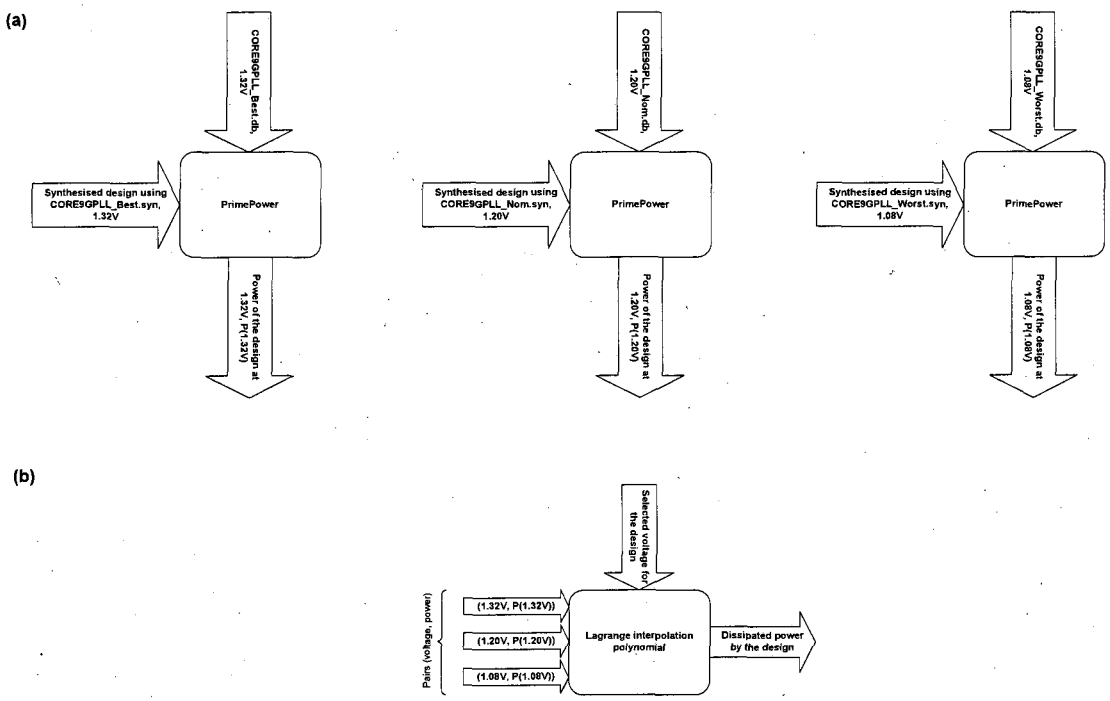


Figure 6.21 Actual power cost flow

Figure 6.22 shows the dynamic power consumption for Design 1 and Design 2 at characterisation voltages 1.08V, 1.20V and 1.32V. Using such power values and characterisation voltages in combination with a 2^{nd} order Lagrange interpolation polynomial resulted in an actual power consumption of 3.3mW at 1.25V for Design 1 and of 2.8mW at 1.15V for Design 2. These actual power consumption values are consistent with the approximated power values from Figure 6.8, where power of the controller is not considered. This assumption is valid for data dominated designs, where datapath power consumption is the major contributor to the total power of the design, as shown later in this section.

Table 6.5 summarises the actual area values shown in Figure 6.19 and actual power values shown in Figure 6.22. It can be seen that Design 2 consumes less power but requires more area than Design 1. Intuitively, a bigger area may imply that more transistors are switching resulting in higher power consumption. This can be seen when both designs are operated at the same voltage in Figure 6.22. For example, considering an operating voltage of 1.2V, Design 2 has a power consumption of 3.3mW whereas Design 1 dissipates 2.8mW. However, the combination of selected clock period and operations throughput (see Table 6.3) allowed a higher voltage scaling for Design 2 than for Design 1, i.e. 1.15V and 1.25V respectively, leading to lower power consumption. This shows the benefit of using PABCOM to analyse

different power-area tradeoffs and to obtain the solution that suits better the optimisation goal. The different power-area tradeoffs were possible due to the selection of diverse clock period and operations throughput, resulting in designs with different scheduling, allocation and binding.

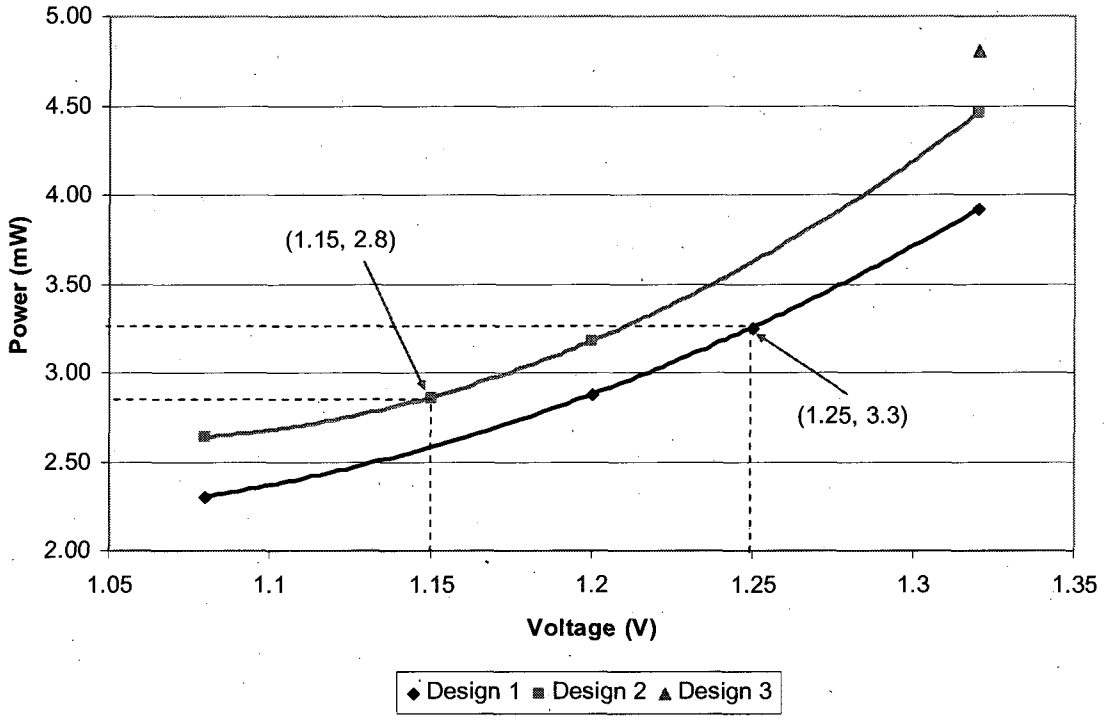


Figure 6.22 Total power consumption of Design 1 and Design 2 at different supply voltages

Table 6.5 Actual area and power values for different solutions

	Area (μm^2)	Power (mW)	Power saving	Area overhead
Design 1	33316	3.3	31%	7%
Design 2	42142	2.8	42%	30%
Design 3	31159	4.8	-	-

Table 6.5 also provides the power savings and area overhead of Design 1 and Design 2 with respect to Design 3, which was obtained using a power unaware datapath synthesis algorithm that aims only area minimisation [57]. Consequently, Design 3 is operated at the maximum supply voltage of the library, i.e. 1.32V. Note that Design 1 presents a power reduction of 31% with an area overhead of only 7%. A more efficient solution in power terms is Design 2, which reduces 42% the power dissipation but at the expense of a higher area overhead, i.e. 30%.

So far only actual values of dynamic power consumption for the designs have been presented. This is because leakage power of the designs contributes insignificantly to the total power consumption. For example, at maximum supply voltage, i.e. 1.32V, Design 2 has a leakage power dissipation of 14.6μW whereas Design 1 consumes 11.6μW. Consequently, leakage power analysis is out of the scope of this section.

Datapath and controller power consumption

To provide a better insight into the power contributions of the datapath and controller in Design 1 and Design 2, consider Figure 6.23 and Figure 6.24. As expected, most of the power is dissipated in the datapath. For example, in Design 1 and Design 2 the datapath consumes respectively 87% and 84% of the total power. It can also be seen that the datapath of Design 2 at operating voltage 1.15V consumed 20% less power than the datapath of Design 1 at operating voltage 1.25V. However, the controller of Design 2 consumes approximately 5% more power than the controller of Design 1. A possible reason for this is that the FSM that controls Design 2 has more states than the FSM that controls Design 1, increasing the area and switching activity of the controller, hence the power consumption.

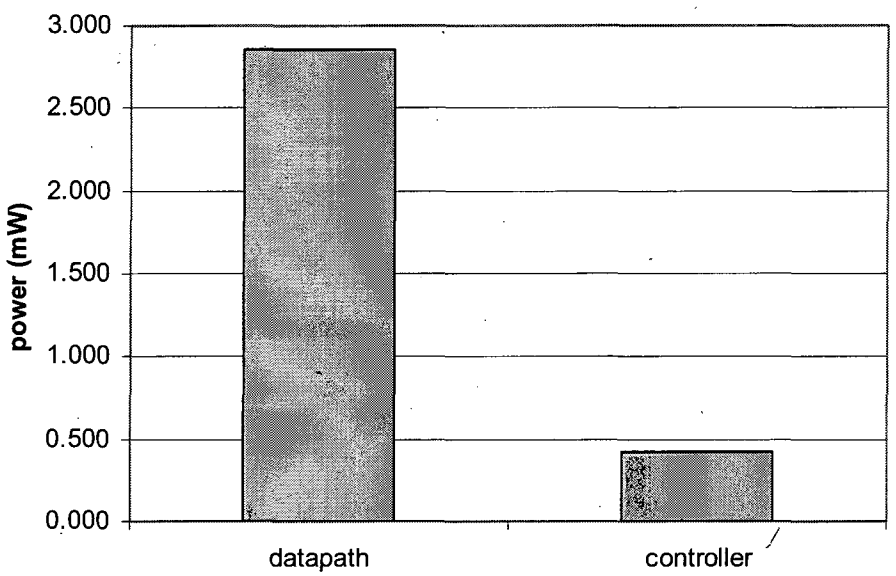


Figure 6.23 Power dissipation in Design 1

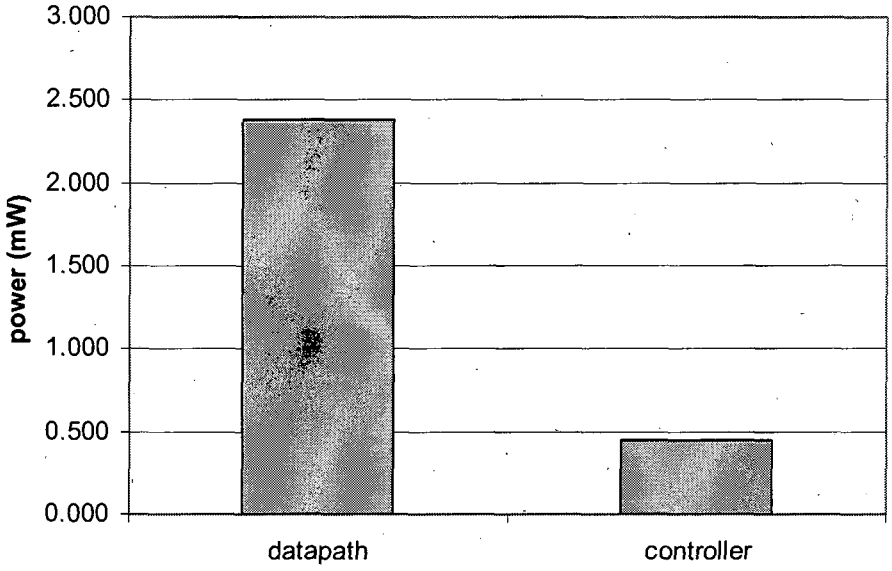


Figure 6.24 Power dissipation in Design 2

Datapath components power consumption

Power dissipation of the datapath individual components is shown in Figure 6.25 and Figure 6.26 for Design 1 and Design 2 respectively. As expected, most of the total power in both designs is consumed by the modules. An interesting point to note is that in Design 1 multiplexers consume more power than registers whereas in Design 2 registers consume more power than multiplexers. This variation of the power contribution from registers and multiplexers may be due to the change of their switching activity because of a different binding and schedule length, as shown in Section 6.3.4.

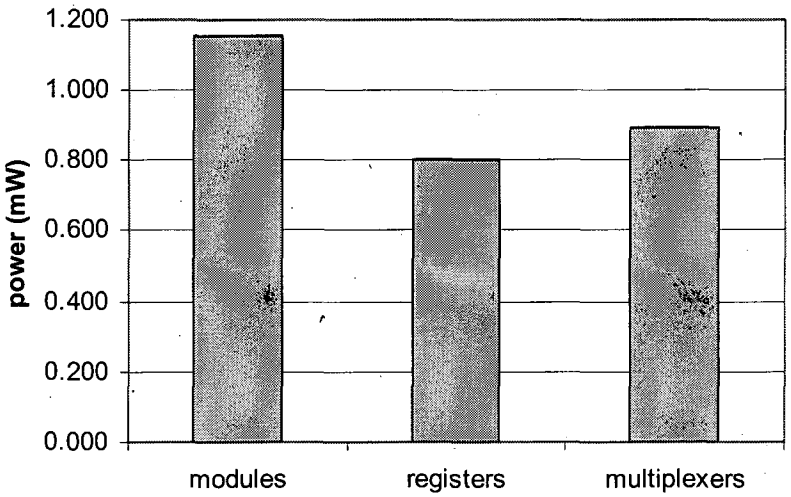


Figure 6.25 Datapath components power dissipation in Design 1

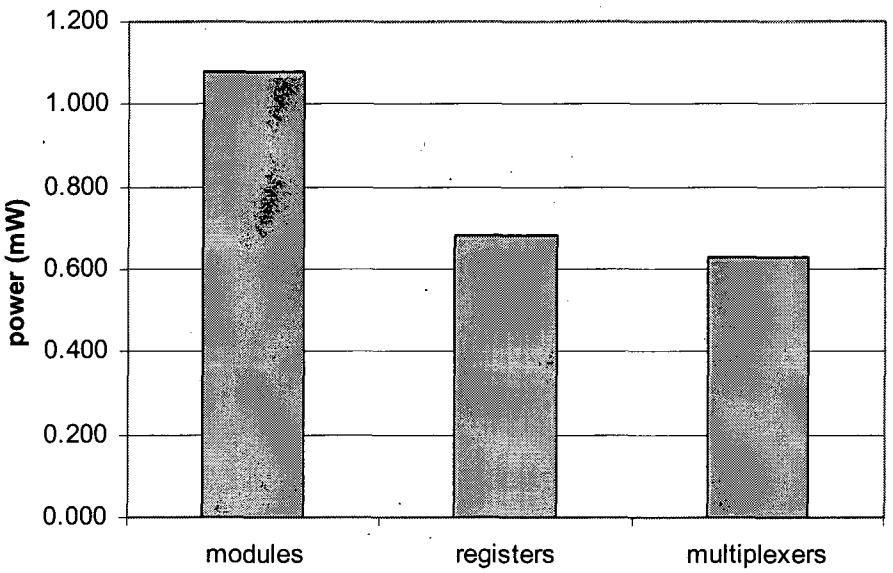


Figure 6.26 Datapath components power dissipation in Design 2

Modules power consumption

To provide a better insight into the power contribution of the modules in Design 1 and Design 2, consider now Figure 6.27 and Figure 6.28. As expected, multipliers consume more power than the other modules in both designs. This is because multiplier is the most power hungry component from the library shown in Table 6.1.

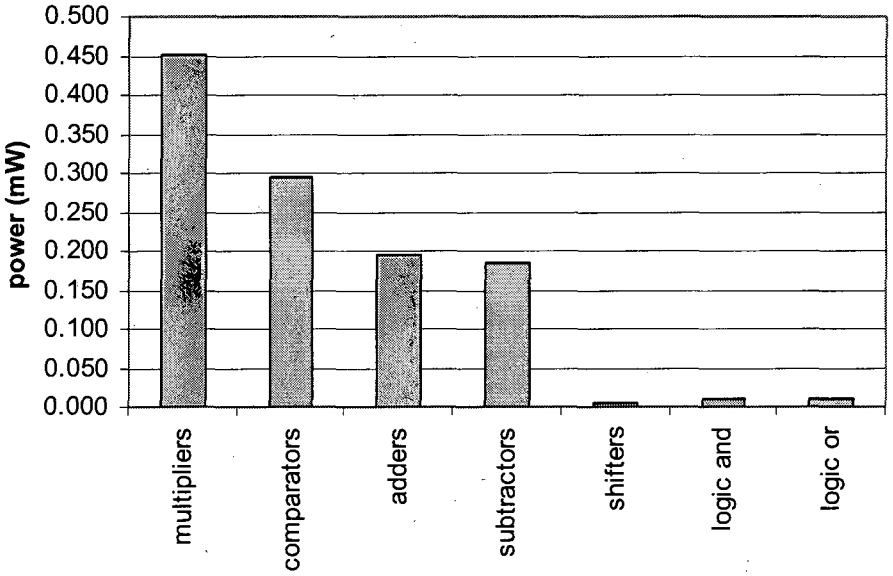


Figure 6.27 Modules power dissipation in Design 1

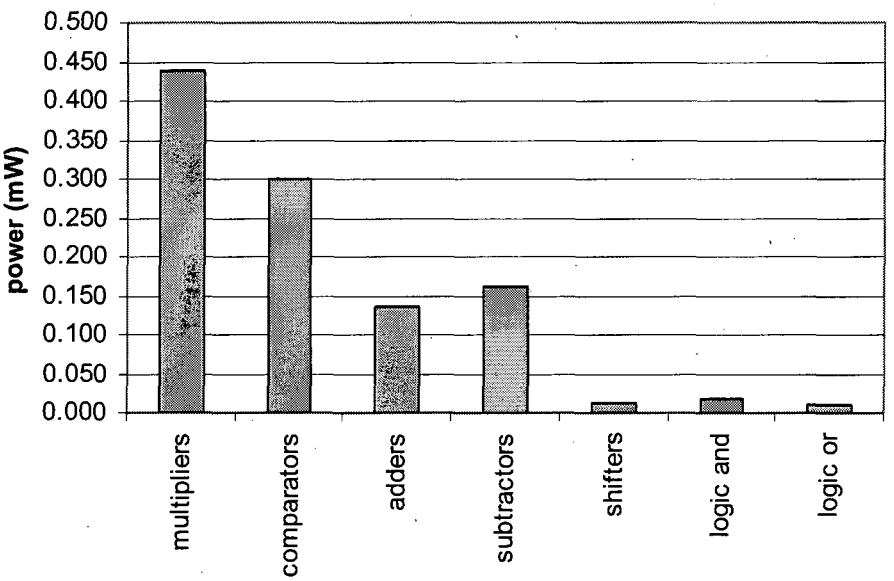


Figure 6.28 Modules power dissipation in Design 2

An interesting point to note is that in Design 1 adders consume more power than subtractors whereas in Design 2 subtractors consume slightly more power than adders. This variation of the power contribution from adders and subtractors may be due to the change of their switching activity because of a different binding and schedule length, as shown in Section 6.3.4.

6.5 Concluding remarks

It has been demonstrated that PABCOM is capable of generating good quality designs in terms of area and power when applied to a real life and complex design example such as MPEG-1 motion vector reconstructor. For example, Design 1 and Design 2 dissipate respectively 31% and 42% less power than a power unaware design operated at the maximum supply voltage of the library components, i.e. 1.32V. These two designs were implemented based on a library component that was previously synthesised using Synplify ASIC with ST 0.12μm technology library. Functional validation of these designs has been performed through timing simulation with ModelSim and comparison of the design outputs with values obtained by the C specification of the motion vector reconstructor. Both designs have undergone extensive analysis of area and power using approximated values based on analytical equations and actual values based on reports obtained after logic synthesis with Synplify ASIC and power analysis with PrimePower.

Chapter 7

Conclusions and suggestions for future research

7.1 Conclusions

It is likely that the demand for low power, low cost and high performance digital circuits will continue to increase in the future to meet the hand-held mobile devices consumer market requirements. A possible design methodology capable of delivering such circuits is low power behavioural synthesis. The work presented in this thesis has focused on the design of low power, low cost and required performance digital circuits starting from behavioural descriptions. To achieve this goal, novel algorithms have been developed that can be used within a behavioural synthesis framework to automate the design process. In particular the following key issues have been addressed:

- Low power scheduling of data dominated designs using a single supply voltage was investigated. A new power-aware time constrained scheduling algorithm based on appropriate voltage scaling after clock and operations throughput selection and capable of identifying power-area tradeoffs has been proposed.
- Development of a datapath synthesis algorithm that performs concurrently scheduling, binding, clock and operations throughput selection.
- Implementation of PABCOM, a power-aware behavioural compiler with a compound cost function that allows optimising power, area or a combination thereof, in a data dominated design given a time constraint.

- Validation of the algorithms developed in this thesis by implementing two solutions of the MPEG-1 motion vector reconstructor with different power and area requirements.

In the following, a summary of the work carried out throughout this thesis is presented. Chapter 2 has presented the foundations to design low power circuits using a behavioural synthesis methodology. The process of behavioural synthesis, its three main tasks (scheduling, allocation and binding) and interrelated tasks such as clock selection have been reviewed. The importance of power consumption in digital design as well as the key parameters that allow dynamic power reduction during behavioural synthesis has also been discussed. The main principles of combinatorial optimisation have been briefly outlined to provide the necessary background for the algorithm developed in Chapter 5.

Chapter 3 has shown that low power behavioural synthesis is an area where extensive research has already been carried out. Dynamic power optimisation algorithms in behavioural synthesis are based on either reduction of supply voltage, switching activity, frequency or a combination thereof. Reducing supply voltage has a higher impact on dynamic power consumption than reducing switching activity or frequency. This is because of the quadratic dependence of dynamic power consumption on the supply voltage. In behavioural synthesis, Multiple Supply Voltages (MSV) or single supply voltage (SSV) are two techniques used to reduce power consumption. Although a MSV technique reduce efficiently the power dissipation, it has a higher routing cost of the supply lines and area/delay overhead due to required level shifters when compared to SSV.

Chapter 4 has focused on developing a power-aware time constrained scheduling algorithm (PATICS) capable of achieving comparable power reduction to MSV but using a single supply voltage yet meeting the same time constraint. For example, some solutions obtained for DIFFEQ and EWF have respectively less than 8% and 2% power increase when compared to solutions that use MSV [126]. PATICS is also capable of obtaining a set of useful power-area tradeoffs by identifying the operating voltage after combining adequate selection of clock period and operations throughput. It has been shown that power dissipation and area have a non-linear relation that results in a large and complex search when aiming the optimisation of power, area or a combination thereof. Moreover, when compared with an area

optimised scheduler [58] using a number of benchmarks, PATICS obtain solutions that meet the same time constraint with the same resource requirements but with less power. For example, a power saving of 13% averaged over DIFFEQ, EWF and DCT was obtained. Although PATICS provides good power savings, it is important to consider the power dissipation of registers and multiplexers when searching for a solution that meets the user requirements. Hence, to find good solutions in terms of power and area, the behavioural synthesis tasks (scheduling and binding) in combination with clock and operations throughput selection need to be performed simultaneously.

Chapter 5 has described PABCOM, a Power-Aware Behavioural COMpiler that given a time constraint, considers concurrently scheduling, binding, clock and operations throughput selection using a simulated annealing based algorithm. Clock and operations throughput selection is performed using an improved version of the algorithm described in Chapter 4. A description of PABCOM is presented including its compound cost function, cooling schedule, choice of annealing parameters and performance. It has been shown that the compound cost function used by PABCOM allows obtaining optimised solutions for area, power, or a combination thereof, according to the designer specifications. For example, for DCT with a time constraint equal to 2.5 times the critical path, a tradeoff with 35% lower power than an area optimised solution and 53% less area than a power optimised solution was obtained. Extensive experimental results using benchmarks examples have further demonstrated the efficiency of the algorithm. It has also been shown that PABCOM is capable of achieving solutions in less computational time and with lower power and area than an algorithm based on previously published work [110]. For example, for a time constraint equal to 2 times the critical path, a power saving of 5% with computational time saving of 89% averaged over AR, EWF and DCT were obtained. Power reductions are mainly due to the use of lower supply voltage and/or lower frequencies, which were obtained after clock and operations throughput selection.

Chapter 6 has demonstrated the capability of PABCOM of generating good quality solutions in terms of power and area through the realisation of the motion vector reconstructor from the Berkeley MPEG-1 player [40]. Results after power-aware behavioural synthesis are presented and two designs were implemented based on a library component previously synthesised using Synplify ASIC with ST 0.12μm

technology library. Functional validation of these designs has been performed through timing simulation with ModelSim. Further validation was carried out by comparing the design outputs with values obtained by the C specification of the motion vector reconstructor [40]. Extensive analysis of area and power was carried out using approximated values based on analytical equations and actual values based on reports obtained with logic synthesis and power analysis tools such as Synplify ASIC and PrimePower. Analysis of actual values for area and power showed that the datapath is the dominant part of the designs, confirming that the study case belongs to the data dominated application domain despite the presence of conditional branches. Because of the lack of reported literature on designing a low power motion vector reconstructor has not been possible to compare. However, to give an insight into the power savings achieved by PABCOM, it has been found that Design 1 and Design 2 dissipate respectively 31% and 42% less power than a power unaware design operated at the maximum supply voltage of the library components, i.e.1.32V. In conclusion, behavioural synthesis is a viable methodology for designing power efficient complex designs. This thesis has investigated algorithms for scheduling and datapath synthesis that aim at reducing the power consumption when using a behavioural synthesis design methodology. Two new algorithms for low power time constrained scheduling and datapath synthesis have been introduced and it was demonstrated that both algorithms are capable of achieving significant power savings. Furthermore, it has been shown the influence that clock and operations throughput selection has on the operating voltage and consequently on the power consumption and area of the design. The practical applicability of the developed algorithms has been demonstrated through the low power realisation of a real life multimedia functional block such as MPEG-1 motion vector reconstructor. It is hoped that findings of this research have contributed further to the maturity of behavioural synthesis in generating low power cost effective solutions for complex design examples starting from behavioural descriptions.

7.2 Suggestions for further research

During the course of this research, a number of challenging research topics directly related to low power behavioural synthesis have been identified. A short review of these topics is given in the following.

Leakage power optimisation

Leakage power is gaining importance in nanometre technology where it can represent up to 40% of the total power consumption [62]. Moreover, predictions for future technologies show that leakage power will become the dominant component in power consumption [146]. Leakage power can be reduced during behavioural synthesis with the help of a dual voltage threshold library component, as shown in [52], [62] and [143]. Here, power reduction was achieved by identifying the frequently idle modules and replacing them with high threshold modules. Other promising technique to reduce leakage power while performing behavioural synthesis is turning off idle Multi-Threshold CMOS (MTCMOS) modules using sleep transistors. MTCMOS modules were used during register and module binding algorithm in [29] and [31]. In order to sustain performance, the sleep transistor needs to be sized to large widths, leading to a significant area overhead. This was overcome in [30] by setting an area constraint while performing the binding task using MTCMOS resources. Leakage power can also be reduced during behavioural synthesis by partitioning a circuit into islands which are powered down when its components are idle [20]. Recently, gate leakage power (gate oxide direct tunnelling) has attracted the attention of the researchers since it is one of the major components of power dissipation for a nanoCMOS of sub-65 nm technology. Gate leakage power minimisation during behavioural synthesis has been addressed in [84], [78], [82] and [81]. It would be interesting to extend PABCOM to include leakage power optimisation through the selection of a module from a library with dual voltage threshold, MTCMOS or multioxide thickness components.

Thermal and power-aware behavioural synthesis

Thermal effects are becoming an important factor in the design of integrated circuits due to their adverse influence on leakage power [88]. Recently, some efforts have been done to incorporate awareness of temperature in behavioural synthesis

algorithms. For example, binding algorithms for temperature constrained resource minimization and resource constrained temperature minimization have been proposed in [90]. Peak temperature minimisation based on uniform switching activity distribution was addressed in [89], whereas thermal-aware floorplanning information was considered in [32] and [65]. All these algorithms reduce efficiently the power consumption but do not consider how temperature may affect the selection of supply voltage(s) and threshold voltage(s) when aiming power minimisation under a given time constraint. Hence, it would be necessary to develop algorithms that consider how the behavioural synthesis tasks interact with the temperature, supply voltage(s) and threshold voltage(s).

Appendix 1

VHDL and Verilog codes

Chapter 6 presented the design of two solutions for the motion vector reconstructor using low power behavioural synthesis algorithms. Important aspects of these designs comprise the VHDL realisation of the library components (Section 6.3.2) and the generation of structural Verilog for the datapaths (Section 6.3.4).

A1.1 VHDL description of the library components

```
------------------------------ adder -------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder is
generic(n:NATURAL:=16);
port(
        x, y : in std_logic_vector(n-1 downto 0);
        z: out std_logic_vector(n-1 downto 0));
end entity adder;

architecture behaviour of adder is
begin
        add: process(x, y) is
        begin
                z <= std_logic_vector(signed(x) + signed(y));
        end process add;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_adder is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic_vector(15 downto 0);
```

```
        y: in std_logic_vector(15 downto 0);
        z: out std_logic_vector(15 downto 0));
end sync_adder;

architecture behaviour of sync_adder is
signal aux_x, aux_y: std_logic_vector(15 downto 0);
signal aux_z: std_logic_vector(15 downto 0);
signal internal_e: std_logic;

begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x, y)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.adder port map (aux_x, aux_y, aux_z);

end behaviour;


---------------------------- comparator ------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity comp is
generic(n:NATURAL:=16);
port(
        x,y : in std_logic_vector(n-1 downto 0);
        op: in std_logic_vector(2 downto 0);
        z : out std_logic);
end entity comp;

architecture behaviour of comp is
begin
        process(x, y, op) is
        begin
                case op is
                        when "000"=>
                                if (signed(x) = signed(y)) then      -- equal to
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when "001"=>
                                if (signed(x) > signed(y)) then      -- greater than
```

```
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when "010"=>
                                if (signed(x) < signed(y)) then      -- less than
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when "011"=>
                                if (signed(x) <= signed(y)) then     -- less or equal than
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when "100"=>
                                if (signed(x) >= signed(y)) then     -- greater or equal than
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when "101"=>
                                if (signed(x) /= signed(y)) then     -- different
                                        Z<='1';
                                else
                                        Z<='0';
                                end if;
                        when others=>null;
                end case;
        end process;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_comp is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic_vector(15 downto 0);
        y: in std_logic_vector(15 downto 0);
        op: in std_logic_vector(2 downto 0);
        z: out std_logic);
end sync_comp;

architecture behaviour of sync_comp is
signal aux_x, aux_y: std_logic_vector(15 downto 0);
signal aux_op: std_logic_vector(2 downto 0);
signal aux_z: std_logic;
signal internal_e: std_logic;

begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;
```

```
        latch_inp: process (internal_e, x, y, op)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                        aux_op <= op;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                        aux_op <= aux_op;
                end if;
        end process latch_inp;

        z <= aux_z;
        u0: entity work.comp port map (aux_x, aux_y, aux_op, aux_z);

end behaviour;


---------- multiplier ----------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity multiplier is
generic(n:NATURAL:=16);
port(
        x, y : in std_logic_vector(n-1 downto 0);
        z: out std_logic_vector(n-1 downto 0));
end entity multiplier;

architecture behaviour of multiplier is
begin
        multiply: process(x, y) is
        variable aux1: signed (2*n-1 downto 0);
        variable aux2: integer;
        begin
                aux1 := signed(x) * signed(y);
                aux2 := to_integer(aux1);
                z <= std_logic_vector(to_signed(aux2, n));
        end process multiply;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_multiplier is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic_vector(15 downto 0);
        y: in std_logic_vector(15 downto 0);
        z: out std_logic_vector(15 downto 0));
end sync_multiplier;

architecture behaviour of sync_multiplier is
signal aux_x, aux_y: std_logic_vector(15 downto 0);
signal aux_z: std_logic_vector(15 downto 0);
```

```
signal internal_e: std_logic;
begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x, y)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.multiplier port map (aux_x, aux_y, aux_z);

end behaviour;


------------------------------ shifter -------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity shifter is
generic(n:NATURAL:=16);
port(
        x: in std_logic_vector(n-1 downto 0);
        z : out std_logic_vector(n-1 downto 0));
end entity shifter;

architecture behaviour of shifter is
begin
        shifting: process(x) is
        begin
                z <= std_logic_vector(signed(x) sll 1);
        end process;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_shifter is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic_vector(15 downto 0);
        z: out std_logic_vector(15 downto 0));
end sync_shifter;

architecture behaviour of sync_shifter is
signal aux_x: std_logic_vector(15 downto 0);
```

```
signal aux_z: std_logic_vector(15 downto 0);
signal internal_e: std_logic;
begin
        enable_gen: process (clk, e) is
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                else
                        aux_x <= aux_x;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.shifter port map (aux_x, aux_z);

end behaviour;


------------------------------ subtractor --------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity subtractor is
generic(n:NATURAL:=16);
port(
        x, y : in std_logic_vector(n-1 downto 0);
        z: out std_logic_vector(n-1 downto 0));
end entity subtractor;

architecture behaviour of subtractor is
begin
        add: process(x, y) is
        begin
                z <= std_logic_vector(signed(x) - signed(y));
        end process add;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_subtractor is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic_vector(15 downto 0);
        y: in std_logic_vector(15 downto 0);
        z: out std_logic_vector(15 downto 0));
end sync_subtractor;

architecture behaviour of sync_subtractor is
signal aux_x, aux_y: std_logic_vector(15 downto 0);
```

```
signal aux_z: std_logic_vector(15 downto 0);
signal internal_e: std_logic;
begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x, y)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.subtractor port map (aux_x, aux_y, aux_z);

end behaviour;
```

---------- or ----------

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lgo is
port(
        x, y : in std_logic;
        z: out std_logic);
end entity lgo;

architecture behaviour of lgo is
begin
        log_or: process(x, y) is
        begin
                z <= x or y;
        end process log_or;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_lgo is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic;
        y: in std_logic;
        z: out std_logic);
end sync_lgo;

architecture behaviour of sync_lgo is
```

```
signal aux_x, aux_y: std_logic;
signal aux_z: std_logic;
signal internal_e: std_logic;
begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x, y)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.lgo port map (aux_x, aux_y, aux_z);

end behaviour;
```

---------- and ----------

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lga is
port(
        x, y : in std_logic;
        z: out std_logic);
end entity lga;

architecture behaviour of lga is
begin
        log_and: process(x, y) is
        begin
                z <= x and y;
        end process log_and;
end architecture behaviour;


library ieee;
use ieee.std_logic_1164.all;

entity sync_lga is
port(
        e: in std_logic;
        clk: in std_logic;
        x: in std_logic;
        y: in std_logic;
        z: out std_logic);
end sync_lga;
```

```
architecture behaviour of sync_lga is
signal aux_x, aux_y: std_logic;
signal aux_z: std_logic;
signal internal_e: std_logic;
begin
        enable_gen: process (clk, e) is   -- registering inputs
        begin
                if (rising_edge(clk)) then
                        internal_e <= e;
                end if;
        end process enable_gen;

        latch_inp: process (internal_e, x, y)
        begin
                if (internal_e = '1') then
                        aux_x <= x;
                        aux_y <= y;
                else
                        aux_x <= aux_x;
                        aux_y <= aux_y;
                end if;
        end process;

        z <= aux_z;
        u0: entity work.lga port map (aux_x, aux_y, aux_z);

end behaviour;
```

A1.2 Structural Verilog for Design 1 and Design 2

```
////////////////////////////////////////////////// Design 1//////////////////////////////////////////////////////////
// module for the whole datapath
module datapath (
  rst, clk,
  eM1, eM2, eM7, eM12, eM13, eM18, eM19, eM21, eM22, eM39,
  ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13, ldR14,
  ldR15, ldR19, ldR20, ldR22,
  ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,
  opM21, opM22,
  sM1a, sM1b, sM2a, sM7a, sM12a, sM12b, sM13a, sM13b, sM19a, sM21a, sM21b, sM22b,
  sM39a, sR1a, sR2a, sR3a, sR4a, sR7a, sR11a, sR13a, sR14a, sR19a, sR22a,
  mhc, mhr, mvc, mvr, f, fp, cero, one, sixteen, _sixteen, thirty2,
  rrprev_i, rdprev_i,
  rrprev_o, rdprev_o, rrp_o, rdp_o,
  stR1o, stR2o, stR3o, stR4o, stR5o, stR6o, stR7o, stR8o, stR9o, stR10o
);
input rst;
input clk;
input eM1, eM2, eM7, eM12, eM13, eM18, eM19, eM21, eM22, eM39;
input ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
ldR14,
    ldR15, ldR19, ldR20, ldR22;
input ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10;
input [2:0] opM21, opM22;
input [1:0] sM1a, sM1b, sM13a, sM13b, sM21a;
input sM2a, sM12a, sM12b, sM19a, sM21b, sM22b, sM39a, sR7a, sR11a, sR13a, sR14a,
    sR19a, sR22a;
```

```
input [2:0] sM7a, sR1a, sR2a, sR3a, sR4a;
input [15:0] mhc, mhr, mvc, mvr, f, fp, cero, one, sixteen, _sixteen, thirty2;
input [15:0] rrprev_i, rdprev_i;
output [15:0] rrprev_o, rdprev_o, rrp_o, rdp_o;
output stR1o, stR2o, stR3o, stR4o, stR5o, stR6o, stR7o, stR8o, stR9o, stR10o;

// module inputs
wire [15:0] M1ao, M1bo, M2ao, M7ao, M12ao, M12bo, M13ao, M13bo, M19ao, M21ao,
            M21bo, M22bo;
wire M39ao;
// module outputs
wire [15:0] M1o, M2o, M7o, M12o, M13o, M19o;
wire M18o, M21o, M22o, M39o;
// register inputs
wire [15:0] R1ao, R2ao, R3ao, R4ao, R7ao, R11ao, R13ao, R14ao;
wire R21ao, R19ao, R22ao;
// register outputs
wire [15:0] R5o, R6o, R7o, R8o, R9o, R10o, R11o, R12o, R13o, R14o, R15o;
wire R19o, R20o, R22o;

  sync_adder       M1  (eM1, clk, M1ao, M1bo, M1o);
  sync_adder       M2  (eM2, clk, M2ao, M12ao, M2o);
  sync_multiplier  M7  (eM7, clk, M7ao, R10o, M7o);
  sync_subtractor M12 (eM12, clk, M12ao, M12bo, M12o);
  sync_subtractor M13 (eM13, clk, M13ao, M13bo, M13o);
  sync_lga        M18 (eM18, clk, R22o, R19o, M18o);
  sync_shifter     M19 (eM19, clk, M19ao, M19o);
  sync_comp        M21 (eM21, clk, M21ao, M21bo, opM21, M21o);
  sync_comp        M22 (eM22, clk, rrprev_o, M22bo, opM22, M22o);
  sync_lgo         M39 (eM39, clk, M39ao, R19o, M39o);

  registro R1 (R1ao, clk, rst, ldR1, rrprev_o);
  registro R2 (R2ao, clk, rst, ldR2, rdprev_o);
  registro R3 (R3ao, clk, rst, ldR3, rrp_o);
  registro R4 (R4ao, clk, rst, ldR4, rdp_o);
  registro R5 (mvr, clk, rst, ldR5, R5o);
  registro R6 (cero, clk, rst, ldR6, R6o);
  registro R7 (R7ao, clk, rst, ldR7, R7o);
  registro R8 (fp, clk, rst, ldR8, R8o);
  registro R9 (rdprev_i, clk, rst, ldR9, R9o);
  registro R10 (f, clk, rst, ldR10, R10o);
  registro R11 (R11ao, clk, rst, ldR11, R11o);
  registro R12 (thirty2, clk, rst, ldR12, R12o);
  registro R13 (R13ao, clk, rst, ldR13, R13o);
  registro R14 (R14ao, clk, rst, ldR14, R14o);
  registro R15 (rrprev_i, clk, rst, ldR15, R15o);
  registro_1b R19 (R19ao, clk, rst, ldR19, R19o);
  registro_1b R20 (M21o, clk, rst, ldR20, R20o);
  registro_1b R22 (R22ao, clk, rst, ldR22, R22o);

  // status registers, outputs to controller...
  registro_1b stR1 (M39o, clk, rst, ldstR1, stR1o);
  registro_1b stR2 (M22o, clk, rst, ldstR2, stR2o);
  registro_1b stR3 (M21o, clk, rst, ldstR3, stR3o);
  registro_1b stR4 (M18o, clk, rst, ldstR4, stR4o);
  registro_1b stR5 (M21o, clk, rst, ldstR5, stR5o);
  registro_1b stR6 (M39o, clk, rst, ldstR6, stR6o);
  registro_1b stR7 (M21o, clk, rst, ldstR7, stR7o);
  registro_1b stR8 (M21o, clk, rst, ldstR8, stR8o);
  registro_1b stR9 (M18o, clk, rst, ldstR9, stR9o);
```

```
  registro_1b stR10 (M21o, clk, rst, ldstR10, stR10o);

  mux5 R1a (mhc, M7o, M1o, M13o, cero, sR1a, R1ao);
  mux5 R2a (mhr, M12o, M2o, M1o, cero, sR2a, R2ao);
  mux6 R3a (M13o, M7o, M2o, M19o, cero, rrprev_o, sR3a, R3ao);
  mux5 R4a (sixteen, M1o, M13o, M19o, rdprev_o, sR4a, R4ao);
  mux2 R7a (one, M13o, sR7a, R7ao);
  mux2 R11a (mvc, M7o, sR11a, R11ao);
  mux2 R13a (_sixteen, M7o, sR13a, R13ao);
  mux2 R14a (M12o, cero, sR14a, R14ao);
  mux2_1b R19a (M21o, M22o, sR19a, R19ao);
  mux2_1b R22a (M22o, M21o, sR22a, R22ao);

  mux4 M1a (R14o, R15o, R9o, rrprev_o, sM1a, M1ao);
  mux4 M1b (rrprev_o, rdp_o, rdprev_o, rrp_o, sM1b, M1bo);
  mux2 M2a (rdprev_o, rdp_o, sM2a, M2ao);
  mux5 M7a (rrprev_o, R11o, R13o, rdp_o, R12o, sM7a, M7ao);
  mux2 M12a (rrp_o, R11o, sM12a, M12ao);
  mux2 M12b (rdprev_o, R5o, sM12b, M12bo);
  mux4 M13a (R10o, rrprev_o, rdp_o, rdprev_o, sM13a, M13ao);
  mux4 M13b (R7o, R14o, rrp_o, rrprev_o, sM13b, M13bo);
  mux2 M19a (rdprev_o, rrprev_o, sM19a, M19ao);
  mux4 M21a (R10o, R11o, rrprev_o, R8o, sM21a, M21ao);
  mux2 M21b (R7o, R6o, sM21b, M21bo);
  mux2 M22b (R6o, R13o, sM22b, M22bo);
  mux2_1b M39a (R22o, R20o, sM39a, M39ao);

endmodule


//////////////////////////////////////////////////// Design 2 ////////////////////////////////////////////////////////////
module datapath (
  rst, clk,
  eM2, eM4, eM6, eM7, eM13, eM14, eM18, eM19, eM20, eM21, eM22, eM39, eM40,
  eM43, eM44,     // enable signals for modules
  ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13, ldR14,
  ldR15, ldR16, ldR18, ldR21, ldR22, ldR23,   // load signals for registers
  ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10,    // load
  signals for status registers
  opM21, opM22, opM43, opM44,   // operation that the comparator will execute
  sM2a, sM2b, sM4a, sM4b, sM6a, sM7b, sM13a, sM13b, sM14a, sM14b, sM21a, sM39a,
  sM39b, sM44a, sM44b,
  sR1a, sR2a, sR3a, sR4a, sR6a, sR9a, sR11a, sR12a, sR23a,
  mhc, mhr, mvc, mvr, f, fp, cero, one, sixteen, _sixteen, thirty2,
  rrprev_i, rdprev_i,
  rrprev_o, rdprev_o, rrp_o, rdp_o,
  stR1o, stR2o, stR3o, stR4o, stR5o, stR6o, stR7o, stR8o, stR9o, stR10o
);
input rst;
input clk;
input eM2, eM4, eM6, eM7, eM13, eM14, eM18, eM19, eM20, eM21, eM22, eM39, eM40,
eM43, eM44;
input ldR1, ldR2, ldR3, ldR4, ldR5, ldR6, ldR7, ldR8, ldR9, ldR10, ldR11, ldR12, ldR13,
ldR14, ldR15, ldR16, ldR18, ldR21, ldR22, ldR23;
input ldstR1, ldstR2, ldstR3, ldstR4, ldstR5, ldstR6, ldstR7, ldstR8, ldstR9, ldstR10;
input [2:0] opM21, opM22, opM43, opM44;
input [1:0] sM7b, sM13a, sM13b, sM14a;
input sM2a, sM2b, sM4a, sM4b, sM6a, sM14b, sM21a, sM39a, sM39b, sM44a, sM44b,
sR6a, sR9a, sR11a, sR12a, sR23a;
input [2:0] sR1a, sR2a, sR3a, sR4a;
```

```
input [15:0] mhc, mhr, mvc, mvr, f, fp, cero, one, sixteen, _sixteen, thirty2;
input [15:0] rrprev_i, rdprev_i;
output [15:0] rrprev_o, rdprev_o, rrp_o, rdp_o;
output stR1o, stR2o, stR3o, stR4o, stR5o, stR6o, stR7o, stR8o, stR9o, stR10o;

// module inputs
wire [15:0] M2ao, M2bo, M4ao, M4bo, M6ao, M7bo, M13ao, M13bo, M14ao, M14bo, M21ao,
M44ao, M44bo;
wire M39ao, M39bo;
// module outputs
wire [15:0] M2o, M4o, M6o, M7o, M13o, M14o, M19o, M20o;
wire M18o, M21o, M22o, M39o, M40o, M43o, M44o;
// register inputs
wire [15:0] R1ao, R2ao, R3ao, R4ao, R6ao, R9ao, R11ao, R12ao;
wire R23ao;
// register outputs
wire [15:0] R1o, R2o, R3o, R4o, R5o, R6o, R7o, R8o, R9o, R10o, R11o, R12o, R13o, R14o,
R15o;
wire R16o, R18o, R21o, R22o, R23o;

  sync_adder      M2  (eM2, clk, M2ao, M2bo, M2o);
  sync_adder      M4  (eM4, clk, M4ao, M4bo, M4o);
  sync_multiplier M6  (eM6, clk, M6ao, R5o, M6o);
  sync_multiplier M7  (eM7, clk, R5o, M7bo, M7o);
  sync_subtractor M13 (eM13, clk, M13ao, M13bo, M13o);
  sync_subtractor M14 (eM14, clk, M14ao, M14bo, M14o);
  sync_lga        M18 (eM18, clk, R18o, R23o, M18o);
  sync_shifter    M19 (eM19, clk, rrprev_o, M19o);
  sync_shifter    M20 (eM20, clk, rdprev_o, M20o);
  sync_comp       M21 (eM21, clk, M21ao, R9o, opM21, M21o);
  sync_comp       M22 (eM22, clk, rrprev_o, R9o, opM22, M22o);
  sync_lgo        M39 (eM39, clk, M39ao, M39bo, M39o);
  sync_lga        M40 (eM40, clk, R22o, R16o, M40o);
  sync_comp       M43 (eM43, clk, rrprev_o, R12o, opM43, M43o);
  sync_comp       M44 (eM44, clk, M44ao, M44bo, opM44, M44o);

  registro R1 (R1ao, clk, rst, ldR1, rrprev_o);
  registro R2 (R2ao, clk, rst, ldR2, rdprev_o);
  registro R3 (R3ao, clk, rst, ldR3, rrp_o);
  registro R4 (R4ao, clk, rst, ldR4, rdp_o);
  registro R5 (f, clk, rst, ldR5, R5o);
  registro R6 (R6ao, clk, rst, ldR6, R6o);
  registro R7 (rrprev_i, clk, rst, ldR7, R7o);
  registro R8 (rdprev_i, clk, rst, ldR8, R8o);
  registro R9 (R9ao, clk, rst, ldR9, R9o);
  registro R10 (one, clk, rst, ldR10, R10o);
  registro R11 (R11ao, clk, rst, ldR11, R11o);
  registro R12 (R12ao, clk, rst, ldR12, R12o);
  registro R13 (thirty2, clk, rst, ldR13, R13o);
  registro R14 (M13o, clk, rst, ldR14, R14o);
  registro R15 (M14o, clk, rst, ldR15, R15o);
  registro_1b R16 (M22o, clk, rst, ldR16, R16o);
  registro_1b R18 (M44o, clk, rst, ldR18, R18o);
  registro_1b R21 (M21o, clk, rst, ldR21, R21o);
  registro_1b R22 (M43o, clk, rst, ldR22, R22o);
  registro_1b R23 (R23ao, clk, rst, ldR23, R23o);

  mux2 M2a (rrp_o, R8o, sM2a, M2ao);
  mux2 M2b (rdp_o, rrp_o, sM2b, M2bo);
  mux2 M4a (rdprev_o, rrprev_o, sM4a, M4ao);
```

```
  mux2 M4b (rrprev_o, R7o, sM4b, M4bo);
  mux2 M6a (rrprev_o, R13o, sM6a, M6ao);
  mux4 M7b (rrp_o, R12o, R11o, R13o, sM7b, M7bo);
  mux4 M13a (R5o, R14o, rrp_o, rdprev_o, sM13a, M13ao);
  mux4 M13b (R10o, rdp_o, rrprev_o, cero, sM13b, M13bo);
  mux4 M14a (R5o, R15o, rrprev_o, R11o, sM14a, M14ao);
  mux2 M14b (R10o, rdprev_o, sM14b, M14bo);
  mux2 M21a (rrp_o, R6o, sM21a, M21ao);
  mux2_1b M39a (R21o, R18o, sM39a, M39ao);
  mux2_1b M39b (R18o, R23o, sM39b, M39bo);
  mux2 M44a (R5o, R6o, sM44a, M44ao);
  mux2 M44b (R10o, R12o, sM44b, M44bo);

  mux5 R1a (mhc, M6o, M4o, M13o, cero, sR1a, R1ao);
  mux5 R2a (mhr, M14o, M4o, M2o, cero, sR2a, R2ao);
  mux6 R3a (mvc, M7o, M2o, M13o, M19o, rrprev_o, sR3a, R3ao);
  mux8 R4a (mvr, M13o, M7o, M2o, M20o, cero, rdprev_o, cero, sR4a, R4ao);
  mux2 R6a (fp, M2o, sR6a, R6ao);
  mux2 R9a (cero, M14o, sR9a, R9ao);
  mux2 R11a (sixteen, M7o, sR11a, R11ao);
  mux2 R12a (_sixteen, M7o, sR12a, R12ao);
  mux2_1b R23a (M22o, M21o, sR23a, R23ao);

  // status registers, outputs to controller...
  registro_1b stR1 (M39o, clk, rst, ldstR1, stR1o);
  registro_1b stR2 (M22o, clk, rst, ldstR2, stR2o);
  registro_1b stR3 (M22o, clk, rst, ldstR3, stR3o);
  registro_1b stR4 (M40o, clk, rst, ldstR4, stR4o);
  registro_1b stR5 (M21o, clk, rst, ldstR5, stR5o);
  registro_1b stR6 (M39o, clk, rst, ldstR6, stR6o);
  registro_1b stR7 (M21o, clk, rst, ldstR7, stR7o);
  registro_1b stR8 (M21o, clk, rst, ldstR8, stR8o);
  registro_1b stR9 (M18o, clk, rst, ldstR9, stR9o);
  registro_1b stR10 (M21o, clk, rst, ldstR10, stR10o);

endmodule
```

Appendix 2

Area-Delay-Power characterisation

This appendix describes the methodology followed for the power-area-delay characterisation of the library components of Table 5.9 and Table 6.1. The flow diagram for the power-area-delay characterisation process is shown in Figure A2.1.

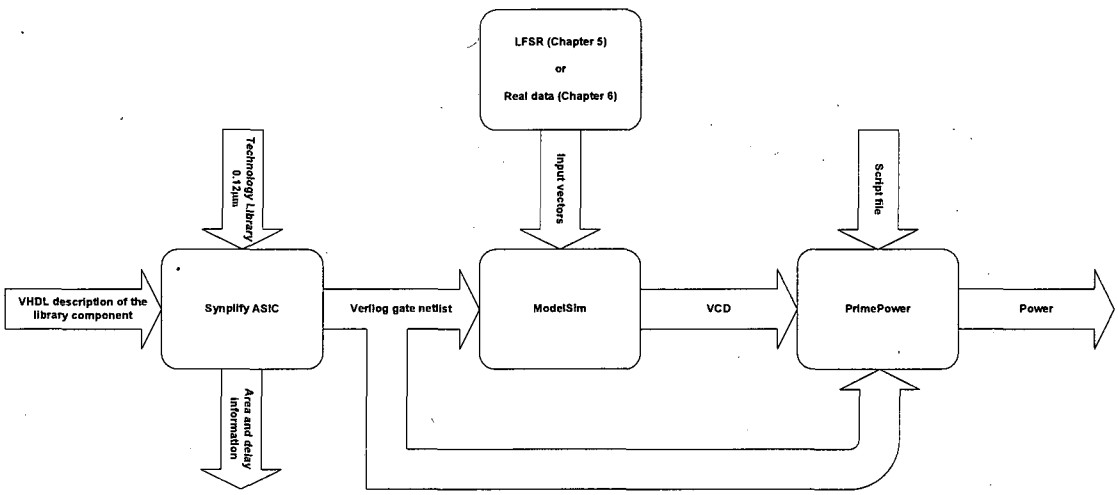


Figure A2.1 Area-delay-power characterisation flow

The area-delay characterisation requires only the application of VHDL description of the library components and Technology library file to the logic synthesis tool Synplify ASIC [128]. The 0.12μm technology library used is CORE9GPLL provided by ST Microelectronics and it is composed of three files: CORE9GPLL_Best.lib, CORE9GPLL_Nom.lib and CORE9GPLL_Worst.lib. CORE9GPLL is a low leakage standard-cell library for HCMOS9 VLSI digital designs that is designed to work at 1.2V (+10%/-10%), as shown in Table A2.1 [132].

Table A2.1 Operating voltages for Technology library CORE9GPLL

	CORE9GPLL_Best.lib	CORE9GPLL_Nom.lib	CORE9GPLL_Worst.lib
V (V)	1.32	1.2	1.08

After performing logic synthesis with each file of CORE9GPLL, the generated report contains information about the area and delay of the component. The delay of the library component after synthesis depends on the technology library file used for synthesis, since each library file has a different operating voltage as shown in Table A2.1. The area of the component is exactly the same in the three cases since the same number and type of standard cells is used. The result after logic synthesis is a Verilog netlist that is used together with a VCD and script files to perform power analysis with PrimePower.

The Value Change Dump (VCD) file can be obtained after simulation with ModelSim [75] using the Verilog netlist and a testbench that provides the input vectors. In the case of Chapter 5, the input vectors correspond to pseudo random numbers generated by a LSFR, whereas in Chapter 6 the input vectors correspond to real data obtained after playing a video with the MPEG-1 Berkeley decoder [40]. The generated VCD is an ASCII file that contains information about simulation time, scope and signal definitions, and signal value changes in the simulation run [43]. VCD file can be generated using verilog's $dumpvar command. Some examples are shown in Table A2.2.

Table A2.2 Verilog commands to generate VCD file

Command	*Function*
Initial $dumpfile("filename.vcd");	// selects this dump file name
Initial $dumpvars;	// dumps all signal info
Initial $dumpvars(0, top);	// dumps all the variables in the hierarchy from top. Here top is an instance name. 0 indicates every level from top.

In addition to the VCD file, a script file is necessary to perform power analysis with PrimePower. An example of the script file (saved as .tcl) is shown in Listing A2.1, where *search_path* specifies the path for the synopsys library, *link_library* specifies the library to be used, i.e. CORE9GPLL_Best.db. This technology file is the same that the one used for synthesising the design but in different format. For example, if

Synplify ASIC has been used for synthesis using the "CORE9GPLL_Best.lib", PrimePower requires CORE9GPLL_Best.db. In Listing A2.1, the command *read_verilog* specifies the Verilog file name including path, and *current_design* indicates the top_level design name. Similarly *read_vcd* specifies the VCD file including path. Note that *read_vcd* command specifies the *strip_path*. This path is the test_bench/top_instance if VCD is generated from test_bench else the path will be top_instance if VCD is generated from top module under test. *set_input_transition* defines a fixed transition time for input ports. Output files can be specified in *set_waveform_options* and *report_power_new* options [43].

Listing A2.1 Example of script file to perform power analysis using PrimePower

```
#adder
#########################################################################
#     link design
#########################################################################
set search_path  "library/library"
set link_library   "* CORE9GPLL_Best.db"

read_verilog      {/ library/adder/best_add/sync_adder.vma}
current_design    sync_adder
link

#########################################################################
#     read switching activity file
#########################################################################
read_vcd -strip_path adder_tb/u0/library/adder/best_add/ sync_adder.vcd

#########################################################################
#     set transition time / annotate parasitics
#########################################################################
set_input_transition        0.1 [all_inputs]

#########################################################################
#     power analysis
#########################################################################
cd /library/adder/best_add
set_waveform_options      -interval 0.01 -file sync_adder -format fsdb
calculate_power           -waveform -statistics
report_power_new          -file sync_adder
remove_design             -all
```

Appendix 3

Input and output files of PABCOM

The input to PABCOM is a standard text file that contains a data flow graph (DFG) with the corresponding user-specified constraints and optimisation parameters. This text file is split into sections as can be seen from the example shown in Listing A3.1. The first part of this file specifies the name of the design and gives the option to save the resultant binding, multiplexers assignment and datapath netlist into a file. In the second part the synthesis constraints and annealing parameters are specified whilst the third part describes the data flow graph. First the DFG inputs are given, followed by the operations with their input values to determine the data dependencies, and finally the outputs are presented. The last part of the input file describes the available functional modules and registers.

Listing A3.1 Input file to PABCOM

```
#config file for DCT benchmark
[STATUS]
Design          Discrete Cosine Transform
results_to_file 1       # 1 save results into a file, 0 print results in screen

[CONFIG]
time            47      # time constraint
min_clk         2       # minimum clock period
alpha           0.1     # optimisation power weight
operations      48
multiplications 16
additions       32
inputs          8
outputs         8
modules         26
registers       30
                        # annealing parameters
x0              0.95    # initial acceptance ratio
d               0.2     # distance parameter
es              0.0001  # stop criterion
```

```
################## DFG in text form ##################
#name  type        inputs
[OPERATIONS]
D_0    input
D_1    input
D_2    input
D_3    input
D_4    input
D_5    input
D_6    input
D_7    input
N_16   add         D_2   D_5
N_17   add         D_1   D_6
N_18   add         D_3   D_4
N_19   add         D_0   D_7
N_20   add         D_2   D_5
N_21   add         D_1   D_6
N_22   add         D_3   D_4
N_23   add         D_0   D_7
N_24   add         N_16  N_17
N_25   add         N_16  N_17
N_26   add         N_18  N_19
N_27   add         N_18  N_19
N_28   add         N_20  N_21
N_29   add         N_22  N_23
N_42   multiply    N_20
N_44   multiply    N_20
N_48   multiply    N_21
N_50   multiply    N_21
N_54   multiply    N_22
N_56   multiply    N_22
N_60   multiply    N_23
N_62   multiply    N_23
N_30   add         N_24  N_26
N_31   add         N_28  N_29
N_32   multiply    N_24
N_36   multiply    N_26
N_38   multiply    N_25
N_40   multiply    N_27
N_46   multiply    N_28
N_58   multiply    N_29
N_34   multiply    N_30
N_52   multiply    N_31
N_64   add         N_42  N_46
N_66   add         N_46  N_48
N_69   add         N_56  N_58
N_71   add         N_58  N_62
N_72   add         N_38  N_40
N_76   add         N_38  N_40
N_65   add         N_52  N_60
N_67   add         N_52  N_54
N_68   add         N_50  N_52
N_70   add         N_44  N_52
N_74   add         N_34  N_36
N_78   add         N_32  N_34
N_73   add         N_70  N_71
N_75   add         N_66  N_67
N_77   add         N_64  N_65
N_79   add         N_68  N_69
```

```
P_0     output          N_72
P_1     output          N_73
P_2     output          N_74
P_3     output          N_75
P_4     output          N_76
P_5     output          N_77
P_6     output          N_78
P_7     output          N_79
##################### end of DFG ##################

[MODULES]
# name type
M_1   add
M_2   add
M_3   add
M_4   add
M_5   add
M_6   add
M_7   add
M_8   add
M_9   add
M_10  add
M_11  multiply
M_12  multiply
M_13  multiply
M_14  multiply
M_15  multiply
M_16  multiply
M_17  multiply
M_18  multiply
M_19  multiply
M_20  multiply
M_21  multiply
M_22  multiply
M_23  multiply
M_24  multiply
M_25  multiply
M_26  multiply

[REGISTERS]
# name
R_01
R_02
R_03
R_04
R_05
R_06
R_07
R_08
R_09
R_10
R_11
R_12
R_13
R_14
R_15
R_16
R_17
R_18
R_19
```

R_20
R_21
R_22
R_23
R_24
R_25
R_26
R_27
R_28
R_29
R_30

The output of PABCOM is a text file that contains the information necessary to generate the datapath and controller of the design. For example, Listing A3.2 and Listing A3.3 present the output files produced by PABCOM after using the input file from Listing A3.1 with $\alpha = 0.1$ and $\alpha = 0.4$. In both output files, four parts can be identified: module binding, register binding, multiplexers assignment and datapath netlist. It can be seen that the first three parts describe the cycle by cycle behaviour of the design. For example in the module binding of Listing A3.2, module M_2 executes operations N_20 in cstep 1 and N_16 in cstep 2, and becomes idle in csteps 3 and 4. In the register binding, input D1 is loaded into register R_13 in cstep 1 and held during cstep 2. Then R_13 is idle in cstep 3 and value N_50 is loaded in cstep 4. A graphical representation of this module binding and register binding was shown respectively in Figure 5.21 and Figure 5.23 (Chapter 5). In the multiplexers assignment of Listing A3.2, M_1_a is a 3-input multiplexer whose inputs are R_13, R05 and R_10. In cstep 1 and 2, the input R_13 is selected, in cstep 3 the input R_5 is selected and then R_10 is selected in cstep 4. It can be seen that M1_b only has one input, i.e. R_02, this means a direct connection from the register R_02 to the input of module M_1. The timing information provided by the binding and multiplexers assignment allows the immediate definition of the control signals in the required finite state machine.

The last part of the output file is the datapath netlist, which specifies the interconnections between modules, registers and multiplexers. For example, in the datapath netlist of Listing A3.2, the inputs of module M7 are connected to multiplexers M_7_a and M_7_b. The inputs of multiplexer M_7_a are connected to registers R_08 and R_07. The input of register R_08 is connected to multiplexer R_08_a whose inputs are connected to modules M_3 and M_7. A graphical representation of this datapath netlist was shown in Figure 5.25 (Chapter 5).

Listing A3.2 Output file produced by PABCOM for DCT with $\alpha = 0.1$

```
################## Module binding ####################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7     cs8     cs9
Module:M_1
N_21    N_17    N_31    N_64    N_71    N_69    N_76    N_72    ./.
Module:M_2
N_20    N_16    ./.     ./.     N_70    ./.     N_65    N_78    ./.
Module:M_3
N_23    N_28    N_19    N_27    N_26    N_30    N_75    N_77    ./.
Module:M_7
./.     N_29    ./.     N_66    N_68    N_67    N_79    N_74    ./.
Module:M_10
N_22    N_18    N_24    ./.     N_25    N_73    ./.     ./.     ./.
Module:M_11
./.     N_42    ./.     N_58    N_56    N_40    ./.     ./.     ./.
Module:M_13
./.     N_48    N_50    N_52    N_54    N_36    N_34    ./.     ./.
Module:M_15
./.     N_44    N_46    N_62    N_60    N_38    N_32    ./.     ./.

################## Register binding  ####################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7     cs8     cs9
Register:       R_02
D_6     D_6     N_28    N_46    N_62    N_56    N_38    N_38    ./.
Register:       R_05
./.     ./.     N_29    N_29    N_66    N_66    N_66    N_79    N_79
Register:       R_06
./.     N_22    N_22    N_22    N_22    ./.     N_73    N_73    N_73
Register:       R_07
D_2     D_2     N_48    N_48    N_52    N_52    N_52    N_34    ./.
Register:       R_08
./.     N_23    N_23    N_23    N_23    N_68    N_68    ./.     N_74
Register:       R_10
./.     ./.     N_42    N_42    N_58    N_58    N_40    N_40    ./.
Register:       R_12
D_7     D_7     D_7     N_24    N_24    N_24    N_24    ./.     ./.
Register:       R_13
D_1     D_1     ./.     N_50    N_50    N_54    N_36    N_36    ./.
Register:       R_17
D_4     D_4     N_16    N_16    N_16    N_70    ./.     ./.     N_78
Register:       R_18
D_5     D_5     N_44    N_44    N_44    N_60    N_60    N_32    ./.
Register:       R_19
./.     N_20    N_18    N_18    N_18    N_25    N_67    N_65    ./.
Register:       R_20
./.     ./.     ./.     ./.     N_27    N_27    ./.     N_75    N_75
Register:       R_21
D_3     D_3     N_17    N_17    N_17    N_71    N_69    N_76    N_76
```

```
Register:       R_23
./.     N_21    N_21    N_31    N_64    N_64    N_64    N_64    N_72
Register:       R_29
D_0     D_0     D_0     N_19    N_19    N_26    N_30    ./.     N_77

#################### Multiplexers ######################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7     cs8     cs9
Mux:    M_1_a, inputs: R_13     R_05    R_10
R_13    R_13    R_05    R_10    R_10    R_10    R_10    R_10    ./.
Mux:    M_1_b , inputs: R_02
R_02    R_02    R_02    R_02    R_02    R_02    R_02    R_02    ./.
Mux:    M_2_a, inputs: R_18
R_18    R_18    ./.     ./.     R_18    ./.     R_18    R_18    ./.
Mux:    M_2_b , inputs: R_07
R_07    R_07    ./.     ./.     R_07    ./.     R_07    R_07    ./.
Mux:    M_3_a, inputs: R_29     R_23    R_05
R_29    R_23    R_29    R_29    R_29    R_29    R_05    R_23    ./.
Mux:    M_3_b , inputs: R_12    R_19
R_12    R_19    R_12    R_19    R_19    R_12    R_19    R_19    ./.
Mux:    M_7_a, inputs: R_08     R_07
./.     R_08    ./.     R_07    R_07    R_07    R_08    R_07    ./.
Mux:    M_7_b, inputs: R_06     R_02    R_13    R_21
./.     R_06    ./.     R_02    R_13    R_13    R_21    R_13    ./.
Mux:    M_10_a, inputs: R_21
R_21    R_21    R_21    ./.     R_21    R_21    ./.     ./.     ./.
Mux:    M_10_b, inputs: R_17
R_17    R_17    R_17    ./.     R_17    R_17    ./.     ./.     ./.
Mux:    M_11_a, inputs: R_19    R_05    R_06    R_20
./.     R_19    ./.     R_05    R_06    R_20    ./.     ./.     ./.
Mux:    M_13_a, inputs: R_23    R_06    R_29
./.     R_23    R_23    R_23    R_06    R_29    R_29    ./.     ./.
Mux:    M_15_a, inputs: R_19    R_02    R_08    R_12
./.     R_19    R_02    R_08    R_08    R_19    R_12    ./.     ./.
Mux:    R_02_a, inputs: I/O     M_3     M_15    M_11
I/O     ./.     M_3     M_15    M_15    M_11    M_15    ./.     ./.
Mux:    R_05_a, inputs: M_7
./.     ./.     M_7     ./.     M_7     ./.     ./.     M_7     ./.
Mux:    R_06_a, inputs: M_10
./.     M_10    ./.     ./.     ./.     ./.     M_10    ./.     ./.
Mux:    R_07_a, inputs: I/O     M_13
I/O     ./.     M_13    ./.     M_13    ./.     ./.     M_13    ./.
Mux:    R_08_a, inputs: M_3     M_7
./.     M_3     ./.     ./.     ./.     M_7     ./.     ./.     M_7
Mux:    R_10_a, inputs: M_11
./.     ./.     M_11    ./.     M_11    ./.     M_11    ./.     ./.
Mux:    R_12_a, inputs: I/O     M_10
I/O     ./.     ./.     M_10    ./.     ./.     ./.     ./.     ./.
Mux:    R_13_a, inputs: I/O     M_13
I/O     ./.     ./.     M_13    ./.     M_13    M_13    ./.     ./.
```

```
Mux:   R_17_a, inputs: I/O     M_2
I/O    ./.     M_2     ./.     ./.     M_2     ./.     ./.     M_2
Mux:   R_18_a, inputs: I/O     M_15
I/O    ./.     M_15    ./.     ./.     M_15    ./.     M_15    ./.
Mux:   R_19_a, inputs: M_2     M_10    M_7
./.    M_2     M_10    ./.     ./.     M_10    M_7     M_2     ./.
Mux:   R_20_a, inputs: M_3
./.    ./.     ./.     ./.     M_3     ./.     ./.     M_3     ./.
Mux:   R_21_a, inputs: I/O     M_1
I/O    ./.     M_1     ./.     ./.     M_1     M_1     M_1     ./.
Mux:   R_23_a, inputs: M_1
./.    M_1     ./.     M_1     M_1     ./.     ./.     ./.     M_1
Mux:   R_29_a, inputs: I/O     M_3
I/O    ./.     ./.     M_3     ./.     M_3     M_3     ./.     M_3

################## Datapath netlist ###################
M_1_a       R_13    R_05    R_10
M_1_b       R_02
M_2_a       R_18
M_2_b       R_07
M_3_a       R_29    R_23    R_05
M_3_b       R_12    R_19
M_7_a       R_08    R_07
M_7_b       R_06    R_02    R_13    R_21
M_10_a      R_21
M_10_b      R_17
M_11_a      R_19    R_05    R_06    R_20
M_13_a      R_23    R_06    R_29
M_15_a      R_19    R_02    R_08    R_12
R_02_a      I/O     M_3     M_15    M_11
R_05_a      M_7
R_06_a      M_10
R_07_a      I/O     M_13
R_08_a      M_3     M_7
R_10_a      M_11
R_12_a      I/O     M_10
R_13_a      I/O     M_13
R_17_a      I/O     M_2
R_18_a      I/O     M_15
R_19_a      M_2     M_10    M_7
R_20_a      M_3
R_21_a      I/O     M_1
R_23_a      M_1
R_29_a      I/O     M_3
```

Listing A3.3 can be explained in a similar way than Listing A3.2, the main difference is the number of csteps (columns) in the listing. This is because of the different schedule lengths of the designs, as shown in Figure 5.19 and Figure 5.20 (Chapter 5). Graphical representations of the bindings and datapath netlist of Listing A3.3 were also given in Chapter 5.

Listing A3.3 Output file produced by PABCOM for DCT with $\alpha = 0.4$

```
################## Module binding     ####################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7
Module:M_1
N_23    N_19    ./.     N_64    N_70    N_78    ./.

Module:M_2
./.     ./.     ./.     ./.     ./.     N_77    ./.

Module:M_4
./.     ./.     ./.     ./.     N_68    N_76    ./.

Module:M_5
N_20    N_16    N_26    N_27    ./.     N_73    ./.

Module:M_7
N_22    N_18    ./.     N_69    N_71    N_79    ./.

Module:M_8
./.     N_28    N_24    N_66    N_67    N_74    ./.

Module:M_9
N_21    N_17    N_25    ./.     N_65    N_72    ./.

Module:M_10
./.     N_29    N_31    N_30    ./.     N_75    ./.

Module:M_11
./.     N_50    N_48    N_62    N_38    ./.     ./.

Module:M_12
./.     N_56    N_58    N_52    N_34    ./.     ./.

Module:M_13
./.     N_54    N_46    N_32    N_36    ./.     ./.

Module:M_15
./.     N_60    N_42    N_44    N_40    ./.     ./.


################## Register binding   ####################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7
Register:       R_01
./.     ./.     N_60    N_60    N_60    N_40    ./.

Register:       R_02
D_5     D_5     N_19    N_19    ./.     N_70    ./.

Register:       R_03
D_2     D_2     N_18    N_18    ./.     N_71    N_79

Register:       R_04
D_1     D_1     N_16    N_26    N_26    N_67    N_77

Register:       R_05
D_6     D_6     N_17    N_48    N_52    N_34    ./.

Register:       R_06
D_3     D_3     N_56    N_56    N_62    N_68    N_76

Register:       R_09
./.     ./.     N_50    N_50    N_50    N_38    ./.

Register:       R_10
./.     ./.     N_54    N_54    N_54    N_36    ./.

Register:       R_11
./.     N_23    N_23    N_23    N_64    N_64    N_78

Register:       R_15
```

```
D_7     D_7     ./.     N_46    N_32    N_32    ./.
Register:       R_20
./.     N_22    N_28    N_24    N_66    N_66    N_74
Register:       R_22
./.     ./.     ./.     ./.     N_69    N_69    ./.
Register:       R_23
./.     ./.     N_29    N_31    N_30    ./.     N_75
Register:       R_24
D_4     D_4     ./.     N_58    N_58    ./.     ./.
Register:       R_26
./.     N_20    N_20    N_20    N_27    ./.     N_73
Register:       R_28
./.     N_21    N_21    N_25    N_25    N_65    N_72
Register:       R_30
D_0     D_0     ./.     N_42    N_44    ./.     ./.

#################### Multiplexers ######################
#cs1    cs2     cs3     cs4     cs5     cs6     cs7
Mux:    M_1_a, inputs:  R_30    R_15
R_30    R_30    ./.     R_30    R_30    R_15    ./.
Mux:    M_1_b, inputs:  R_15    R_05
R_15    R_15    ./.     R_15    R_05    R_05    ./.
Mux:    M_2_a, inputs:  R_28
./.     ./.     ./.     ./.     ./.     R_28    ./.
Mux:    M_2_b, inputs: R_11
./.     ./.     ./.     ./.     ./.     R_11    ./.
Mux:    M_4_a, inputs: R_05     R_01
./.     ./.     ./.     ./.     R_05    R_01    ./.
Mux:    M_4_b, inputs: R_09
./.     ./.     ./.     ./.     R_09    R_09    ./.
Mux:    M_5_a, inputs: R_02
R_02    R_02    R_02    R_02    ./.     R_02    ./.
Mux:    M_5_b, inputs: R_03
R_03    R_03    R_03    R_03    ./.     R_03    ./.
Mux:    M_7_a, inputs: R_06
R_06    R_06    ./.     R_06    R_06    R_06    ./.
Mux:    M_7_b,inputs: R_24      R_22
R_24    R_24    ./.     R_24    R_24    R_22    ./.
Mux:    M_8_a, inputs: R_26     R_04    R_15    R_10
./.     R_26    R_04    R_15    R_10    R_10    ./.
Mux:    M_8_b, inputs: R_28     R_05
./.     R_28    R_05    R_05    R_05    R_05    ./.
Mux:    M_9_a, inputs: R_04     R_01
R_04    R_04    R_04    ./.     R_01    R_01    ./.
Mux:    M_9_b, inputs: R_05     R_09
R_05    R_05    R_05    ./.     R_05    R_09    ./.
Mux:    M_10_a, inputs: R_11    R_23    R_04
./.     R_11    R_23    R_04    ./.     R_04    ./.
```

```
Mux:   M_10_b, inputs: R_20
./.    R_20   R_20   R_20   ./.    R_20   ./.

Mux:   M_11_a, inputs: R_28   R_11
./.    R_28   R_28   R_11   R_28   ./.    ./.

Mux:   M_12_a, inputs: R_20   R_23
./.    R_20   R_23   R_23   R_23   ./.    ./.

Mux:   M_13_a, inputs: R_20   R_04
./.    R_20   R_20   R_20   R_04   ./.    ./.

Mux:   M_15_a, inputs: R_11   R_26
./.    R_11   R_26   R_26   R_26   ./.    ./.

Mux:   R_01_a, inputs: M_15
./.    ./.    M_15   ./.    ./.    M_15   ./.

Mux:   R_02_a, inputs: I/O    M_1
I/O    ./.    M_1    ./.    ./.    M_1    ./.

Mux:   R_03_a, inputs: I/O    M_7
I/O    ./.    M_7    ./.    ./.    M_7    M_7

Mux:   R_04_a, inputs: I/O    M_5    M_8    M_2
I/O    ./.    M_5    M_5    ./.    M_8    M_2

Mux:   R_05_a, inputs: I/O    M_9    M_11   M_12
I/O    ./.    M_9    M_11   M_12   M_12   ./.

Mux:   R_06_a, inputs: I/O    M_12   M_11   M_4
I/O    ./.    M_12   ./.    M_11   M_4    M_4

Mux:   R_09_a, inputs: M_11
./.    ./.    M_11   ./.    ./.    M_11   ./.

Mux:   R_10_a, inputs: M_13
./.    ./.    M_13   ./.    ./.    M_13   ./.

Mux:   R_11_a, inputs: M_1
./.    M_1    ./.    ./.    M_1    ./.    M_1

Mux:   R_15_a, inputs: I/O    M_13
I/O    ./.    ./.    M_13   M_13   ./.    ./.

Mux:   R_20_a, inputs: M_7    M_8
./.    M_7    M_8    M_8    M_8    ./.    M_8

Mux:   R_22_a, inputs: M_7
./.    ./.    ./.    ./.    M_7    ./.    ./.

Mux:   R_23_a, inputs: M_10
./.    ./.    M_10   M_10   M_10   ./.    M_10

Mux:   R_24_a, inputs: I/O    M_12
I/O    ./.    ./.    M_12   ./.    ./.    ./.

Mux:   R_26_a, inputs: M_5
./.    M_5    ./.    ./.    M_5    ./.    M_5

Mux:   R_28_a, inputs: M_9
./.    M_9    ./.    M_9    ./.    M_9    M_9

Mux:   R_30_a, inputs: I/O    M_15
I/O    ./.    ./.    M_15   M_15   ./.    ./.

################## Datapath netlist ###################
M_1_a          R_30   R_15
M_1_b          R_15   R_05
M_2_a          R_28
```

M_2_b	R_11			
M_4_a	R_05	R_01		
M_4_b	R_09			
M_5_a	R_02			
M_5_b	R_03			
M_7_a	R_06			
M_7_b	R_24	R_22		
M_8_a	R_26	R_04	R_15	R_10
M_8_b	R_28	R_05		
M_9_a	R_04	R_01		
M_9_b	R_05	R_09		
M_10_a	R_11	R_23	R_04	
M_10_b	R_20			
M_11_a	R_28	R_11		
M_12_a	R_20	R_23		
M_13_a	R_20	R_04		
M_15_a	R_11	R_26		
R_01_a	M_15			
R_02_a	I/O	M_1		
R_03_a	I/O	M_7		
R_04_a	I/O	M_5	M_8	M_2
R_05_a	I/O	M_9	M_11	M_12
R_06_a	I/O	M_12	M_11	M_4
R_09_a	M_11			
R_10_a	M_13			
R_11_a	M_1			
R_15_a	I/O	M_13		
R_20_a	M_7	M_8		
R_22_a	M_7			
R_23_a	M_10			
R_24_a	I/O	M_12		
R_26_a	M_5			
R_28_a	M_9			
R_30_a	I/O	M_15		

Appendix 4

Least Mean Square Error (LMSE) scheduler

This appendix provides a brief description of the LMSE scheduler [58], which is the base for the power-aware time constrained scheduling algorithm presented in Chapter 4. Given a time constraint, the LMSE scheduler assigns operations to control steps such that the cost of hardware resources is reduced. The LMSE scheduler is based on the probability that a certain operation is to be scheduled into a specific control step. This requires the determination of ASAP and ALAP schedules which define the time frame [$ASAP_i$, $ALAP_i$] into which a particular operation (o_i) can be assigned. The probability of scheduling a particular operation (o_i) into any control step (*cstep*) is:

$$prob(o_i, cstep) = \begin{cases} \dfrac{1}{ALAP_i - ASAP_i + 1} & ASAP_i \leq cstep \leq ALAP_i \\ 0 & otherwise \end{cases} \quad \textbf{(A4.1)}$$

All calculated probability values are then used to create distribution graphs (DGs) where values of operations having the same type are added such that:

$$DG(type, cstep) = \sum_{i \in I_{type}} prob(o_i, cstep) \quad \textbf{(A4.2)}$$

where I_{type} is a set containing indices to all operations of the same type. To optimise the utilisation of functional units, it is necessary to assign operations to *csteps* such that the maximum distribution values decrease and the overall DG is more balanced. The LMSE scheduler achieves this balancing task by assessing DGs and the effect of operation assignments upon it using a mean square error (MSE) function approach, as shown in Listing A4.1. The first step of the LMSE scheduler is to sort all DFG operations in a list according to their mobility, i.e. low/high, increasing ASAP time and decreasing number of succeeding nodes. When using the LMSE scheduler in the

algorithm developed in Chapter 4, a new criterion is included to sort the DFG operations, i.e. power consumption. Hence, the DFG operations are sorted in a list according to their mobility, i.e. low/high, power consumption, increasing ASAP time and decreasing number of succeeding nodes. This new sorted list gives higher priority to schedule the most power hungry DFG operations and results in better schedules in terms of resource requirements when compared to the original version.

Listing 4.1 Pseudocode of the LMSE scheduler

```
1  sort all operations according to operation mobility, ASAP time, number of successors
2  while there are unscheduled operations do
3        take the next operation out of the sorted list
4        for all csteps into which the operation could be scheduled do
5              assign the operation temporarily to the cstep
6              update time frames of preceding and succeeding nodes
7              calculate distribution graph for the modified data flow graph
8              evaluate the mean square error function
9        end for
10       schedule operation into the cstep for which the lowest MSE value was found
11       update time frames of preceding and succeeding nodes
12       update distribution graph
13 end while
```

The operation mobility calculated in Listing A4.1 is defined as:

$$mobility(o_i) = ALAP_i - ASAP_i \tag{A4.3}$$

and the group of low mobility operations is

$$O_{low\_mob} = \left\{ o_i \in O \middle| mobility(o_i) < \frac{Min\_mob + Max\_mob}{2} \right. \tag{A4.4}$$

where O is the group of all operations and Min_mob and Max_mob are the lowest and highest mobility values of all operations respectively.

The next step of the algorithm (line 3) consists on taking out of the sorted list the first unscheduled operation. Then, the operation is assigned to all valid *csteps* $j \in [ASAP_i, ALAP_i]$ within its time frame. For each operation assignment to a *cstep j,* modified distribution graphs $DG_j'(type, i)$ should be determined. These modified distribution graphs are used in combination with an average value to evaluate the mean square error (MSE) function:

$$MSE(j,type) = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(DG_j'(type,i) - AVG_{type})^2} \tag{A4.5}$$

where DG'_j $(type, i)$ is the modified distribution graph for an operation assignment into *cstep j* and N is the number of *csteps* in the schedule. AVG_{type} is the average value obtained from the original distribution graph using:

$$AVG_{type} = \frac{1}{M_{type}} \sum_{i=0}^{N-1} DG(type,i) \quad \textbf{(A4.6)}$$

where M_{type} is the theoretical number of *csteps* into which operations of this type can be scheduled.

Since there is one MSE value for each operation type, all MSE values are added to find an overall rate:

$$MSE(j) = \sum_{type} MSE(j,type) \quad \textbf{(A4.7)}$$

Having determined the MSE values for all valid *csteps* (lines 4 to 9), the operation is finally scheduled into the *cstep* that results in the lowest MSE value (line 10). This is followed by adjusting ASAP and ALAP times of preceding and succeeding operations (line 11) and updating the DG values (line 12).

A4.1 Conditional branches

To support scheduling DFGs that contain conditional branches, the LMSE scheduler number all subgraphs consecutively, as shown in Figure A4.1. A subgraph is formed by two or more parallel conditional branches. The branches within a subgraph are assigned a unique two digit number {subgraph, branch}. Having numbered subgraphs and branches, the next step is to generate a set of DGs for each branch of the DFG using equations (A4.1) and (A4.2). To obtain the DG for a particular subgraph the DG values of all branches within the subgraph are combined using the following formula:

$$DG_{subgraph}(j) = \max_{i \in Branches} (DG_i(j)) \quad \textbf{(A4.8)}$$

where *Branches* is a set containing indices to all branches in a subgraph. Usually there is more than one subgraph and to obtain DG values of the equivalent graph the following equation is used

$$DG_{subgraph}(j) = \sum_{i \in Branches} DG_i(j) \quad \textbf{(A4.9)}$$

Nested conditional branches require that equations (A4.8) and (A4.9) are used recursively. This yields finally a DG of an equivalent unconditional DFG which is used by the LMSE scheduler.

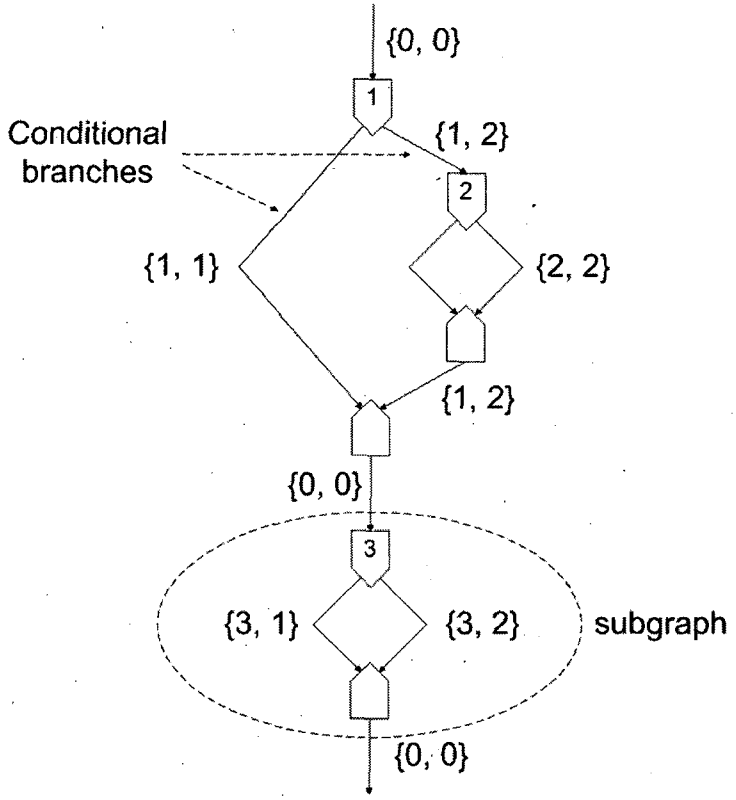


Figure 4.1 Data flow graph with conditional branches [58]

A4.2 Multicycled operations

The LMSE scheduler [58] is also capable of scheduling DFGs with multicycled operations. To achieve this, the probability and mobility equations need to be modified as follows. The single cycle probability value calculation (equation A4.1) is modified to:

$$prob(o_i, cstep) = \frac{\min(cycles_i, cstep - ASAP_i + 1, ALAP_i - cstep + 1)}{ALAP_i - ASAP_i - cycles_i + 2} \quad \textbf{(A4.10)}$$

where the parameter $cycles_i$ refers to the operation multicycled length measured in control steps.

The mobility calculation (equation A4.3) is modified to:

$$mobility(o_i) = ALAP_i - ASAP_i - cycles_i + 1 \quad \textbf{(A4.11)}$$

Equations A4.10 and A4.11 allow scheduling of multicycled functional units without further modifications to the LMSE scheduler.

References

[1] Aarts E. and Korst J., "Simulated Annealing and Boltzman Machines. A stochastic approach to combinatorial optimization and neural computing", Wiley, 1989.

[2] Aarts E. H. L. and van Laarhoven P. J. M., "Statistical cooling: a general approach to combinatorial optimization problems", Philips Journal of Research, 40, pp. 193-226, 1985.

[3] Ahmadi A. and Zwolinski M., "Word-Length Oriented Multiobjective Optimization of Area and Power Consumption in DSP Algorithm Implementation", 25th International Conference on Microelectronics, pp. 614-617, May 2006.

[4] Amellal S. and Kaminska B., "Functional Synthesis of Digital Systems with TASS", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 13, No. 5, pp. 537-552, May 1994.

[5] Blythe S. A. and Walker R. A., "Toward a practical methodology for completely characterizing the optimal design space", Proceedings of the 9th International Symposium on System Synthesis, pp. 8 – 13, November 1996.

[6] Burden R. L. and J. Douglas Faires, "Numerical Analysis", BROOKS/COLE Thomson Learning, 2001.

[7] Chabini N. and Wolf W., "Unification of scheduling, binding and retiming to reduce power consumption under timing and resources constraints", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 10, pp. 1113 – 1126, October 2005.

[8] Chandrakasan A. P., Potkonjak M., Mehra R., Rabaey J. and Brodersen R. W., "Optimising Power Using Transformations", IEEE Transactions on

Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No.1, pp. 12 – 31, January 1995.

[9] Chang E. S., Gajski D. D. and Narayan Sanjiv, "An Optimal Clock Period Selection Method Based on Slack Minimization Criteria", ACM Transactions on Design Automation of Electronics Systems, Vol. 1, No. 3, pp. 352-370, July 1996.

[10] Chang J. M. and Pedram M., "Energy Minimization Using Multiple Supply Voltages", IEEE Transactions on Very Large Scale Integration Systems, Vol. 5, No. 4, pp. 436-443, December 1997.

[11] Pedram M. and Chang J. M., "Module Assignment for Low Power", Proceedings of the conference on European Design Automation, pp. 376 – 381, 1996.

[12] Chang J. M. and Pedram M., "Register Allocation and Binding for Low Power", 32nd Design Automation Conference, DAC, pp. 29 – 35, 1995.

[13] Chaudhuri S. and Walker R.A., "ILP-based scheduling with time and resource constraints in high level synthesis", Proceedings of the Seventh International Conference on VLSI Design, pp. 17-20, January 1994.

[14] Chauduri S., Blythe S. A. and Walker R. A., "A solution methodology for exact design space exploration in a three dimensional design space", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 5, No. 1, pp. 69 – 81, March 1997.

[15] Chiariglione L., http://www.chiariglione.org/mpeg/standards/mpeg-1/mpeg-1.htm, Short MPEG-1 description, International Organisation for standardisation ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio, 1996.

[16] Chtourou S. and Hammami O., "SystemC space exploration of behavioral synthesis options on area, performance and power consumption", The 17th International Conference on Microelectronics, ICM, pp. 67 - 71, December 2005.

[17] Clarke J. A., Gaffar A. A., Constantinides G. A. and Cheung P. Y. K., "Fast word-level power models for synthesis of FPGA-based arithmetic", Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS, pp. 1299 - 1302, May 2006.

[18] Cong J. and Minkovich K., "Optimality Study of Logic Synthesis for LUT-Based FPGAs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 2, pp. 230-239, February 2007.

[19] Cong, J., Yiping F., Guoling H., Wei J. and Zhiru Z., "Platform-Based Behavior-level and System-Level Synthesis", IEEE International System-on-Chip Conference, pp. 199-202, September 2006.

[20] Dal D., Nunez A. and Mansouri N., "Power islands: a high-level technique for counteracting leakage in deep sub-micron", 7th International Symposium on Quality Electronic Design, ISQED, pp. 6, March 2006.

[21] De Micheli G., "Synthesis and Optimization of Digital Circuits", McGraw Hill, 1994

[22] Deming C., Cong J., Yiping F. and Junjuan X., "Optimality study of resource binding with multi-Vdds", 43rd ACM/IEEE Design Automation Conference, pp. 580-585, July 2006.

[23] Deming C., Cong, J., and Junjuan X., "Optimal module and voltage assignment for low-power", Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, Vol. 2, pp. 850-855, January 2005.

[24] Devadas S. and Malik S., "A survey of optimization techniques targeting low power VLSI circuits", Proceedings of the 32nd Design Automation Conference, DAC, pp. 242-247, June 1995.

[25] Elgamel M. A. and Bayoumi M. A., "On low power high level synthesis using genetic algorithms", 9th International Conference on Electronics, Circuits and Systems, Vol. 2, pp. 725-728, September 2002.

[26] Gajski D. D., "Principles of digital design", Prentice-Hall International, 1997.

[27] Gerez S. H., "Algorithms for VLSI Design Automation", Wiley, 1999.

[28] Gi-Joon N., Reda S., Alpert C.J., Villarrubia P.G. and Kahng A.B., "A fast hierarchical quadratic placement algorithm", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 4, pp. 678-691, April 2006.

[29] Gopalakrishnan C. and Katkoori S., "Behavioral synthesis of datapaths with low leakage power", IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 4, pp. 699-702, May 2002.

[30] Gopalakrishnan C. and Katkoori S., "Knapbind: An area efficient binding algorithm for low leakage datapaths", Proceedings of the 21st International Conference on Computer Design, pp. 430 – 435, October 2003.

[31] Gopalakrishnan, C. and Katkoori, S., "Resource allocation and Binding approach for low leakage power", Proceedings of the 16th International Conference on VLSI Design, pp. 297-302, January 2003.

[32] Gu Z., Yang Y., Wang J., Dick R. P. and Shang L., "TAPHS: Thermal-Aware Unified Physical-Level and High-Level Synthesis", Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, pp. 879 - 885, January 2006.

[33] Gupta S. and S. Katkoori, "Force-Directed Scheduling for Dynamic power Optimization", Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp. 68-73, April 2002.

[34] Gupta S., "Tutorial for the SPARK parallelizing High-Level Synthesis Framework. Version 1.1", Center for Embedded Computer Systems. University of California, San Diego and Irvine, 2004.

[35] Gupta S., Gupta R. K., Dutt N. K. and Nicolau A., "SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits", Kluwer Academic Publishers, 2004.

[36] Halsall F., "Multimedia Communications. Applications, Networks, Protocols and Standards", Addison-Wesley, 2001.

[37] Hariyama M., Aoyama T. and Kameyama M., "Genetic approach to minimizing energy consumption of VLSI processors using multiple supply voltages", IEEE Transactions on Computers, Vol. 54, No. 6, pp. 642-650, June 2005.

[38] Hashimoto A. and Stevens J., "Wire routing by optimising channel assignment with large apertures", Proceedings 8th Design Automation Workshop on Design Automation, DAC, pp. 155-169, June 1971.

[39] Hsueh-Chih Y. and Lan-Rong D., "On multiple-voltage high-level synthesis using algorithmic transformations", Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, Vol. 2, pp. 872-876, January 2005.

[40] http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html, The Berkeley MPEG player.

[41] http://en.wikipedia.org/wiki/MPEG, Moving Picture Experts Group.

[42] http://www.chiariglione.org/mpeg/, The MPEG Home Page.

[43] http://esdcad.ecs.soton.ac.uk/bin/view/Knowledgebase/KBArticle_KarthikBaddam_PrimePower, Estimating power consumption of digital logic using PrimePower.

[44] Huaxiang L., Yan L., Zhifang T. and Shoujue W., "SOC Dynamic Power Management Using Artificial Neural Network", Sixth International Conference on Intelligent Systems Design and Applications, Vol. 1, pp. 133-137, October 2006.

[45] Jianfeng H., Jinian B., Zhipeng L. and Yunfeng W., "A fast algorithm for power optimization using multiple voltages in data path synthesis", 6th International Conference on ASIC, ASICON, Vol. 2, pp. 900-904, October 2005.

[46] Jiong L., Lin Z., Yunsi F. and Jha N.K., "Register binding-based RTL power management for control-flow intensive designs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, No. 8, pp. 1175-1183, August 2004.

[47] Johnson M. C. and Roy K., "Data path Scheduling with Multiple Supply Voltages and Level Converters", ACM Transactions on Design Automation of Electronics Systems, TODAES, Vol. 2, No. 3, pp. 227 - 248, July 1997.

[48] Johnson M. C. and Roy K., "Optimal selection of Supply Voltages and Level Conversions during data path scheduling under resource constraints", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 72 – 77, October 1996.

[49] Johnson M. C. and Roy K., "Scheduling and optimal Voltage Selection for Low Power Multi-Voltage DSP datapaths", IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 3, pp. 2152 – 2155, June1997.

[50] Katkoori S. and Vemuri R., "Scheduling for Low Power under Resource and Latency Constraints", IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 2, pp. 53 – 56, May 2000.

[51] Katkoori S., Kumar N., Rader L. and Vemuri R., "A Profile Driven Approach

for Low Power Synthesis", Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, pp. 759-765, September 1995.

[52] Khouri K. S. and Jha N. K., "Leakage Power Analysis and Reduction During Behavioural Synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 10, No. 6, pp. 876 – 885, December 2002.

[53] Khouri K. S., Lakshminarayana G. and Jha N. K., "High-Level Synthesis of Low-Power Control-Flow Intensive Circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 12, pp. 1715-1729, December 1999.

[54] Kim T., Yonezawa N., Liu J. W. S. and Liu C. L., "A scheduling algorithm for conditional resource sharing - a hierarchical reduction approach", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 13, No. 4, pp. 425-438, April 1994.

[55] Kirovski D. and Potkonjak M., "System-level Synthesis of Low-power Hard Real-time Systems", Proceedings of the 34th Design Automation Conference, DAC, pp. 697-702, June 1997.

[56] Kirpatrick S., Gellat C. D. and Vecchi M. P., "Optimization by simulated annealing", Science, 220, pp. 671-680, 1983.

[57] Kollig P. and Al-Hashimi B.M., "Simultaneous scheduling, allocation and binding in high level synthesis", Electronics Letters, Vol. 33, No. 18, pp. 1516 – 1518, August 1997.

[58] Kollig P., Al-Hashimi B.M. and Abbott K.M., "Efficient scheduling of behavioural descriptions in high-level synthesis", IEE Proceedings Computer and Digital Techniques, Vol. 144, No. 2, pp. 75 – 82, March 1997.

[59] Krishna V., Ranganathan, N. and Vijaykrishnan, N., "Energy efficient datapath synthesis using dynamic frequency clocking and multiple voltages", Proceedings of the 12th International Conference On VLSI Design, pp. 440-445, January 1999.

[60] Krishnan V. and Katkoori S., "A genetic algorithm for the design space exploration of datapaths during high-level synthesis", IEEE Transactions on Evolutionary Computation, Vol. 10, No. 3, pp. 213 – 229, June 2006.

[61] Kumar A. and Bayoumi M., "Multiple Voltage-based scheduling methodology for low power in the high level synthesis", Proceedings of the

IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 1, pp. 371-374, June 1999.

[62] Kumar A., Goel S., and Bayoumi M., "A fast heuristic for leakage-aware synthesis", 48th Midwest Symposium on Circuits and Systems, pp. 903-906, August 2005.

[63] Kun-Lin T., Szu-Wei C., Feipei L. and Shanq-Jang R., "A low power scheduling method using dual Vdd and dual Vth", Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 1, pp. 684-687, May 2005.

[64] Laarhoven, P.J.M. van. and Aarts E., "Simulated annealing: Theory and applications", Kluwer academic, 1988.

[65] Lim P. and Kim T., "Thermal aware high level synthesis based on network flow method", Proceedings of the 4th International Conference Hardware/Software codesign and system synthesis, CODES+ISSS, pp. 124-129, October 2006.

[66] Lin Y. R., Hwang C. T. and Whu C. H., "Scheduling Techniques for variable voltage low power designs", ACM Transactions on Design Automation of Electronics Systems, TODAES, Vol. 2, No. 2, pp. 81 - 97, April 1997.

[67] Lin Z. and Jha N.K., "Interconnect-aware low-power high-level synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, No. 3, pp. 336-351, March 2005.

[68] Ling W., Yingtao J. and Selvaraj H., "Synthesis scheme for low power designs with multiple supply voltages by tabu search", Proceedings of the International Symposium on Circuits and Systems, ISCAS, Vol. 5, pp. 261-264, May 2004.

[69] Luo J. and Jha N. K., "Power-Efficient Scheduling for Heterogeneous Distributed Real-Time Embedded Systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 6, pp. 1161 – 1170, June 2007.

[70] Manzak A. and Chakrabarti C., "A Low Power Scheduling scheme With Resources Operating at Multiple Voltages", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 10, No. 1, pp. 6-14, February 2002.

[71] McFarland M. C., Parker A. C. and Camposano R., "The High-Level Synthesis of Digital Systems", Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-318, February 1990.

[72] Mehra R. and Rabaey J., "Exploiting Regularity for Low-Power Design", Proceedings of the International Conference on Computer-Aided Design, ICCAD, pp. 166 – 172, November 1996.

[73] Mehra R. and Rabaey J., "Behavioral level power estimation and exploration", Proceedings of the International Workshop on Low Power Design, pp. 197-202, 1994.

[74] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A. and Teller E., "Equation of state calculations by fast computing machines", Journal of Chemical Physics, Vol. 21, No. 6, pp. 1087-1092, June 1953.

[75] ModelSim SE User's manual, Version 6.1, Mentor Graphics Corporation, May 2005

[76] Mohanty S. P., Ranganathan N. and Chapidi S. K., "Simultaneous peak and average power minimization during datapath scheduling for DSP processors", GLSVLSI, pp. 215 – 220, April 2003.

[77] Mohanty S.P. and Ranganathan N., "Simultaneous peak and average power minimization during datapath scheduling", IEEE Transactions on Circuits and Systems-I: Regular Papers, Vol. 52, No. 6, pp. 1157-1165, June 2005.

[78] Mohanty S.P. and Kougianos E., "Modelling and reduction of gate leakage during behavioral synthesis of nanoCMOS circuits", 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems and Design, pp. 6, January 2006.

[79] Mohanty S.P. and Kougianos E., "Simultaneous Power Fluctuation and Average Power Minimization during Nano-CMOS Behavioral Synthesis", 20th International Conference on VLSI Design, pp. 577-582, January 2007.

[80] Mohanty S.P. and Ranganathan N., "A framework for energy and transient power reduction during behavioral synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 6, pp. 562-572, June 2004.

[81] Mohanty S.P., Kougianos E., Velagapudi R. and Mukherjee V., "Scheduling and binding for low gate leakage nanoCMOS datapath circuit synthesis",

Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS, pp. 5291 - 5294, May 2006.

[82] Mohanty S.P., Ramakrishna V. and Kougianos E., "Physical-aware simulated annealing optimization of gate leakage in nanoscale datapath circuits", Proceedings of Design, Automation and Test in Europe, DATE, Vol. 1, pp. 6, March 2006.

[83] Mohanty S.P., Ranganathan N. and Chappidi S.K., "An ILP-based scheduling scheme for energy efficient high performance datapath synthesis", Proceedings of the International Symposium on Circuits and Systems, ISCAS, Vol. 5, pp. 313-316, May 2003.

[84] Mohanty S.P., Velagapudi R. and Kougianos E., "Dual-K versus dual-T technique for gate leakage reduction: a comparative perspective", 7th International Symposium on Quality Electronic Design, ISQED, pp. 6, March 2006.

[85] Mohanty S. P. and Ranganathan N., "Energy Efficient Scheduling for Data path Synthesis", Proceedings of the 16th International Conference on VLSI Design, pp. 446 - 451, January 2003.

[86] Mohanty S. P., N. Ranganathan and V. Krishna, "Data path Scheduling using Dynamic Frequency Clocking", Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp. 58 – 63, April 2002.

[87] Moore G. E., "No Exponential is Forever: But "Forever" Can Be Delayed!", IEEE International Solid-State Circuits Conference. Digest of Technical Papers., Vol. 1, pp. 20 – 23, February 2003.

[88] Mukherjee R. and Memik S. O., "An Integrated Approach to Thermal Management in High-Level Synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 14., No. 11, pp. 1165 – 1174, November 2006.

[89] Mukherjee R., Memik S. O. and Memik G., "Peak Temperature Control and Leakage Reduction During Binding in High Level Synthesis", Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED, pp. 251 – 256, August 2005.

[90] Mukherjee R., Memik S. O. and Memik G., "Temperature-Aware Resource Allocation and Binding in High-level Synthesis", Proceedings of the 42nd Design Automation Conference, DAC, pp. 196 – 201, June 2005.

[91] Murugavel A.K. and Ranganathan N., "A game-theoretic approach for binding in behavioral synthesis", Proceedings of the 16th International conference on VLSI Design, pp. 452 - 458, January 2003.

[92] Murugavel A.K. and Ranganathan N., "Game theoretic modelling of voltage and frequency scaling during behavioral synthesis", Proceedings of the 17th International Conference on VLSI Design, pp. 670 - 673, 2004.

[93] Murugavel A.K., and Ranganathan N., "A game theoretic approach for power optimization during behavioral synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 6, pp. 1031 - 1043, December 2003.

[94] Musoll E. and Cortadella J., "Scheduling and resource binding for low power", Proceedings of the 8th International Symposium on System Synthesis, pp. 104 – 109, September 1995.

[95] Muthumala W.H., Hariyama M. and Kameyama M., "GA-Based Assignment of Supply and Threshold Voltages and Interconnection Simplification for Low Power VLSI Design", Proceedings of the Asia Pacific Conference on Circuits and Systems, APCCAS, pp. 1264 - 1267, December 2006.

[96] Narayanan S. and Gajski D. D., "System clock estimation based on clock slack minimization", Proceedings of the European Design Automation Conference, EURO-DAC, pp. 66-71, September 1992.

[97] Nestor J. A. and Thomas D. E., "Behavioural Synthesis with Interfaces", Proceedings on the International Conference on Computer Aided Design, ICCAD, pp. 112-115, 1986.

[98] Ochoa-Montiel M.A., Al-Hashimi B.M. and Kollig, P., "Exploiting power-area tradeoffs in behavioural synthesis through clock and operations throughput selection", Proceedings of Asia and South Pacific Design Automation Conference, ASP-DAC, pp. 517 – 522, January 2007.

[99] Ochoa-Montiel M.A., Al-Hashimi B.M. and Kollig, P., "Impact of Multicycled Scheduling on Power-Area Tradeoffs in Behavioural Synthesis",

Proceedings of the International Symposium on Circuits and Systems, ISCAS, Vol. 4, pp. 4163 – 4166, May 2005.

[100] Patel K., Smith B. C and Rowe L. A., "The Berkeley Software MPEG-1 Video Decoder", ACM Transactions on Multimedia Computing, Communications and Applications, Vol. 1, No. 1, pp. 110 - 125, February 2005.

[101] Paulin P. G. and Knight J. P., "Force-Directed Scheduling for the Behavioural Synthesis of ASICs", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 8, No. 6, pp. 661 - 679, June 1989.

[102] Paulin P.G., Knight J.P. and Girczyc E.F., "HAL: A multi-paradigm approach to automatic datapath synthesis", Proceedings of the 23th Design Automation Conference, DAC, pp. 263-270, June 1986.

[103] Pedram M. and Abdollahi A., "Low-power RT-level synthesis techniques: a tutorial", IEE Proceedings on Computer Digital Techniques, Vol. 152, No. 3, pp. 333-343, May 2005.

[104] Pedram M., "Power Minimization in IC Design : Principles and Applications", ACM Transactions on Design Automation of Electronic Systems, Vol. 1, No. 1, pp. 3-56, January 1996.

[105] PrimePower Full-Chip Dynamic Power Analysis for Multimillion-Gate Designs Data Sheet Synopsys, 2004.

[106] Private communication, Peter Kollig, Philips Semiconductors, Southampton, 2004.

[107] Qiang W., Renfa L., Wei W., Wei X., Jinian B., Yunfeng W. and Haili W., "Extend Force-directed Scheduling for System-level Synthesis in Time-constrained System-on-Chip Design", Second International Conference on Embedded Software and Systems, pp. 174 - 180, December 2005.

[108] Rabaey J. M., "Digital Integrated Circuits. A design perspective", 2nd. Edition, Prentice Hall, 2003.

[109] Radhakrishnan B. and Venkatesan M., "Multiple voltage and frequency scheduling for power minimization", Proceedings of the Euromicro Symposium on Digital System Design, pp. 279 – 284, September 2003.

[110] Raghunathan A. and Jha N. K., "SCALP: an iterative-improvement-based low-power data path synthesis system", IEEE Transactions on Computer-

Aided Design of Integrated Circuits and Systems, Vol. 16, No. 11, pp. 1260 – 1277, November 1997.

[111] Raghunathan A. and Jha N. K., "Behavioural Synthesis for Low Power", Proceedings of the IEEE International Conference on Computer Design, pp. 318 - 322, October 1994.

[112] Raghunathan A., Jha N. K. and Dey S., "High-level power analysis and optimization", Kluwer Academic Publishers, 1998.

[113] Raje S. and Sarrafzadeh M., "Scheduling with multiple voltages", Integration: The VLSI Journal, Vol. 23, No. 1, pp. 37 –59, 1997.

[114] Raje S. and Sarrafzadeh M., "Variable Voltage Scheduling", Proceedings of the International Symposium on Low Power Electronics and Design, pp. 9 – 14, 1995.

[115] Ramanujam J., Deshpande S., Hong J. and Kandemir M., "A Heuristic for clock selection in high level synthesis", Proceedings of the 15th International Conference on VLSI Design, pp. 414 – 419, January 2002.

[116] Ranganathan N., Namballa R., Hanchate N., "CHESS: a comprehensive tool for CDFG extraction and synthesis of low power designs from VHDL", IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, pp. 6, March 2006.

[117] Reeves C. R., "Modern Heuristics Techniques for Combinatorial Problems", McGraw-Hill International (UK), 1995.

[118] Rettberg A. and Rammig F.J., "A new design partitioning approach for low power high-level synthesis", Third IEEE International Workshop on Electronic Design, Test and Applications, DELTA, pp. 6, January 2006.

[119] Rim M., Mujumdar A., Jain R. and De Leone R., "Optimal and heuristic algorithms for solving the binding problem", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 2, No. 2, pp. 211 – 225, June 1994.

[120] Ruiz-Sautua R., Molina M. C. and Mendias J. M., "Exploiting Bit-Level Delay Calculations to Soften Read-After-Write Dependences in Behavioural Synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 9, pp. 1589 – 1601, September 2007.

[121] Safari S., Esmaeilzadeh H. and Jahangir, A.H., "A novel improvement technique for high-level test synthesis", Proceedings of the International Symposium on Circuit and Systems, ISCAS, Vol. 5, pp. 609 - 612, May 2003.

[122] San Martin R. and Knight J. P., "Optimising Power in ASIC Behavioural Synthesis", IEEE Design & Test of Computers, Vol. 13, No. 2, pp. 58 - 70, Summer 1996.

[123] San Martin R. and Knight J. P., "Power-Profiler: Optimising ASICs Power Consumption at the Behavioural Level", 32nd Design Automation Conference, DAC, pp. 42 – 47, 1995.

[124] Sharp R., "Higher-Level Hardware Synthesis", Series: Lecture Notes in Computer Science, Springer, Vol. 2963, 2004.

[125] Shiue W. T. and Chakrabarti C., "ILP-Based scheme for low power scheduling and resource binding", IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 3, pp. 279 – 282, May 2000.

[126] Shiue W. T. and Chakrabarti C., "Low Power Scheduling with resources operating a multiple voltages", Proceedings of the International Symposium on Circuits and Systems, ISCAS, Vol. 2, pp. 437 – 440, June 1998.

[127] Shiue W. T. and Chakrabarti C., "Low Power Scheduling with Resources Operating at Multiple Voltages", IEEE Transactions on Circuits and Systems II. Analog and Digital Signal Processing, Vol. 47, No. 6, pp. 536 – 543, June 2000.

[128] Synplify ASIC and Amplify ASIC, Reference manual, Synplicity Inc, October 2004.

[129] Szedo G., "Color-Space Converter: RGB to YCrCb", Xilinx, XAPP930 (v1.0), Application Note: Virtex-4, Virtex-II, Virtex-II Pro, Spartan3, May 2006.

[130] Thomas D. E. and Moorby P. R., The Verilog Hardware Description Language, 5th edition, Springer, 2002.

[131] Tseng C.-J., Wei R.-S., Rothweiler S. G., Tong M. M. and Bose A. K., "Bridge: A versatile behavioral synthesis system", Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 415 - 420, May 1988.

[132] UNICAD 2.4 Release Notes, Library CORE9GPLL version 4.1, September 2003.

[133] VBrick MPEG-1 Encoder/Decoder Datasheet, VBrick Systems Inc., 2004.

[134] Veendrick H, "Deep-Submicron CMOS ICs: from Basics to ASICs", Kluwer Academic Publishers, 1999.

[135] Virage Logic Corporation, "Technical overview. Power Reduction Techniques for Ultra-Low-Power Solutions", white paper, June 2004.

[136] Wakabayashi K. and Yoshimura T., "A resource sharing and control synthesis method for conditional branches", Proceedings of the International Conference on Computer Aided Design, ICCAD, pp. 62 - 65, November 1989.

[137] Walker R.A. and Chaudhuri S.; "Introduction to the scheduling problem", IEEE Design & Test of Computers, Vol. 12, No. 2, pp. 60 - 69, Summer 1995.

[138] Wang L., Jiang Y. and Selvaraj H., "Synthesis scheme for low power designs with multiple supply voltages by heuristic algorithms", Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC, Vol. 2, pp. 826 – 830, 2004.

[139] Weng J.-P. and Parker A. C., "CSG: Control path synthesis in the ADAM system", Proceedings of the 6th International Workshop on High Level Synthesis, pp. 52-64, 1992.

[140] Weste N. H. E. and Harris D., "CMOS VLSI Design. A circuits and systems perspective", 3rd. edition, Addison Wesley, 2005.

[141] Williams A. C., Brown A. D. and Zwolinski M., "Simultaneous optimisation of dynamic power, area and delay in behavioural synthesis", IEE Proceedings Computers and Digital Techniques, Vol. 147, No. 6, pp. 383 – 390, November 2000.

[142] Wu C-H. and Irwin J. D., "Emerging Multimedia Computer Communication Technologies", Prentice Hall, 1998.

[143] Xiaoyong T., Hai Z. and Banerjee P., "Leakage power optimization with dual-Vth library in high-level synthesis", Proceedings of the 42nd Design Automation Conference, DAC, pp. 202 - 207, June 2005.

[144] Xiaoyong T., Tianyi J., Jones A. and Banerjee P., "Behavioral synthesis of data-dominated circuits for minimal energy implementation", 18th International Conference on VLSI Design, pp. 267 – 273, 2005.

[145] Yamada A., Yamazaki T., Ishiura N., Shirakawa I. and Kambe T., "Datapath scheduling for conditional resource sharing", Asia Pacific Conference on Circuits and systems, APCCAS, pp. 169 – 174, December 1994.

[146] Yan L., Jiong L. and Jha N.K., "Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, No. 7, pp. 1030 - 1041, July 2005.

[147] Yoonseo C. and Taewhan K., "An efficient low-power binding algorithm in high-level synthesis", IEEE International Symposium on Circuits and Systems, ISCAS, Vol. 4, pp. 321 - 324, May 2002.

[148] Zhen Z., Jinian B., Zhipeng L., Yunfeng W. and Kang Z., "High Level Synthesis with Multiple supply Voltages for Energy and Combined Peak Power Minimization", IEEE Asia Pacific Conference on Circuits and Systems, APCCAS, pp. 864 – 867, December 2006.

[149] Zhipeng L., Jinian B., Jianfeng H. and Yunfeng W., "Fast and efficiently binding of functional units for low power design", 6th International Conference On ASIC, ASICON, Vol. 1, pp. 10 - 13, October 2005.

[150] Zhipeng L., Jinian B., Qiang Z., Liu Y. and Yunfeng W., "Interconnect Power Optimization Based on the Integration of High-level Synthesis and Floorplanning", Proceedings of the International Conference on Communications, Circuits and Systems, Vol. 4, pp. 2286 – 2290, June 2006.

[151] Zwolinski M., "Digital System Design with VHDL", 2nd. Edition, Prentice Hall, 2004.