UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

Architecture Level Power-Performance Trade-offs in Data-dominated Designs

by

Haider Farhan Ali

Thesis for the degree of Doctor of Philosophy

April 2008

UNIVERSITY OF SOUTHAMPTION

ABSTRACT

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Architecture Level Power-Performance Trade-offs in Data-dominated Designs

by Haider Farhan Ali

As the demand for feature-rich portable devices continues to increase, new techniques are needed to minimise power consumption. This thesis is concerned with the development and validation of new systematic architectural methods of determining pipeline stage insertion in data-dominated designs with the aim of reducing dynamic power consumption. The methods place special emphasis on the number of latches used in pipeline stages and voltage scaling. The first part of the thesis addresses power minimisation through systematic analysis of the number of latches in pipeline stages. A new pipeline stage insertion (PSI) method operating at the architectural level is developed which takes into account system clock period and FEs outputs and delays. A PSI algorithm based on analytical heuristic equations is formulated to ensure the successful application of this method to any given data-dominated design. The input to the algorithm is designer clock period and naively inserted pipeline stages. The output from the algorithm is a pipelined design fulfilling the timing constraint with the least dynamic power consumption. To support efficient power-performance trade-offs exploration, the algorithm was fully automated.

The second part of the thesis focuses on the validation of the PSI method using two real-life case studies: triple data path floating-point adder and MPEG-1 motion compensation module. These designs are common in many portable devices and have numerous implementation challenges including large number of FEs and significant power consumption. Extensive experimental results show that for the motion compensation module, the PSI is able to reduce dynamic power consumption by up to 30% compared with other reported approaches.

The final part of the thesis concentrates on voltage scaling (VS) and its impact on pipeline stages. The timing slack available in each stage is investigated, with the aim of further reducing power consumption by lowering the supply voltage. The PSI method is modified to support voltage scaling, and as a result, a new pipeline stage insertion with voltage scaling (PSI-VS) method is proposed. Experimental results show that the PSI-VS can lead to significant power saving compared with PSI without VS. For the MPEG-1 motion compensation case study, a power saving of 68% is observed. All the developed methods have linear time complexity as the number of pipeline stages increases, facilitating their application to large designs without incurring run time penalty. The results for the case studies were based on a synthesisable RTL implementation using 90nm technology together with accurate power analysis using commercial tools.

Contents

List of Figures

List of Tables

Acknowledgments

My supervisor, Professor Bashir Al-Hashimi, has provided me with guidance and encouragement needed to complete the work presented in this thesis. I am very grateful to him for teaching me how to conduct research, how to clearly and effectively present my work, and how to organise and write technical papers.

I would like to thank the members of the Electronics System Design Group (ESD) at the University of Southampton. In particular, I am grateful to my colleague Marco A Ochoa-Montiel for his friendship and providing feedback on several occasions.

This work was funded by Nokia UK Ltd and I am thankful for this financial support. I would like to acknowledge the valuable support from my colleagues Alice Wilson and Graeme Dixon and for providing excellent discussion and comments; I am grateful to them.

Last but certainly not least, I have to and I wish to acknowledge the people who are the closest to me. My dad Farhan, my mum Salima, my wife Hawra'a and my two sons Sajad and Laith have all been there for me when I needed them. In particular, I am indebted to Hawra'a for her unlimited and unconditional support and love. At times, it was hard for Hawra'a to look after the boys and maintain a successful career while I studied, but she always supported and encourage me to do this research. I dedicate this thesis to you.

Chapter 1

Introduction

Low power design has been an active area of research over the last decade and a half and numerous techniques have been reported at different levels of the design flow including device level (silicon-on-insulator (SOI)) [136], gate level [137], architecture level [131] and system level [138]. The main driver for this continuous research has been the ability to produce digital circuits and system designs with low power consumption for the battery-powered electronic products. The remainder of this chapter is organised as follows. Section 1.1 provides motivation for this work, section 1.2 defines the aims for this thesis, section 1.3 outlines preliminaries and section 1.4 details thesis contribution and organisation.

1.1 Motivation

Whilst there has been significant progress in low power design techniques, there is still a need to develop more efficient techniques at different levels of the design flow. In the past, the device density and operating frequency were low enough that power was not a constraining factor in the design of those devices. However, as smaller technologies are becoming more available and the scale of integration grows, power dissipation is now a major issue. Demands for low power designs are emerging from different sources including:

a) Increased competition in portable consumer devices has lead to an increase in smaller, lighter and more durable electronic products. Battery life is a product differentiator in many portable electronic devices.

b) Power dissipation has direct impact on packaging and cooling cost of devices.

c) High power dissipation tends to exacerbate silicon failures due to running at high temperatures.

Since this thesis is concerned with power consumption in the context of pipelining, research has shown that the higher the power consumption issue is tackled in the design flow, the more reward is gained in terms of trying to minimise it [1, 2, 131]. A typical VLSI design flow constitutes a number of levels and each level comprises a transformation of the design from the previous level to the next level. For example, once the specification has been agreed, the first highest level of the design flow after that is the system level. At this level, the decisions taken include how many processors are needed, what the appropriate bus system is, what is the partition between hardware and software is, and so on. The next level is the algorithm level where the aim is to reduce the complexity of the algorithms used and minimise unnecessary operations. The lack of implementation-specific information limits the accuracy of estimation tools at the system and algorithmic level and therefore makes pipelining analysis at this level impractical. The next level of the design flow is the architecture level. More accurate results can be obtained at the architecture level since the data paths, memory, control and interconnect structures are fully defined. Optimisation at this level of the design flow means finding the best architectural composition for a given design. Lower levels of the design flow such as register-transfer level (RTL), gate level and device level do provide an opportunity to reduce power consumption; however, these levels are near the end of the design flow with less freedom and more cost involved in implementing changes to the design [1]. It is for those reasons that this thesis addresses pipeline stage insertion at the architecture level to take advantage of more flexibility and the ability to explore different options without incurring long iteration time. In this chapter, pipelining is put in the context of architecture level, and how different pipeline stages can impact power-performance during design space exploration.

1.2 Aims

The aim of this research is to develop efficient low power design methods at the architecture level. The architecture level of the design flow represents the appropriate level at which to optimise the design for low power due to the existence of full implementation of the design components. This thesis also aims to further reduce the power consumption by utilising the slack available in the design to lower the supply voltage. The methods presented in this thesis will determine the appropriate pipeline stage insertion that yields the least power consumption while meeting the target performance in data-dominated designs.

1.3 Preliminaries

This section introduces definitions and outlines concepts used in this thesis with special emphasis on pipelining. Section 1.3.1 provides the background to pipelining; section 1.3.2 outlines power consumption in CMOS and section 1.3.3 gives an overview of floating-point data representation. This thesis is interested in floating-point representation as a background to the literature survey in chapter 2, section 2.2, and the case study presented in chapter 4 will validate the PSI method using triple data path floating-point adder.

1.3.1 Pipelining Background

Pipelining is an implementation technique that allows the processing of multiple input data to overlap [4, 39]. Data enters the pipe from one end, progresses through the stages, and exits at the other end as shown in Figure 1.1. When the intermediate result from stage 1 enters stage 2, new data can be processed at stage 1; thus the stages are processing data in parallel.

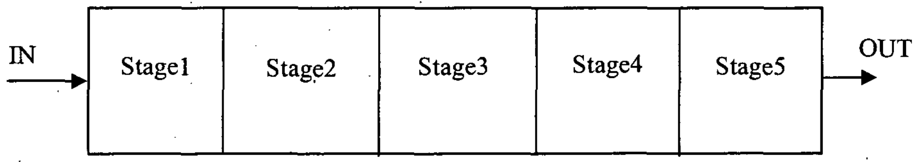


Figure 1.1 Basic Pipeline Structure for Five Stages

If a design is divided into *N*-stages of equal delays, then latency is defined as the delay of the non-pipelined design (*t*) plus the delay of the latching element (ϵ), [4]:

$$Latency = t + N\varepsilon \tag{1.1}$$

Throughput is determined by how often results exit the pipeline. If data are applied to stage1 at time *T1* shown in Figure 1.2, after stage delay of T_{stage} plus latch delay of ϵ data is safely stored in latches *L1*. At time *T2* a new set of data can be applied to the pipeline. The difference between *T2* and *T1* is called system clock period or machine cycle. The inverse of system clock period is throughput:

$$Throughput = \frac{1}{clock\ period} = \frac{1}{T_{stage} + \varepsilon} \tag{1.2}$$

In a pipelined design, latency assumes a secondary role, and generally, the main focus is on increasing throughput. The goal of designers is to be able to balance stages delay such that smaller system clock period and hence higher throughput can be achieved.

Pipelining adds latching elements (in this thesis the words latches and registers are used interchangeably to mean a storage element. In an actual implementation registers are frequently used since they require a single-phase clock, which simplifies the clocking task) to create stages, which result in parallel processing of data. In deep submicron designs, these latches are power hungry and represent a significant proportion of dynamic and leakage power consumption [86-88]. In order to minimise power consumption, the stages need to be carefully planned such that the number of latches inserted is kept to a minimum. The insertion of latches is by definition creating more stages; therefore inserting latches in the context of this thesis is referred to as stage insertion.

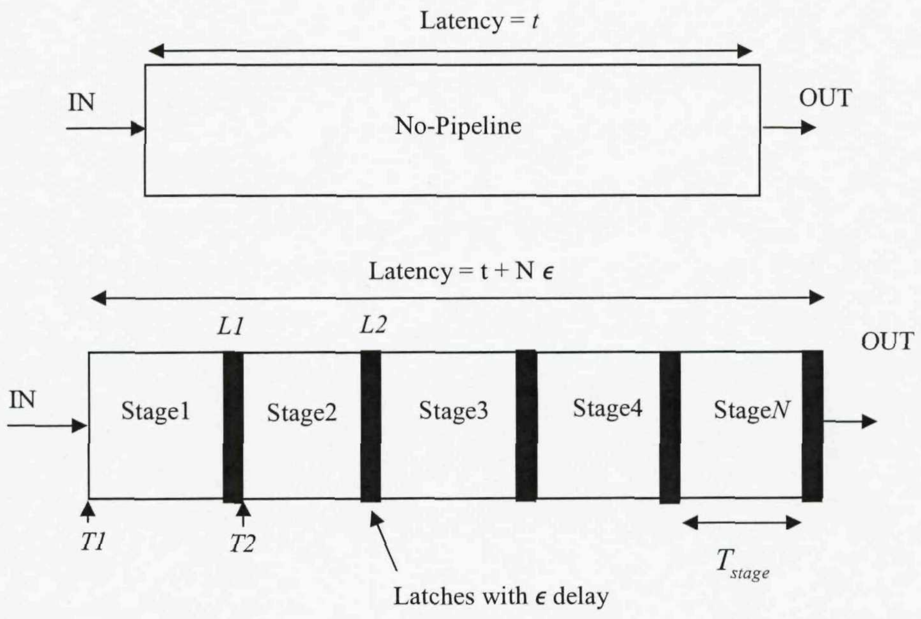


Figure 1.2 Latency and Throughput of a Pipeline

As already mentioned in section 1.1, the architecture level is a compromise between the system level (lacking implementation specific information), and the circuit level (more complexity and high cost in implementing design changes). The architecture level of a design, comprises building blocks called functional elements (FE) as shown in Figure 1.3(a). These elements have well defined functionality and interfaces (in this thesis, interfaces are also referred to as FE boundaries). Accurate analysis of power consumption due to an architectural decision such as pipelining, would require these FEs to be implemented in hardware description language and synthesised to the target technology library. Chapter 4 describes the implementation and synthesis of a floating-point adder architecture.

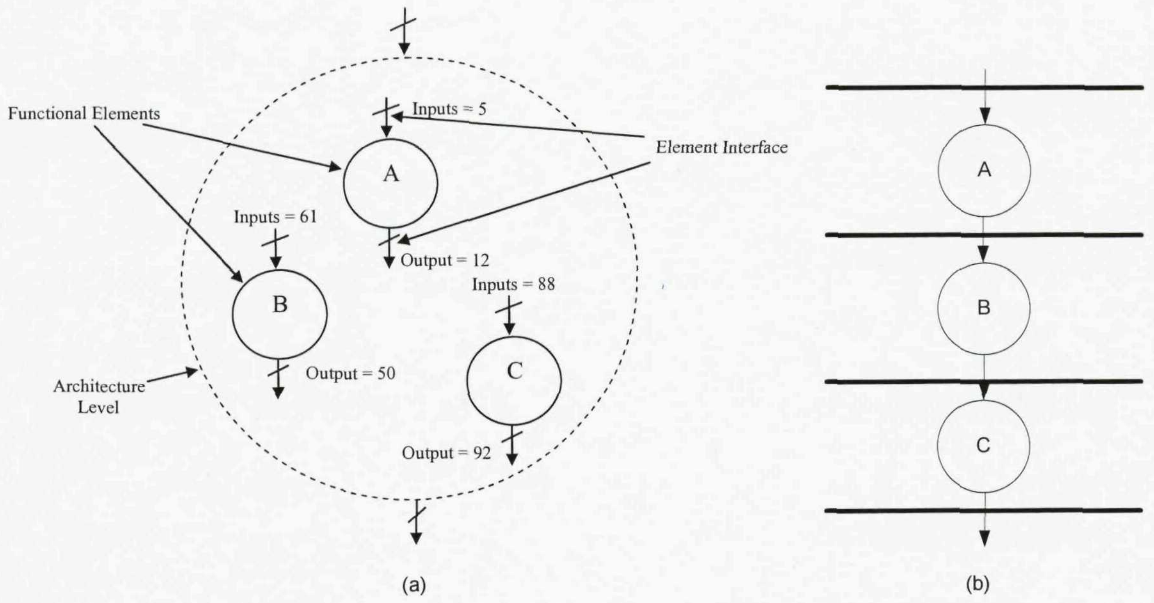


Figure 1.3 (a) Architecture Level and FEs; (b) Naive Stages

The simplest stage insertion, or *naive insertion*, as it is referred to in this thesis, is implemented by latching every output interface without considering the number of outputs as shown in Figure 1.3(b). Such pipelining costs the least in terms of designer effort because it is simple to implement. As each element has a different number of outputs, the number of latches inserted will be different from one interface to the other. In general, removing stages without compromising the system clock period reduces the number of latches and thus reduces power consumption. If a choice exists between elements' outputs to be latched in order to create a stage, the element with the smallest number of outputs should be selected. The focus of this thesis is to explore these choices in order to minimise power while maintaining performance as considered in chapter 3.

For a pipelined design, system performance is directly linked to throughput; hence, higher throughput means higher performance. To achieve higher performance, the system clock period has to decrease, and in terms of frequency, this implies higher system operating frequency. As frequency has direct impact on power (see Equation 1.4), this thesis offers a way of exploring multiple clock frequencies and their impact on power consumption, such that designers can choose the result that fits their requirements. This is considered in detail in chapters 4 and 5.

The type of design architecture addressed by the thesis is data dominated architecture. Arithmetic units represent one of the challenging data dominated architectures in terms of power and performance [6, 20]. These types of designs are widely used in Digital Signal Processing (DSP) systems [21, 59, 60, 62, 65]. DSPs are generally classified into two types: fixed-point and floating-point [4]. One of the many decisions explored at the architecture level is the use of binary Floating-Point (FP) versus binary Fixed-Point representation of data [3, 25, 26, 27, 29]. This decision is of paramount importance for many DSP applications where arithmetic units are intensively used and cost-sensitivity is at the heart of such applications. When the intended application is well understood and the range of data is well defined, integer DSPs (based on fixed-point) with data scaling are used for such applications. However when the data values are highly variable or unpredictable, or scaling impact presents an unacceptable overhead, floating-point arithmetic units become mandatory. Therefore, addressing the power and performance challenges in the floating-point units will help reduce the overall system power and help with meeting the target system performance.

The simplicity, speed and relatively low cost of arithmetic operations in Fixed-Point format means that a number of algorithms use it in their implementation. However, computation often involves a wider range of numerical values, which are not easily represented in Fixed-Point format. Increasing the bit width of Fixed-Point representation or scaling the Fixed-Point numbers will mean increased complexity of arithmetic operations and the ability to represent both large and small numbers will require even larger bit width. Hence, one of the downsides of Fixed-Point representation is lack of range. Binary FP representations like the IEEE 754 [5] have been designed with greater range in mind. The IEEE-754 specifies that any single-precision FP number can be represented using 32-bits. This gives a dynamic range of $1.99999988\times 2^{-126}$ to 1.99999988×2^{127}, which is substantially wider than the Fixed-Point range of 1 to 2^{31}. FP range is suitable for most operations that demand wider ranges. Due to operation complexity, FP units have become synonymous with significant power consumption. More processing to the input operands compared with Fixed-Point requires more complex functional units and hence more power.

As more DSPs are used in applications where a more dynamic range is required to represent data, this necessitates the usage of Floating-Point arithmetic units. Since these applications tend to require higher throughput, pipelining of Floating-Point arithmetic unit is essential to improving the overall application throughput.

This thesis aims to use a floating-point arithmetic unit as a benchmark circuit to analyse the impact of pipeline stage insertion on power and performance. This will be covered in more detail in chapter 4.

1.3.2 Power Consumption in CMOS

There are three contributors to power dissipation in CMOS circuits: Dynamic Switching Power, Short Circuit Power and Leakage Power. The total average power dissipation in CMOS is the sum of these three components:

$$P_{total} = P_{dynamic} + P_{short-circuit} + P_{leakage} \tag{1.3}$$

The dynamic component of power dissipation arises from the simple charging and discharging of a device capacitance from a constant voltage supply as depicted in Figure 1.4 for a CMOS inverter.

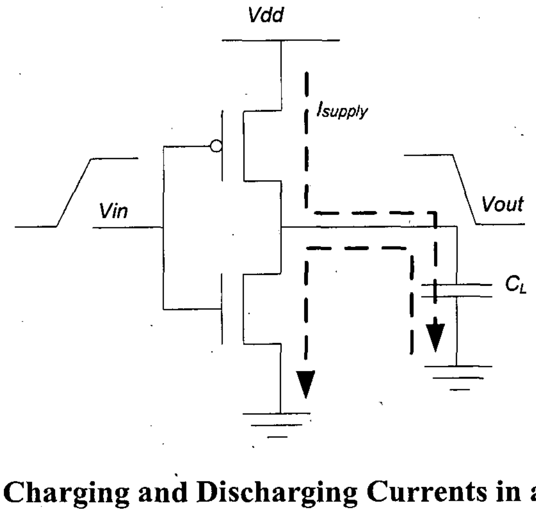


Figure 1.4 Charging and Discharging Currents in an Inverter

This component was the most dominant source of power dissipation in early CMOS devices; however, in sub-100nm designs, leakage is becoming a significant component of overall power [40-43]. The following equation shows the variables that affect dynamic power dissipation [148]:

$$P_{dynamic} = \frac{1}{2} \alpha \, C_L V_{dd}^{\;2} f \qquad (1.4)$$

Where:

α Switching activity, activity is defined as the number of 0-1 transition per clock cycle.

C_L Load capacitance

f Device frequency

V_{dd} Supply voltage

The short circuit power is due to simultaneous conduction of pMOS and nMOS transistors creating a direct path from *Vdd* to ground. This causes current to flow leading to this power dissipation. Short circuit current depends on the following factors [2]:

1. The duration and slope of the input signal. This current will increase with increased rising and falling times of input transition. Figure 1.5 depicts the short circuit current of a CMOS inverter during input transition. The current is zero when the input voltage is below *V1* or above *V2*. The mid point between V1 and V2 is when both transistors are conducting and therefore where maximum short circuit current occurs.
2. The I-V curves of the pMOS and nMOS transistors, which depend on their sizes, process technology, temperature, etc. The current will decrease and eventually tend to zero when *Vdd* is scaled down but the threshold voltages are not scaled proportionally.
3. This current is inversely proportional to the output capacitance. Current will decrease for larger output load capacitance.

In static conditions (where all inputs and internal nodes have constant voltages) a small leakage current flows in CMOS from power supply to ground. This current is responsible for the leakage power and it is sometimes referred to as static power.

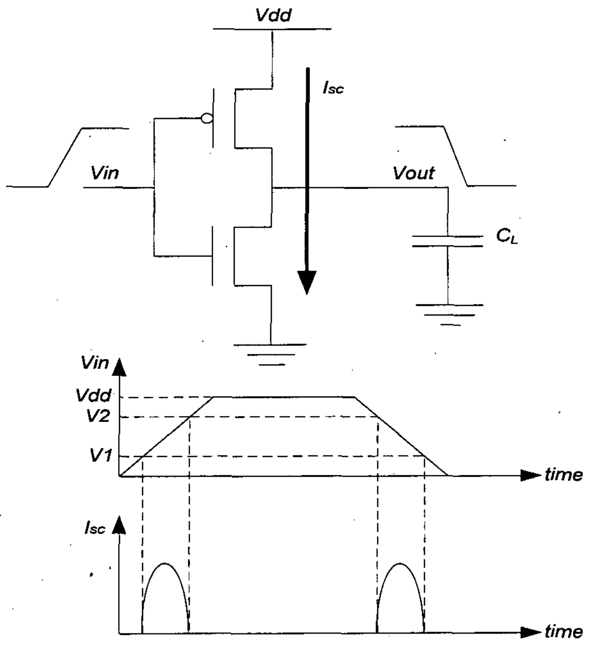


Figure 1.5 Inverter Short Circuit Current during Input Transition

Leakage current has become a major portion of total power consumption, and in many scaled technologies leakage contributes 30-50% of the overall power consumption under nominal operating conditions [146]. There are two major sources of leakage current, *Subthreshold Current* and *gate leakage*. PN-junction current is largely independent of operating voltage but has dependency on fabrication process, junction area and temperature. Equation 1.5 shows the subthreshold current [148].

$$I_{sub} = I_0 e^{\frac{(V_{gs}-V_t)}{\alpha V_{th}}} \tag{1.5}$$

Where:

I_0 Current when $V_{gs} = V_t$

V_{gs} Gate voltage

V_t Threshold voltage

V_{th} Thermal voltage

α Constant dependent on device fabrication process ranging from 1.0 to 2.5

As technology scaling continues, subthreshold current is becoming a limiting factor in low voltage and low power designs. Reducing the operating voltage to reduce power in future technologies has to be accompanied by a corresponding reduction in threshold voltage to compensate for loss in switching speed. Traditionally, the threshold voltage will have to scale down to a level capable of sustaining a 30% gate delay reduction when scaling the supply voltage [146]. However reducing threshold voltage will increase subthreshold current and therefore increase leakage power. To elaborate on this, in a MOS transistor the drive current of an ON-state MOS transistor which is used to charge or discharge the output capacitor is proportional to $(Vdd - V_t)^a$. This indicates that if Vdd is reduced there needs to be a corresponding reduction in V_t to maintain the drive current. When the transistor approaches the off-state, the V_{gs} goes below V_t. This means the drain current changes exponentially with V_{gs} for V_{gs} below V_t. Figure 1.6 illustrates how reduction in V_t results in larger subthreshold leakage current. In this figure, reducing V_t by 85mV resulted in subthreshold current (*Ioff*) increasing by 10X [146].

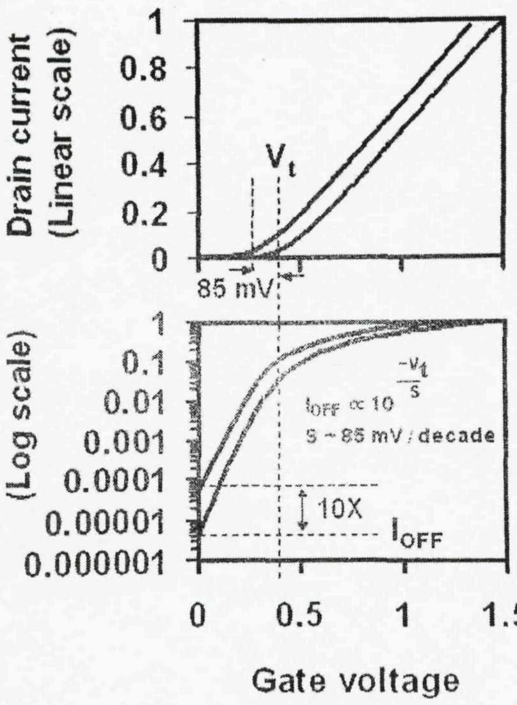


Figure 1.6 Relationship between Threshold Voltage (V_t) and Subthreshold Leakage Current (*Ioff*) for NMOS Transistor [146]

Scaling the channel length for technology shrink requires maintaining a good transistor aspect ratio and this in turn requires a comparable scaling of gate oxide thickness, junction depth and depletion depth [40]. These are important parameters for ideal MOS transistor behaviour. Unfortunately, with technology scaling, maintaining good transistor aspect ratio has been a challenge. This is because vertical dimensions have been harder to reduce than horizontal dimensions. With the silicon oxide gate dielectric thickness approaching scaling limits there is now a rapid increase in gate direct tunnelling leakage current [146]. Figure 1.5 shows the area component of gate leakage current in A/cm^2 versus gate voltage for various oxide thicknesses. From Figure 1.7, it can be seen that the gate oxide can be scaled to 2.0nm (represented by the 20 Å curve), before exceeding the 1 A/cm^2 limit. Below 2.0nm, however, the oxide tunnelling current quickly becomes problematic. Note, 1 A/cm^2 tunnelling current density corresponds to a leakage level of 1nA per 1um device width for 100nm channel length MOSFETs. Unless a new gate dielectric material is discovered, this sets a limit for bulk CMOS scaling.

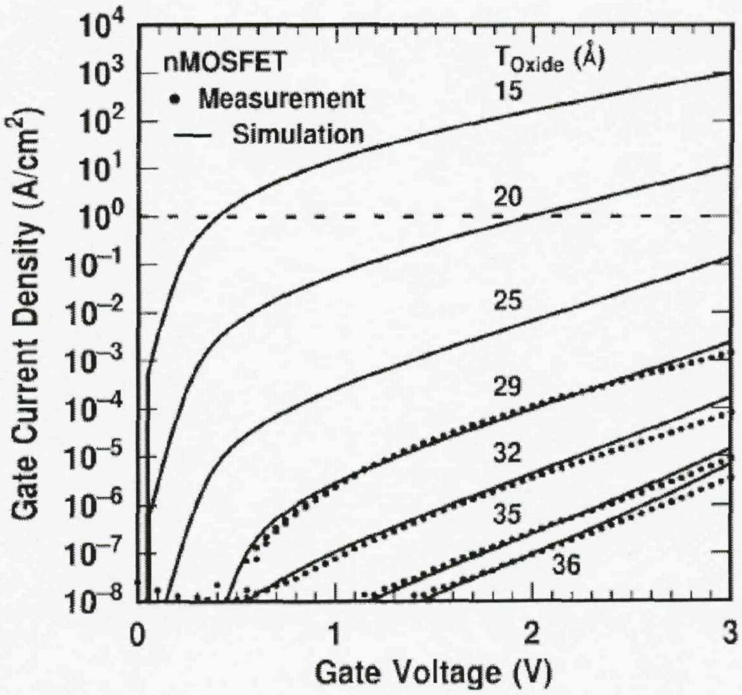


Figure 1.7 Gate Leakage versus Gate Voltage for Various Oxide Thicknesses [147]

Therefore, technology scaling beyond 100nm channel length will have to tackle the leakage power consumption in order to have a reasonable standby power. Any optimisation method, including pipelining, should aim to reduce the number of devices on the chip in order to reduce leakage power. Chapter 3 will introduce a pipelining method that aims to reduce the number of registers used in implementing the pipeline stages

1.3.3 Floating-Point Overview

This section presents an overview of floating-point (FP) data representation as the background to the literature survey in chapter 2, section 2.2. There are several ways to represent real numbers on computers. Fixed point places a radix point somewhere in the middle of the digits, and this is equivalent to using integers that represent portions of some units. Another approach is to use rationals and represent every number as the ratio of two integers. Floating-point representation, the most common solution, basically represents real numbers in scientific notation. Scientific notation represents numbers as a base number and an exponent. Floating-point solves a number of representation problems. Fixed point has a fixed window of representation, which limits it from representing very large or very small numbers. In addition, fixed point is prone to loss of precision when two large numbers are divided. Floating-point on the other hand employs a sort of sliding window of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease. In this thesis the IEEE standard 754-1985 [5] is assumed when discussing FP representation. Figure 1.8 shows the IEEE 754 single precision format.

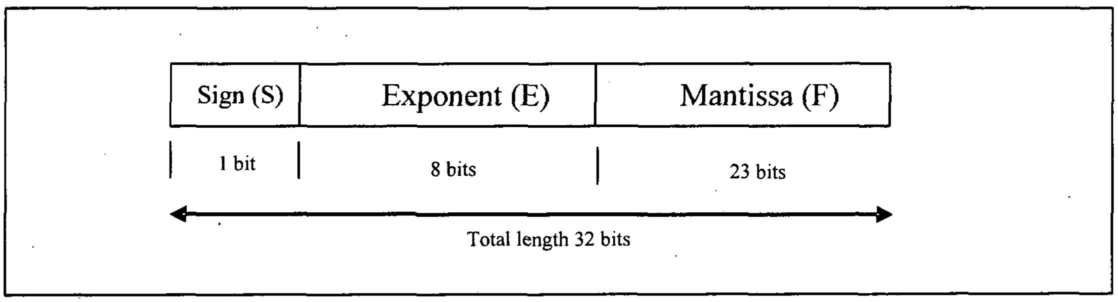


Figure 1.8 IEEE-754 Floating-Point Standard

The mantissa also known as fraction or significand defines the precision of a number and this in turn is a measure of the exactness of that number. The exponent defines the range of a number, which is a measure of how big or how small it can be. For the IEEE-754 standard, the radix or base is assumed to be binary two. The mantissa of an IEEE-754 format is normalised, that is to say the fractional mantissa F lie in the range$1 \leq F < 2$. This range corresponds to a mantissa with an integer part equal to one. The IEEE floating number Y represents a number formally defined as:

$$Y = (-1)^S x 2^{E-B} x 1.F \tag{1.6}$$

Where,

S is the sign bit: 0 positive mantissa, 1 negative mantissa

E is the exponent biased by B

F is the fractional mantissa (note that the leading 1 is implicit and not written in the format shown in Figure 2.5)

The exponent takes care of both negative and positive numbers. This is achieved through biasing, B. The IEEE single precision format bias number is 127 and the maximum exponent is 255.

1.4 Thesis Contribution and Organisation

The original contribution in this thesis can be summarised as follows:

- Development of a new systematic architecture-level method to determining pipeline stage insertion (PSI) in data-dominated designs with the aim of reducing dynamic power consumption. A PSI algorithm is formulated to ensure the successful application of this method for any given data-dominated design. Automation of the algorithm along with power analysis tasks enables efficient power-performance trade-offs exploration.
- The PSI method is validated on two real-life and challenging data dominated designs: Triple Data Path Floating Point Adder (TDPFPADD) [48] and MPEG-1 motion compensation [99]. The validation results are based on synthesisable RTL implementation of the designs using a 90nm technology together with accurate power estimation using commercial tools.
- Investigation of voltage scaling impact on pipeline stages and proposed PSI with voltage scaling (PSI-VS) method as a mean to further reduce dynamic power consumption.

The rest of the thesis is organised as follows. Chapter 2 reviews previously reported power minimisation approaches. Chapter 3 presents the proposed Pipeline Stages Insertion (PSI) method. Chapter 4 demonstrate the PSI capabilities using triple data path floating-point adder case study. Chapter 5 provides further validation of PSI with the use of MPEG-1 motion compensation case study. Chapter 6 investigates voltage scaling and proposes PSI-VS method. Finally, Chapter 7 concludes the thesis and outlines some future work.

Chapter 2

Low Power Design Literature Survey

As technology nodes scales beyond 90nm and more functionality is integrated in ASICs, power consumption is becoming more challenging to tackle, in particular for battery-powered devices. Numerous low power techniques have been proposed over the last decade and a half and while they offer good power reduction, new and efficient techniques are still needed. The aim of this chapter is to give an overview of the most relevant previous work in the area of low power techniques, emphasising those techniques related to pipelining and data path optimisation at the architecture level.

Most of the approaches in the design for low power can be classified under the following techniques [90, 130, 148]:

1. *Reducing chip and package capacitance*, which can be achieved through process development such as SOI [1].
2. *Scaling the supply voltage.* As shown in Equation 1.4 reducing voltage has quadratic effect on power and hence it is one of the most effective techniques. However, threshold voltage is not scaling as fast as supply voltage because it results in excessive leakage power at deep submicron [148]. This in turn imposes limits on how far the supply voltage can be reduced while maintaining cell drive capability.
3. *Employing better design techniques.* This will prove to be the most rewarding technique as it has not been explored fully and requires little investment compared with other techniques. Architecture level optimisation is one such design technique.

4. *Switching Activity* has direct impact on power consumption as depicted in Equation 1.4 and minimising it will help in generating designs with low power consumption.
5. *Using power management strategies*. This strategy is very much application dependent but it can be very effective [148].

There are numerous papers that encompass one or more of the above techniques. From this thesis' point of view the three techniques 2, 3 and 4 will have special focus since they are the most relevant to the work carried out in this study. Although this thesis is concerned with the analysis of pipeline stages and their impact on power-performance trade-offs, it is important to consider the wider picture. In effect, the PSI method can be regarded as data path optimisation (as will be shown in chapter 4) and therefore this chapter will review data path optimisation techniques operating at the architecture level. The remainder of the chapter is organised as follows. Section 2.1 reviews data path low power optimisation techniques and section 2.2 outlines the work reported on low power pipelining techniques. Finally, section 2.3 concludes this chapter.

2.1 Data Path Low Power Optimisation Techniques

This section reviews papers related to data path optimisation at the architecture level with the aim of reducing power consumption focusing particularly on floating-point data path optimisation. There are two reasons for this; first, floating-point hardware is now commonly integrated in portable devices due to increased demand for video and graphic applications. These devices are in general battery operated and this means the hardware power consumption must be reduced to allow it to operate in those devices. Secondly, it is widely accepted that floating-point hardware is complex and using it to validate an algorithm brings confidence in terms of ensuring success of the algorithm when applied to a real-life design problem. This is because if the algorithm works on one such complex design, then there is very good chance it will work on other designs. For these two reasons, one of the case studies used in the verification of the PSI algorithm is a floating-point adder described in more detail in chapter 4. As will be highlighted in this section, floating-point optimisation for low power is in essence a data-path optimisation technique applied at the architecture level and this aligns very well with the PSI method as the low power technique in PSI can also be considered a data path optimisation technique.

Many programmers favour the use of FP hardware because of the dynamic range it offers, as well as freeing programmers from writing cumbersome manual scaling codes required of fixed-point representation. Indeed, it remains common practice [114] to prototype algorithms in FP, and then manually translate them into appropriate set of integer operations. Unfortunately, FP hardware is power hungry and is an expensive component in a processor's power budget. This has limited the use of FP hardware with most low power processors not including any FP hardware. In this section, some solutions to the power consumption of FP hardware are presented showing FP can be used without excessive power consumption.

Tong *et al* [6] explored ways of reducing floating-point power consumption by minimising the bit width representation of FP data. Specifically the author examined how software can employ minimal number of mantissa and exponent bits in FP hardware, to reduce power consumption, yet maintain a program's overall accuracy. The author argues that instead of rendering computations as integer operations, these should be translated into custom-precision FP by modifying the program to use as few FP bits as possible without violating the accuracy criteria. The relationship between program accuracy and number of bits used in the representation of their data is studied experimentally. The main finding shows that programs which manipulate human sensory inputs e.g. speech and image recognition, suffer no loss of accuracy with reduced bitwidth in the mantissa or exponent.

In Fang *et al* [20], a lightweight floating-point design flow targeting signal processing algorithms is presented. This flow differs from [6] in that it is an automated flow that takes an algorithm as input, and generates FP hardware as depicted in Figure 2.5. The main contribution of this work falls in the bitwidth optimisation engine (Figure 2.5) with a cost function related to hardware cost and power consumption.

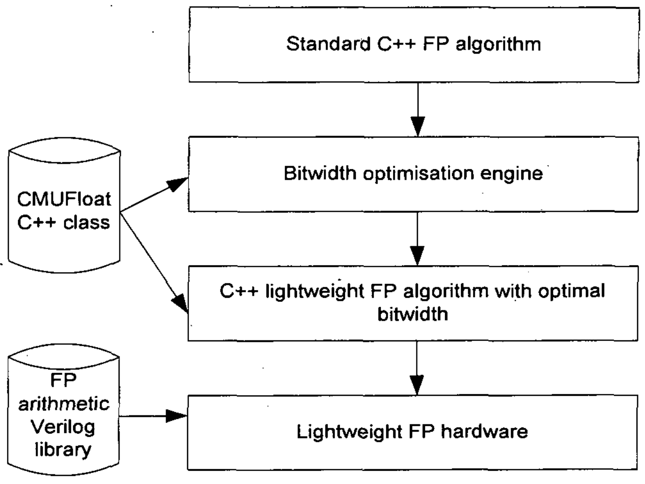

Standard C++ FP algorithm
Bitwidth optimisation engine
CMUFloat C++ class
C++ lightweight FP algorithm with optimal bitwidth
FP arithmetic Verilog library
Lightweight FP hardware

Figure 2.1 Lightweight FP System Design Flow

To automate the flow, the author created a C++ data type called CMUFloat that has a number of FP arithmetic operations with different numbers of bitwidth. Such implementation offers two advantages; firstly, it provides a transparent mechanism to embed CMUFloat numbers in an algorithm. Designers can use CMUFloat just as standard C++ data types and the overall structure of the source code is preserved. Secondly, the arithmetic operators in CMUFloat emulate the hardware implementation so giving better consistency between simulation and the final hardware implementation. With the support of CMUFloat, it is shown that both mantissa and exponent can be parameterised and the optimisation engine can determine the bit configuration during simulation. To simplify the problem further, the exponent was fixed and only the mantissa was allowed to have different bitwidth. It was noted that for a given signal processing algorithm, there are usually hundreds of FP variables. If all the variables were to be assigned to different mantissa bitwidth, the complexity of the optimisation problem will be unacceptably high. In addition, hardware implementation of all the different mantissa bitwidth will be more complex. In reality, for embedded DSPs or custom ASICs, both of which can only offer one or very few FP formats, *variable grouping* according to hardware implementation topology was proposed to reduce the complexity. Effectively this means variables associated with a given hardware shall all be set to the same mantissa bitwidth (hence variable grouping), leaving the rest of the variables to be set according to the association with their hardware.

In the previous two papers [6, 20], FP low power optimisation was presented from a design flow point of view and has proved to be quite successful at reducing power consumption. In Pillai *et al* [37, 48], the power consumption minimisation is done at the architectural level of a floating-point adder. The proposed low power FP adder called the triple data path floating-point adder (TDPFPADD) is fast, low latency and partitioned into three distinct functional paths. Among the three distinct data paths, two are computing data paths while the third is a non-computing data path. The non-computing data path or bypass data path becomes operational during those situations when the process of floating-point addition is guaranteed, to produce a result that is known *a priori.* For the other two computing paths, one is activated when subtraction of one significand from another and the exponent difference is zero or one while the other path is activated for all other cases. Note, power saving comes from the fact that at any given time, there is only one path active allowing the remaining two paths to have their activity scaled by latching the inputs with the previous values. In addition to the power saving through transition activity scaling, the proposed architecture also offers power saving due to the simplification of data paths. For example, replacing barrel shifter in one of the computing data paths with simple single level multiplexers achieves one such simplification. The results for the new architecture show that at worst, 50% power consumption reduction is achieved when compared with conventional high speed floating-point adders. Due to the advantages of this FP architecture, this research makes use of the architecture for comparison purposes.

Chandrakasan *et al* [92] present a high-level synthesis system, HYPER-LP, to minimise power consumption in application specific data path intensive CMOS circuits. The paper addresses the problem of automatically finding computational structures that result in low power consumption for specified throughput given a high-level algorithmic specification. The basic approach is to scan the design space utilising various flow graph transformations, high-level power estimation and efficient heuristic/probabilistic search mechanisms. HYPER-LP uses more than 20 different transformations, including control step reduction, operation reduction, operation substitution, resource sharing, transition activity reduction and word length reduction. For the control step reduction, the basic idea is to reduce the number of control steps, so that slower control clock cycles can be used for a fixed throughput, allowing for a reduction in supply voltage. The reduction in control step requirements is most often possible due to exploitation if concurrency. Many transformations profoundly affect the amount of concurrency including retiming/pipelining and algebraic and loop transformations. The results show that the application of a particular transformation can have conflicting effects on different components of power consumption. In addition, the application of transformations in a combined fashion usually results in lower power consumption than the isolated application of a single transformation. Operation reduction transformation in HYPER-LP targets the switched capacitance by reducing the number of operations in data and control flow graph. There is, however, a trade-off between reducing the number of operations, and the effect this has on the critical path. Resource sharing transformation aims to reduce the required amount of hardware, while preserving the number of control steps. This is possible because after certain transformations (retiming, associativity, distributivity and commutivity), operations are more uniformly distributed over the available time, resulting in a denser schedule. However, the results in this paper show that the amount of multiplexers and control logic circuitry will typically increase with more time-sharing of resources. Therefore the optimisation strategy must take into account the trade-off between interconnect capacitance and the control logic (which determines the effective capacitance being switched). Certain operations inherently require less energy per computation that others, e.g. substituting multiplication by addition. This type of trade-off is done by the operation substitution transformation. Transition activity reduction transformation tackles spurious transitions due to finite propagation delays in logic blocks. The paper shows that increasing logic depth will increase the capacitance due to glitching while reducing the logic depth will increase register power. Hence the decision to increase or decrease logic depth is based on a trade-off between glitching capacitance versus register power. Word length reduction transformation has been shown to reduce switching events, lead to faster operation and reduce the average interconnect length and capacitance. The HYPER-LP synthesis tool was applied to a number of real-world examples and the results show that up to 18% power reduction is observed for a wavelet filter using 200nm CMOS technology library.

2.2 Low Power Pipelining Techniques

This section deals mainly with research focusing on low power pipelining techniques. Since the PSI method concerns optimising pipeline stages for power, it is better to review such research in a separate section to distinguish it from the previous sections.

Zyuban and Brooks [34] present a way of analysing at gate level the key constraints for choosing an optimal pipeline depth, which directly influences the frequency target of a microprocessor. An optimisation methodology is also presented, based on both analytical models and detailed simulations, for power and performance as a function of pipeline depth. Using a benchmark which includes transaction based applications, Zyuban and Brooks show that when both power and performance are considered for optimisation, the optimal clock period is around 18FO4 (FO4: fanout-of-four is defined as the delay of one inverter driving four equally sized inverters). The authors conduct sensitivity analysis on key parameters - latch growth factor, latch power ratio, latch insertion delay, glitch factor and leakage factor - used in the energy models and their impact on optimal pipeline depth. For example, varying leakage factor shows that for future CMOS technologies unless leakage reduction techniques becomes the standard practice in the design of high-end microprocessors, high values of leakage factor may tend to shift the optimum pipeline depth towards deeper pipelines.

Another similar paper to that of Zyuban and Brooks reported by Heo and Asanovic [36], starts by defining the baseline pipeline stage model and assigning to it a depth of 24 FO4 delays which represent the current high performance processor circuit. Using simulation, the authors then investigate pipelining and its effect on switching power while making an assumption that the number of latches increases linearly with the number of pipeline stages. The results show an optimal logic depth of 6 FO4 gives the least switching power. Leakage power was also investigated with similar results to switching power. Finally, clock gating was applied to the stages and the results show optimum pipeline stage depth was 8 FO4. In summary the power-optimum logic depth is 6 to 8 FO4 and the power saving varies from 55 to 80% compared with 24 FO4 model depending on threshold voltage, activity factor and presence of clock gating.

Research papers broadly similar in context to [34] and [36] are numerous [120-129]. Recently though, pipelining has been implemented as a cost function in high-level synthesis flow with the aim of reducing power consumption. In Kim *et al* [115], a systematic approach to minimise power and maximise throughput given a constant number of pipeline stages and set of resource constraints is presented. Pipelining is used to retime operations, such that as many operations as possible take a common operand (CO) as their inputs and bind them to the same functional module so that the input activity of the shared module is reduced. The proposed pipelining technique increases the opportunity of sharing a single functional module, for CO operations. Figure 2.6 illustrates the pipelining technique for a simple second-order IIR filter [115]. Figure 2.6(a) depicts the data flow graph (DFG) of the filter without pipelining which require two multipliers to implement the two multiplication operations. Figure 2.6(b) shows the same DFG but now with two pipeline stages created by moving one of the delays from the incoming edge to the left multiplier to its outgoing edge. With this pipelining, the multiplication operations are now in separate stages allowing the sharing of a single multiplier for both operations. Since both operations have common input *s*, the shared multiplier will have reduced switching activity and therefore reduced power consumption. While experimental results show good power saving achieved using this technique, the authors does acknowledge that pipelining comes with the overhead of controller and registers due to increase in the number of pipeline stages.

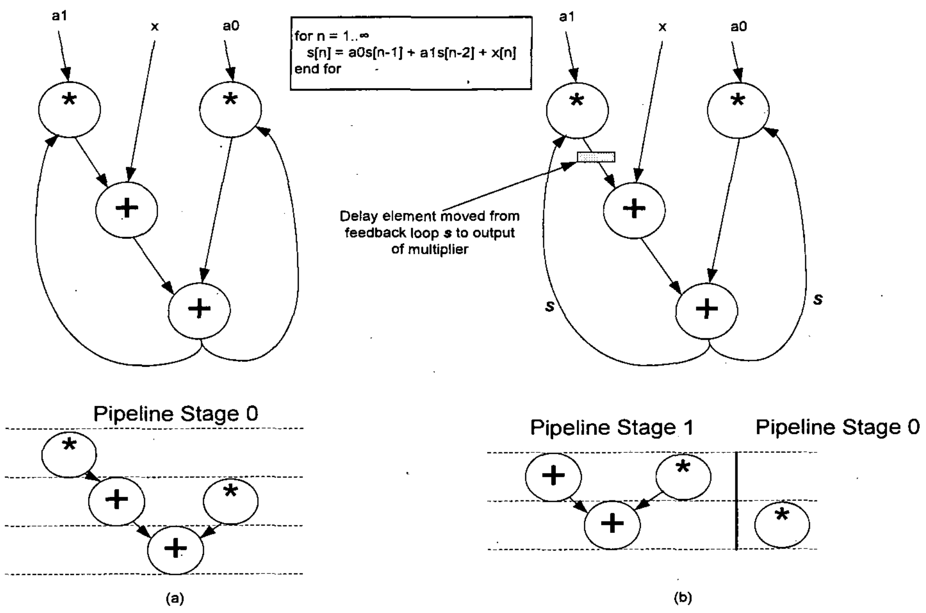

a1
x
a0
for n = 1..∞
s[n] = a0s[n-1] + a1s[n-2] + x[n]
end for
Delay element moved from feedback loop s to output of multiplier
s
s
Pipeline Stage 0
Pipeline Stage 1
Pipeline Stage 0
(a)
(b)

Figure 2.2 Pipelining of a Simple Second-Order IIR Filter

In Dao *et al* [117], the problem of pipelining was investigated at the circuit level, providing further fundamental understanding about how pipeline stage optimisation could affect the overall pipelined system optimisation. The low power pipeline solution is achieved by exploring different optimisation methods for a single pipeline stage. The dependency of pipeline stage energy on input size (total size of transistors attached to the input circuit) and output load is formulated, and the relationship is used in the optimisation of a single stage. It is stated that for a pipeline stage, the objective is to minimise energy for the performance target and under input and output load constraints. Varying input size and output load has a critical impact on the stage delay and energy. For this, the author proposed using the logical effort model [118] to investigate the delay and linear energy model to determine energy consumed in the stage. It is observed that for fixed delay (determined by processor clock period requirement) and fixed output load, the minimum energy for a given stage corresponds to the largest input size. For a larger load, it is observed that a cube of solutions (the axis for the cube is input size, energy and output load) is available to the designer with the optimal solution tending to be at a higher energy and larger input sizes. However, the energy dependence on output load for delay-optimal and energy-minimal points, remains relatively linear.

The impact of maximally pipelining a design implemented in FPGA while operating it at the original frequency was investigated by Wilton *et al* [149]. In this paper, Wilton stated that during high-speed operation, the switching of interconnect tracks laden with parasitic capacitance causes significant power dissipation. Hence, FPGA can consume up to two orders of magnitude more power than an ASIC in comparable technology. Wilton showed that by using a 130nm FPGA implementation of a 64-bit unsigned multiplier, the difference in dynamic system energy between the most pipelined variant and the least pipelined variant of the multiplier is 81%. In other benchmark circuits, this difference ranges between 40% and 82%. A 180nm FPGA was also used to implement the benchmarks and the results show that the impact on power – and hence energy per operation – is similar to that from the 130nm implementations, although less pronounced. The reduction in dynamic power consumption is because of reduced interconnects' length and therefore reduced interconnects' capacitance. These results are significant because they show that system level optimisation (choosing the number of pipeline stages) has more impact on power than a lower level physical design optimisation, which typically reduces energy by up to 23% [150]. Based on the results seen in this paper, chapter 4, section 4.3 will analyse the impact of different pipeline stages on the interconnect capacitance for a floating-point adder using 90nm, 6-metal layer standard CMOS technology library. This experiment will provide analytical results that will help in understanding whether ASIC technology and tools gives similar dynamic power reduction results to those seen in FPGA.

In Yi and Woods [151], a synthesis environment for hierarchical intellectual property (IP) core design was presented. The environment addresses the challenges in integrating IP cores that have been optimised for specific functionality and not for current system level requirement. This calls for an efficient hierarchical approach to integrate cores with limited access to internal structures. The paper presents the IRIS synthesis tool as a mean of inserting pipeline stages at the system level while taking care of any registers already existing in the original IP cores. For any IP that includes registers, the process of determining the pipelining period at the system level can be a complex task as the pipelining period for the core is determined in advance. To simplify the problem, IRIS assumes the pipelining period of the core to be one. This allows IRIS to change it to the maximum period of all cores. IRIS then proceeds by effectively inserting pipeline stages after adders and multipliers outputs at the core level. This pipelining approach is seen for FPGA to give the best performance results. The pipeline registers, imply that the data coming out of the core output would be delayed by a number of clock cycles corresponding to the number of pipeline stages. The data delay at the output becomes a target for all paths feeding to that output to have the same delay. Balancing the delay on the different paths is done by a retiming procedure, which is aware of the pipeline registers and the registers that came with the original core. At the system level, further retiming between the pipelined cores is applied so that the overall timing of system is respected. The task of pipelining and retiming gets harder when the core has multiple inputs and outputs both at the core level and system level. Although the pipelining approach in this paper is applied in a systematic way, it does appear that maximum number of pipeline stages is being targeted. When the pipelining period remains the same, excessive number of pipeline stages will increase the overall dynamic power consumption as will be shown in chapter 4, section 4.3. As the pipeline stage in this paper is applied at the outputs of multipliers and adders, it is difficult to infer whether the IRIS tool is able to insert stages at the outputs of IP cores rather than adders and multipliers. The advantage of this is that discrimination between cores' outputs becomes a possibility (when the clock period is large), and therefore total number of registers can be reduced.

Minnesota ARchitecture Synthesis (MARS) [152] applied high-level synthesis methodologies for dedicated DSP architectures. MARS employs concurrent scheduling and resource allocation algorithms that in turn implicitly perform algorithmic transformations such as pipelining and retiming. These transformations help to improve the quality of the schedule and reduce the number of hardware resources. MARS can synthesise valid architectures for simple DSP algorithms with lumped loop delays as well as complex algorithms with randomly distributed delays. The input to MARS is a DFG, which during initial scheduling of the DFG, pipelining and retiming is applied. The pipelining procedure within MARS is based on the work done by Leiserson and Saxe [91], and can be simply described as an extension to the retiming problem with the target of achieving the minimum clock period. MARS application of pipelining is systematic however, it does not consider the number of FEs outputs, and instead, pipelining is applied mainly to improve system performance and therefore implement a maximum number of stages. Chapter 3 will show that it is possible to consider FEs outputs and reduce pipeline stages while maintaining required performance.

2.3 Concluding Remarks

This chapter has given a comprehensive presentation of the different low power techniques. Section 2.1 reviewed reported approaches on the data path optimisation at the architecture level with the aim of reducing power consumption. The focus was on floating-point architectures and was shown [6, 20] that an ad-hoc design flow aiming to reduce the number of bits used in representing floating-point data, can have a significant impact on power consumption. These flows ran multiple software simulations each time, changing the number of data bits to determine if the application was still functioning as it was supposed to. Although these approaches achieved good power reduction, simulation imposed a heavy burden on the design flow and reduced its capability of investigating large designs. In [37, 48], a triple data floating-point adder architecture was proposed that had good low power features. The author makes two observations about this work. First, although the power saving would have been good at the time of publication, for today's technology with sub 90nm gate channel length and high throughput requirement, the power reduction would have to improve dramatically to meet today's power budget. The second point is that the author presented this architecture suitable for a fixed throughput without addressing power-performance trade-offs when using different pipeline stages. To solve these problems the proposed PSI method will use this architecture in chapter 4 as a case study analysing the different pipeline stages and their impact on power-performance trade-offs. It was shown [92] that different transformations operating at the high-level synthesis, could lead to the implementation of low power designs. The paper emphasised that the application of the transformations in a combined way would reduce power more effectively than the isolated application of a single transformation.

Finally, a thorough review of pipelining techniques, representing the focal point of this thesis, was presented in section 2.3. The work reported in [34, 36, 115, 117, 120-129] lacked the complete and systematic approach to the insertion of pipeline stages while approaching the issue at a lower level of the design flow. For example, Zyuban *et al* [34] tackled the pipeline stage insertion at the gate level whereas in Dao *et al* [117], the technique presented was applied at the circuit level both of which had limited success in reducing the power consumption due to lack of flexibility at these levels. Furthermore, the effect of FEs outputs and therefore the cost in terms of the number of latches on power-performance trade-offs was not investigated. In chapter 3, the pipeline stage insertion is addressed in a systematic manner at the architecture level, taking into account FEs outputs and delays and showing that this is the most appropriate level in the design flow to tackle the issue and enhancing the investigation of power-performance trade-offs. In [149], it was shown that a maximally pipelined design targeting FPGA would have shorter interconnects and therefore smaller interconnects capacitance, leading to reduced dynamic power consumption. In [151], pipelining was applied in a systematic fashion at the system level while integrating IP cores with embedded registers. This resulted in a system that was maximally pipelined. In chapter 4, it will be shown that for ASIC design using CMOS technology library, maximally pipelining a design could significantly impact on dynamic power consumption.

While the previous techniques offer good power reduction, new and efficient techniques are still needed. In chapters 3 and 6, the PSI and PSI-VS methods respectively will be presented. These methods address the issues in the previously reported approaches while facilitating power-performance trade-offs of pipelined designs at the architecture level.

Chapter 3

Pipeline Stage Insertion (PSI) Method

3.1 Introduction

Pipelining data-dominated architectures to achieve higher throughput while optimising for power has been the focus of extensive research [34-36, 44-47]. In the past, proposed approaches have targeted designs at the gate level to determine specific stage delay resulting in both power and performance optimal solution [36]. The stage delay was generally specified in terms of number of fan-out-of-four (FO4: defined as the delay of one inverter driving four copies of equally sized inverters). The algorithms used in the previous approaches [34, 36, 117] did not take into account FEs outputs and their impact on the number of latches. Moreover, no attempt was made to discriminate between the elements in terms of number of outputs when stage insertion was considered. While operating at the gate level, those approaches assume flat designs with no FE boundary (the boundary of FEs was defined in section 1.3.1 as the input/output interface of FE) resulting in almost impossible task in discriminating between elements based on their outputs. This leads to the insertion of latches to create stages at inefficient points in the design resulting in usage of excessive number of latches and hence more power consumption as will be demonstrated in this chapter, section 3.3.

This chapter presents a new systematic approach to pipelining data-dominated designs. The proposed method is non-intrusive with no special directives in the RTL code and operates at the architecture level while maintaining elements boundaries. Furthermore, a new PSI algorithm guides the insertion of latches while considering the cost of different numbers of outputs of FEs. When the system clock period is large, trade-offs between elements outputs can result in reduced number of latches, and therefore reduced power consumption. The rest of the chapter is organised as follows. Section 3.2 briefly outlines the scope of the proposed approach. Section 3.3 presents a motivational example. In section 3.4, a detail description of the underlying method for the new approach is presented. Section 3.5 addresses formulation of equations for Pipeline Stage Insertion (PSI) algorithm. Section 3.6 outlines the PSI flow. Section 3.7 briefly describes the design space exploration covered in this thesis, and finally, concluding remarks are given in section 3.8.

3.2 Scope of the Proposed Approach

The focus of the approach is to be able to reduce the number of latches used in creating stages in a pipelined architecture in order to reduce power consumption. The target architecture is a data-dominated one such as a floating-point adder. This type of architecture is widely used in DSP design and commonly found in consumer electronic devices. In general, designs incorporating DSP processors are concerned with throughput requirements, and possible pipeline depth increase is of secondary concern.

In the proposed approach, throughput is the primary target and pipeline depth increase is not analysed. In order to increase throughput, it is assumed that whatever is providing the input data (sampling unit or storage buffer) must use a clock that has the same period as that of the pipeline stages. Data arrives at the pipelined architecture at each clock period and each stage will concurrently process the data, pass the data from one stage to the next using this clock and finally the result is generated at the end of the last stage. The requirement to have input data arriving at the same rate as that of the processing time for a pipeline stage is mandatory for throughput centric architectures such as network processors and graphic engines [36].

In order to allow input data to be processed at a given system clock period (e.g. sampling clock), stage delay needs to be within the bound of the clock period. Assume, for example, two FEs A and B in a data-dominated architecture C that is under investigation for pipelining as depicted in Figure 3.1. If A and B are connected to each other and the sampling clock period is such that stage latches could be inserted at A or B outputs, this would mean meeting the clock period, and it makes sense to put the boundary of the stage at the outputs of B since this means less latches to create the stage. In effect, element A outputs have been hidden inside the stage and only element B outputs makes it to the next stage. It is these alternatives that the new approach will explore and capitalise upon when possible.

In the proposed method, these FEs are sub-blocks that perform a given function; they could contain a single logic gate or a complex function such as a signed-magnitude addition. What is important is that given a system clock period, the slowest of the FEs must have delay that is smaller than the clock period in order to stand a chance of creating pipeline stages that meet the target clock period.

So far, the discussion has concentrated on the data path with little discussion about the control signals that keep the design functioning correctly and how these signals are latched. Since one of the aims of this thesis is to investigate data-dominated architectures, control paths are therefore by definition not dominant and considered to cost less in terms of latches when compared to data paths. However, since the PSI method will be applied to real designs (as will be shown in chapters 4 and 5) and generate real low power solutions that actually work, control signals will be latched and be part of the PSI optimisation; from here on the term 'FE outputs' will be referring to data path signals as well as control signals.

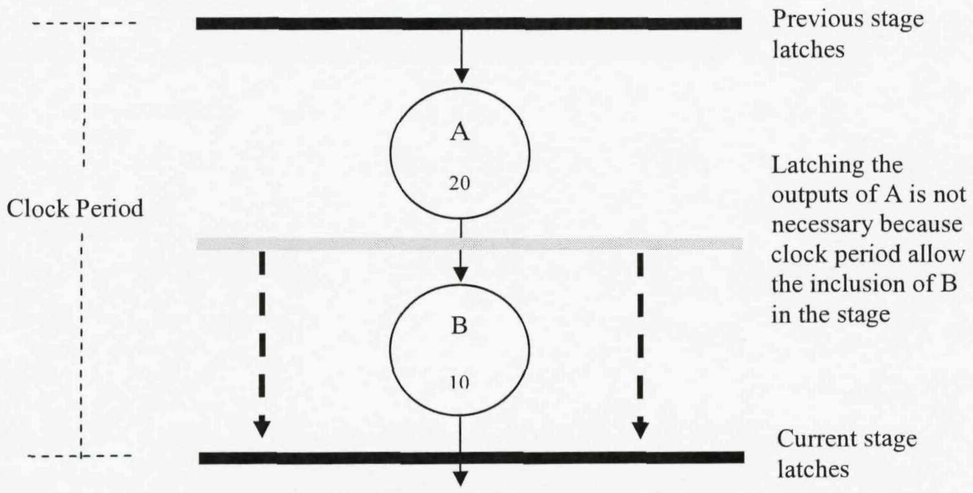

Previous stage latches
A
20
Latching the outputs of A is not necessary because clock period allow the inclusion of B in the stage
Clock Period
B
10
Current stage latches

Figure 3.1 Pipeline Stage with Element A and B in the Same Stage

3.3 Motivational Example

In section 3.2, it was stated that the primary aim of the proposed approach is the reduction of the number of latches, which in turn leads to power consumption reduction. This section shows through an example of a data-dominated architecture (floating-point adder), how pipeline stage insertion could lead to reduced power consumption without impacting on performance. Floating-point arithmetic units have been the subject of extensive research in the past. The main focus has been on delay optimisation [49-56] with some research tackling low power arithmetic architectures [20, 37, 48, 57-65].

Pillai *et al* [37, 38, 48] presented a low power architecture based on three independent data paths called Triple Data Path Floating Point Adder (TDPFPADD). Each path is enabled/disabled according to certain criteria. Pillai's architecture had some advantages:

1) Paths not involved in operation are disabled through clock gating technique which allows for transition activity scaling
2) One of the paths is a bypass path which is used for operations that have exceptions, hence saving power

3) Compared to other architectures there are also simplifications to the data paths
4) Each path is enabled using specific criteria based on: type of operation (i.e. addition or subtraction), exponent difference and operands that are of exceptional nature (i.e. +- infinity +- number, +-0 +- number or alignment shift > 24), 4), and only one path is active at any given cycle.

Figure 3.3 shows a top-level overview of the architecture. As can be seen from this figure, the architecture contains well-defined elements performing certain functional tasks that contribute to the overall floating-point adder. Each of these elements has been covered by extensive research looking into ways to reduce power while improving performance [73-85]. To highlight the impact of different stage insertion on power, Shah *el al* [66] proposed that 5-stages pipelined TDPFPADD (Figure 3.5) is used as a baseline to compare it against the ad hoc pipeline stages (Figure 3.4). What Shah [66] presented was to demonstrate the low power high throughput this architecture offers, compared with a similar floating-point adder [67]. As far as it can be established from the work described by Shah [66], the stages have been envisaged without considering the different number of outputs of FEs. The ad hoc stages shown in Figure 3.4 have been implemented by the author, with minimum effort in trying to find stage insertion points, that results in fewer number of latches compared to the 5-stages (Figure 3.5). The designs were implemented in VHSIC Hardware Description Language (VHDL) and synthesised using the Magma BlastRTL tool [33] with a target system frequency of 200MHz (for full details about the implementation see chapter 4). A 1.3V, 90nm technology library was used during the synthesis and the final netlist was analysed for power consumption in Synopsys PrimePower tool [32] using toggle data from gate-level simulation. Full clock tree synthesis was implemented for both designs using Magma BlastFusion tool [33].

The synthesis results, without applying the PSI method, show that both 3-stages and 5-stages architectures satisfy the target clock period of 5ns which corresponds to the system target frequency of 200MHz. Figure 3.6 shows the power consumption of the two architectures and Figure 3.7 depicts the area reduction. From these results, it can be seen that the 3-stages adder consumes 17.5% less power and is 25% smaller compared to 5-stages adder.

As expected, the 5-stages pipeline adder has higher power consumption compared with the 3-stages due to the extra number of latches needed to achieve the pipeline. These latches are power hungry components and reduction in the number of latches can have significant impact on power. In this example, the ad hoc 3-stages pipeline did not have an impact on performance: both 3-stages and 5-stages adders met the target clock period. On examination of Figure 3.3, it is clear that there could be a whole range of insertion points where a stage could theoretically exist and still meet the target clock period.

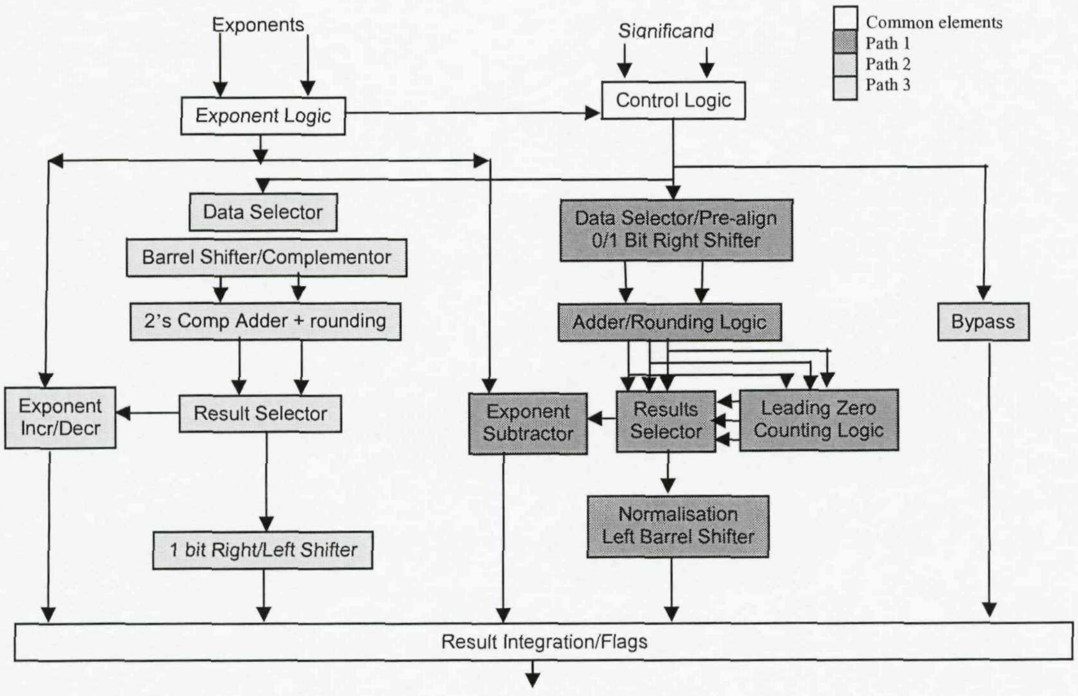


Figure 3.2 TDPFPADD Architecture [48]

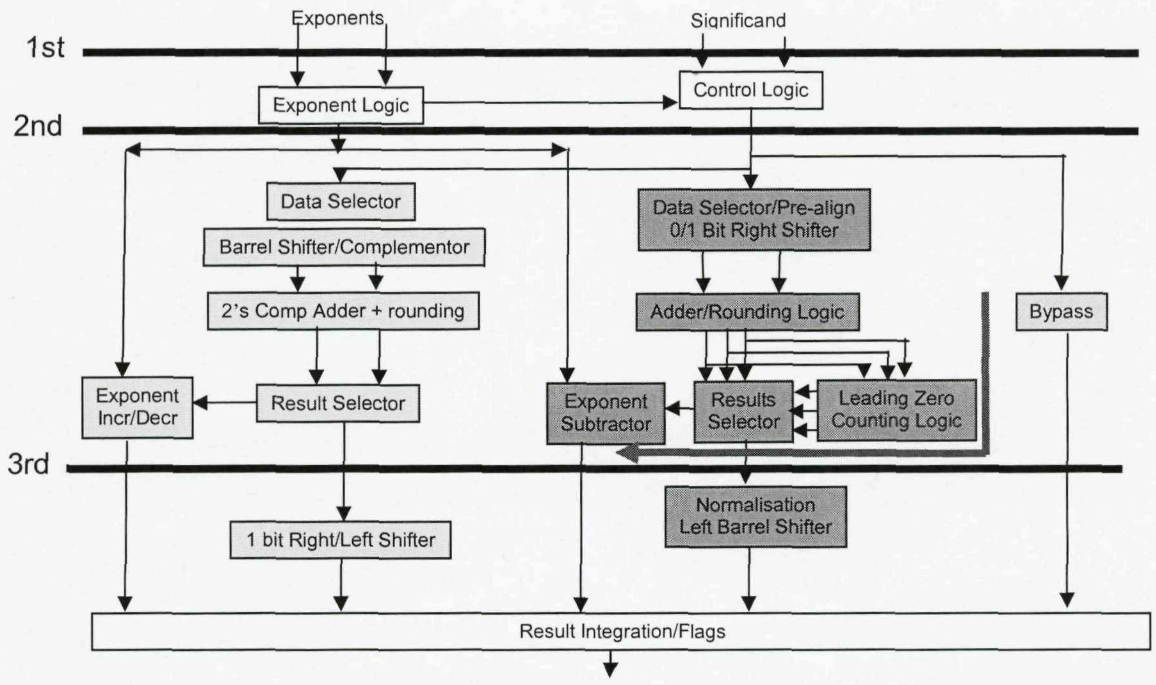


Figure 3.3 3-Stages TDPFPADD Architecture (ad hoc)

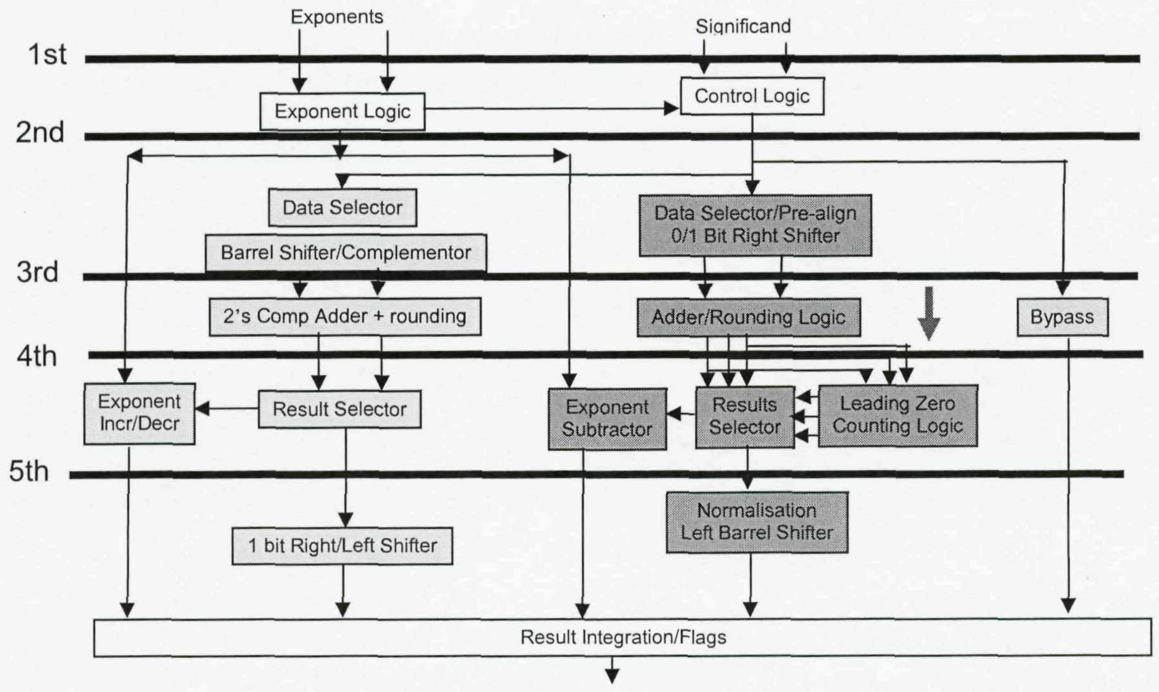


Figure 3.4 5-Stages TDPFPADD Architecture [66]

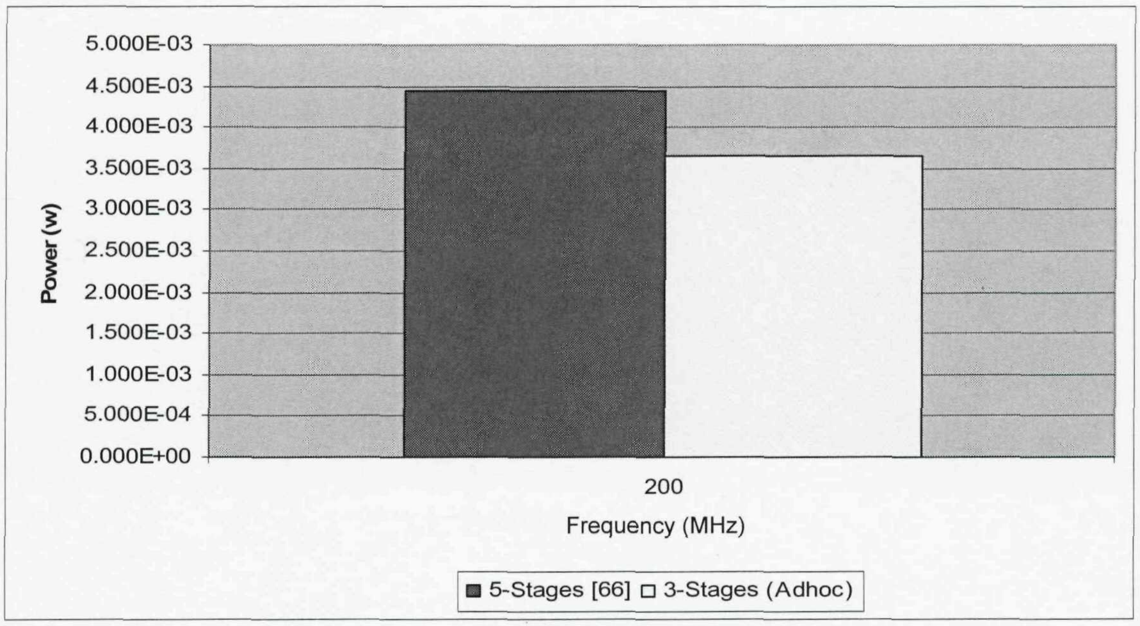


Figure 3.5 Power Consumption for 3-stages and 5-stages TDPFPADD

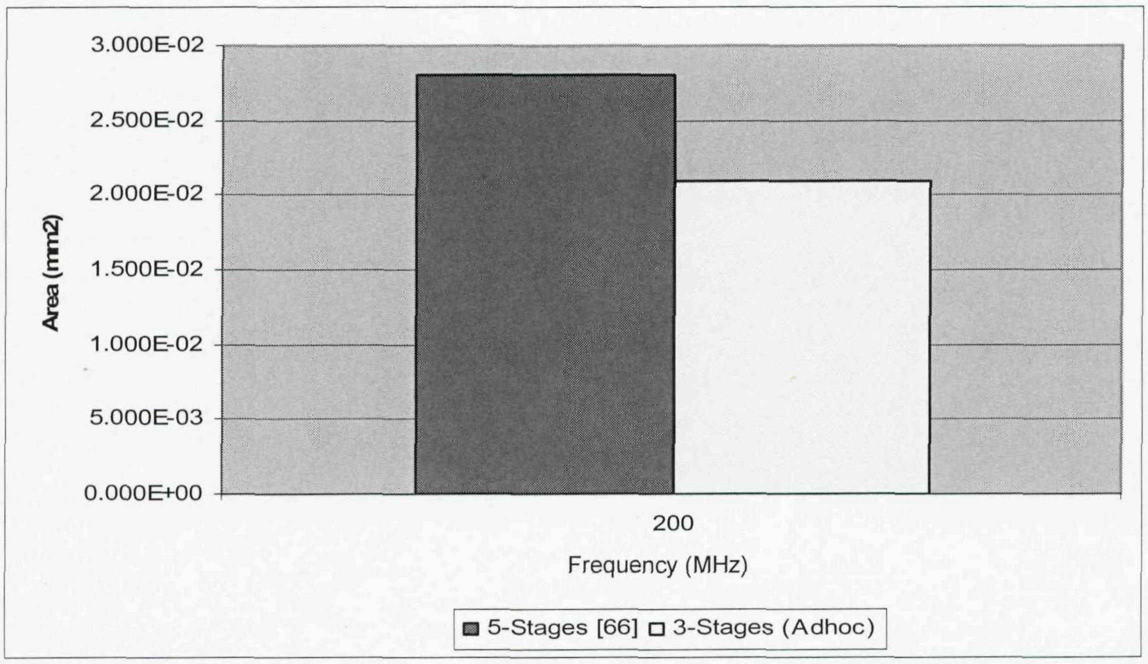


Figure 3.6 Total Area for 3-Stages and 5-Stages TDPFPADD

Note; the solution for obtaining the least power consumption could have more than 5-stages with total latches less than the ad-hoc 3-stages and the 5-stages architectures. This depends on where the stages are created and in turn, depends on the total outputs of FEs.

This motivational example has shown the potential to maintain required throughput while reducing power consumption through reduction in total latches. The impact of stage insertion and the number of latches required to implement the stages on power, has been demonstrated. Finally, this example has highlighted the need for a systematic approach to the analysis of pipeline stage insertion and their impact on power-performance trade-offs which is the focus of the next section.

3.4 Outline of PSI Method

In section 3.3, it was shown - using a typical example of data-dominated architecture (TDPFPADD) - how implementing two different pipelined TDPFPADD results in different power consumption. This section describes the method used to systematically analyse a given architecture in order to be able to make decisions about the number of pipeline stages, and their impact on power and performance. Pipeline stages are created by inserting latches at certain points in the architecture such that the stages are able to operate at the targeted clock period. The stages imbed in them one or more FEs and each of the FEs has different delay and number of outputs. As shown in Figure 3.1, the proposed method creates a pipeline stage after considering both elements' outputs and elements' delays. When the clock period is such that more than one element outputs can be a candidate for latching; the method chooses the element with the least number of outputs. This method is non-intrusive with no special directives in the RTL code.

The proposed method expects design architecture and the connections between the FEs of the architecture to be in the form of DFG. From this, the method is then able to obtain the delay for each element through synthesis. Elements delays are used in creating naive pipeline stages (these represent the least-effort pipeline stages, defined in section 1.2) for the architecture. Using pipeline stage insertion algorithm, the naive stages are examined for the possibility of removing redundant stages without affecting performance to reduce power consumption by reducing the number of latches. Required clock period, elements delays and elements outputs are all considered by the algorithm before removing stages.

The PSI method can be summarised in the following three steps:

1) Start with non-pipelined design architecture, e.g. adder shown in Figure 3.3. Generate a DFG showing the architecture FEs connections.
2) Synthesise each element of the architecture to determine its minimum delay. The architecture may operate the element at a different frequency to that used in determining the minimum delay, and therefore it is necessary to estimate the power consumption of the element. From this, an equation of power versus frequency is generated and used at the architecture power estimation with the desired frequency. Finally, naive pipeline stages are generated.
3) Using the information in step 1 and 2, naive stages are examined using the PSI algorithm to determine if stages could be removed without impacting performance.

The following section details the development of the new PSI algorithm and its usage in optimising pipeline stages.

3.5 Pipeline Stage Insertion (PSI) Algorithm

3.5.1 Equation Formulation

Section 3.4, step 3 outlined how the PSI algorithm is used in the proposed pipeline stage insertion method. This section formulates the problem of pipeline stage insertion, links the equations to power and performance analysis and finally presents the equations in generalised form. The concept behind the equation formulation is based on Integer Linear Programming (ILP) similar to that in [17, 18] which was covered in section 2.1.1.

Starting with a simple diagram of a general architecture shown in Figure 3.8 where circles represent FEs, the number of outputs of each element is given as O_i, element delay T_i (which corresponds to FE minimum delay) and the system clock period C. Before formulating the PSI algorithm, three variables are introduced:

P_i : A flag indicating the existence of a stage after element outputs O_i. Allowed values are 1 or 0, where 1 means stage does not exist and 0 means stage does exist.

W_i : The ratio of the number of next stage outputs over current stage outputs, O_{i+1} / O_i. If this inequality $\{W_i > \Phi\}$ is true then W_i is set to 1 otherwise 0. Note Φ can take positive values $\geq$ 0. The variable Φ is called the Delayed Stage Insertion Variable (DSIV) and it depends on the distribution of elements outputs. The maximum value is obtained by dividing the largest number outputs by the smallest. This is discussed in more detail in section 4.3.3.

I_i : A flag indicating a stage after outputs O_i should be inserted or not. This variable is computed from the ratio of FEs delays divided by the clock period, $(T_i + T_{i+1} + T_{i+2} + \ldots)/C$. If this ratio is > 1 then I_i is set to 1 which indicate a stage after O_i should be inserted, otherwise I_i is set 0.

The variables I_i, W_i and P_i can take only two values 0 or 1. This is because whether a stage can be inserted or not, the outcome can only be *yes* or *no* and hence fractions are meaningless. The subscript *i* in all the above variables represents stage number.

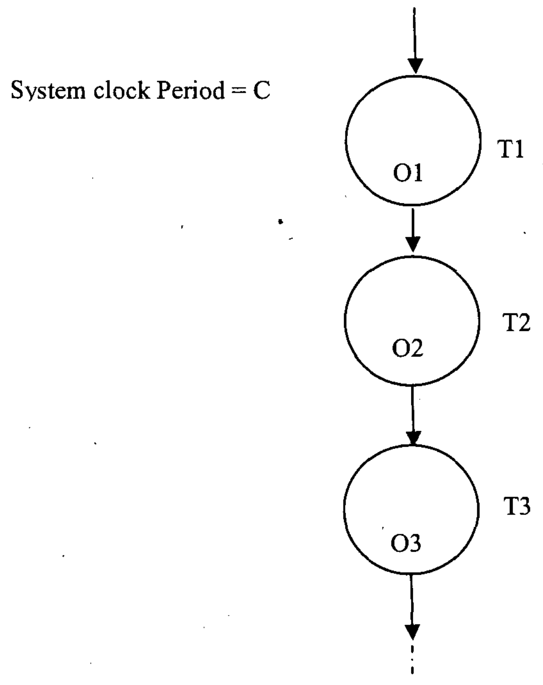


Figure 3.7 Simple Design Architecture

Consider the architecture (Figure 3.8) and with C > any of the elements delay i.e. $C > T_1, C > T_2$ and $C > T_3$ then simply insert the naive stages in this architecture to reveal the stages shown in Figure 3.9. This naive approach does not consider if two elements can be allocated to single stage when their combined delay is $< C$. If it is assumed that the inputs to the architecture are latched, then Stage 0 in Figure 3.9 always exist and no need to be considered any further. Therefore, starting with Stage 1, the flag I_1 indicates whether a stage should be inserted or not and is given as follows:-

$$I_1 = (T_1 + T_2)/C \tag{3.1}$$

If this ratio is >1, I_1 is set to 1 otherwise 0. This indicates that if I_1=1 a stage needs to be inserted otherwise the combined delay of element 1 and element 2 would violate clock period requirement. The variable W_1 is:-

$$W_1 = O_2/O_1 \tag{3.2}$$

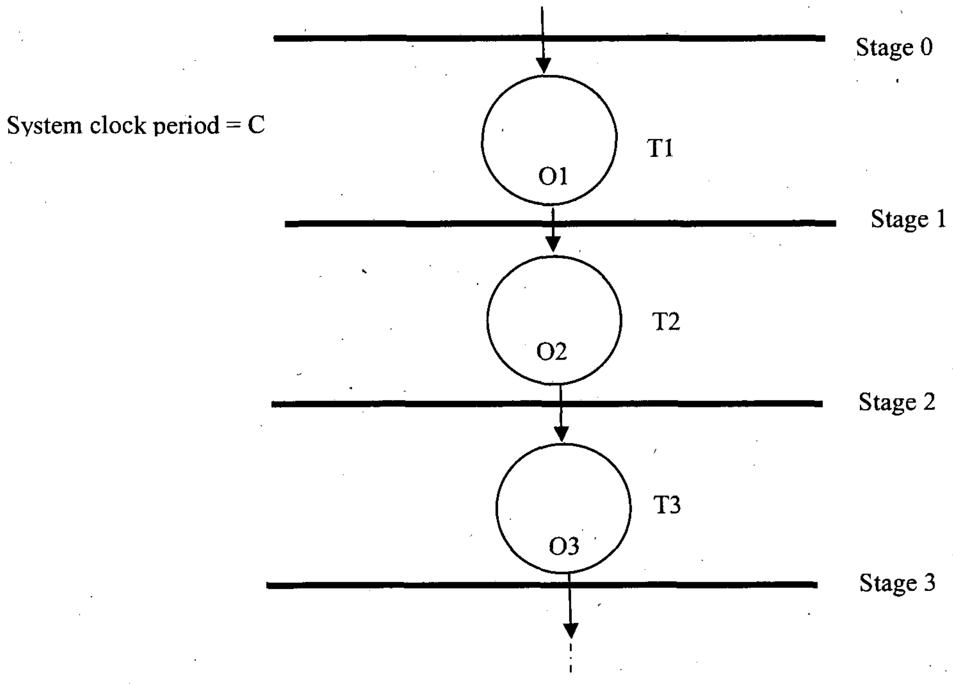


Figure 3.8 Naive Stage Insertions for the Simple Architecture

If the inequality $\{W_1 > \Phi\}$ is true, then W_1 is set to 1, otherwise 0. As an example, assuming Φ=1 and W_1=2 (e.g. $O_2 = 20$ and $O_1 = 10$) this makes the inequality $\{W_1 > \Phi\}$ true and hence W_1 is set to 1. The implication of W_1=1 for Stage 1 can be further understood from the equation for P_1 (the flag when set to 1 indicate Stage1 does not exist):

$$P_1 = |I_1 - 1| * |W_1 - 1| \tag{3.3}$$

In this example P_1 becomes 0 meaning Stage 1 has to exist because it requires less latches compared with Stage 2. The above equation is a fundamental equation in the algorithm because it indicates stage existence based on delays of elements which is represented by I_1 and also on the number of outputs given by W_1. Therefore, P_1 is aware of the cost of both timing and number of latches.

Assuming I_1=1 makes P_1=0 which means from timing point of view, Stage 1 has to exist, otherwise clock period requirement is violated as the combined delay of element 1 and element 2 is > C. In this example Φ was deliberately to be 1 such that the inequality $\{W_i > \Phi\}$ becomes true and hence W_1 is set to 1. In chapter 4 it is shown how this inequality can be tested for different values of Φ and its impact on the number of stages inserted and hence power consumption.

Consider inserting a pipeline stage at Stage 2; flag I_2 is given as:-

$$I_2 = (T_1 + T_2 + T_3)/C \quad \text{(if } I_2\text{>1 then set } I_2 \text{ to 1 otherwise 0)} \qquad (3.4)$$

However to include delay for FE 1 (T_1) in the above equation is only true if Stage 1 does not exist. If Stage 1 does exist, then T_1 must be set to zero so that it does not influence the value of I_2. The flag P_1 indicates if Stage 1 exists or not, and thus it can qualify the value of T_1 with P_1 and the new equation for I_2 is:

$$I_2 = (T_1P_1 + T_2 + T_3)/C \quad \text{(if } I_2\text{>1 then set } I_2 \text{ to 1 otherwise 0)} \qquad (3.5)$$

The above example demonstrates the choice to represent the flag P when stage exists by the value 0 rather than 1 because it is used to cancel the delay term of the previous element in the above equation. For W_2 (the ratio of number of next stage outputs over previous stage outputs) and P_2 (indicates stage existence after element outputs O_2), their equations are:

$$W_2 = O_3 / O_2 \quad \text{(if} W_2 > \Phi \text{ then set } W_2 \text{ to 1 otherwise 0)} \qquad (3.6)$$

$$P_2 = |I_2 - 1| * |W_2 - 1| \qquad (3.7)$$

Consider pipeline Stage 3. Similar analysis as for the first two stages can be performed on Stage 3 resulting in the following equations for I_3, W_3 and P_3:-

$$I_3 = (T_1P_1P_2 + T_2P_2 + T_3 + T_4)/C \quad \text{(if } I_3\text{>1 then set } I_3 \text{ to 1 otherwise 0)} \qquad (3.8)$$

$$W_3 = O_4 / O_3 \quad (\text{if } W_3 > \Phi \text{ then set } W_3 \text{ to 1 otherwise 0}) \tag{3.9}$$

$$P_3 = |I_3 - 1| * |W_3 - 1| \tag{3.10}$$

For pipeline Stage 4, the expression for I_4, W_4 and P_4 is:

$$I_4 = (T_1P_1P_2P_3 + T_2P_2P_3 + T_3P_3 + T_4 + T_5)/C \quad (\text{if } I_4 > 1 \text{ then set } I_4 \text{ to 1 otherwise 0}) \tag{3.11}$$

$$W_4 = O_5 / O_4 \quad (\text{if } W_4 > \Phi \text{ then set } W_4 \text{ to 1 otherwise 0}) \tag{3.12}$$

$$P_4 = |I_4 - 1| * |W_4 - 1| \tag{3.13}$$

From the above equations 3.1-3.13 a pattern is emerging from which general equations can be derived. For any *i*-th stage, where *i* is stage number, and when $i > 1$ the following equations are obtained:

$$\begin{aligned} I_i &= \frac{\left[\sum_{k=1}^{k=i-1} T_k \left[\prod_{m=k}^{m=i-1} P_m\right]\right] + T_i + T_{i+1}}{C^{\cdot}} \\ P_i &= |I_i - 1| * |W_i - 1| \\ W_i &= \frac{O_{i+1}}{O_i} \end{aligned} \tag{3.14}$$

When $i = 1$, the following equations for I_1, W_1 and P_1 are used:

$$\begin{aligned} I_1 &= \frac{T_1 + T_2}{C} \\ P_1 &= |I_1 - 1| * |W_1 - 1| \\ W_1 &= \frac{O_2}{O_1} \end{aligned} \tag{3.15}$$

In order for P (the flag indicating stage existence) to have only values of 1 or 0, I and W are restricted in terms of values they can take with the following rules:

$$\begin{aligned}&\text{If } I_i > 1, \text{ set } I_i \text{ to 1 otherwise 0}\\&\text{If } \{W_i > \Phi\} \text{ is true, set } W_i \text{ to 1 otherwise 0, where } \Phi \geq 0\\&\text{From the above rules } P_i \text{ can only be 1 or 0}\end{aligned} \tag{3.16}$$

Equations 3.14 and 3.15 are labelled as Pipeline Stage Insertion (PSI) equations. The flowchart in Figure 3.10 summarises the relationships between P, I and W.

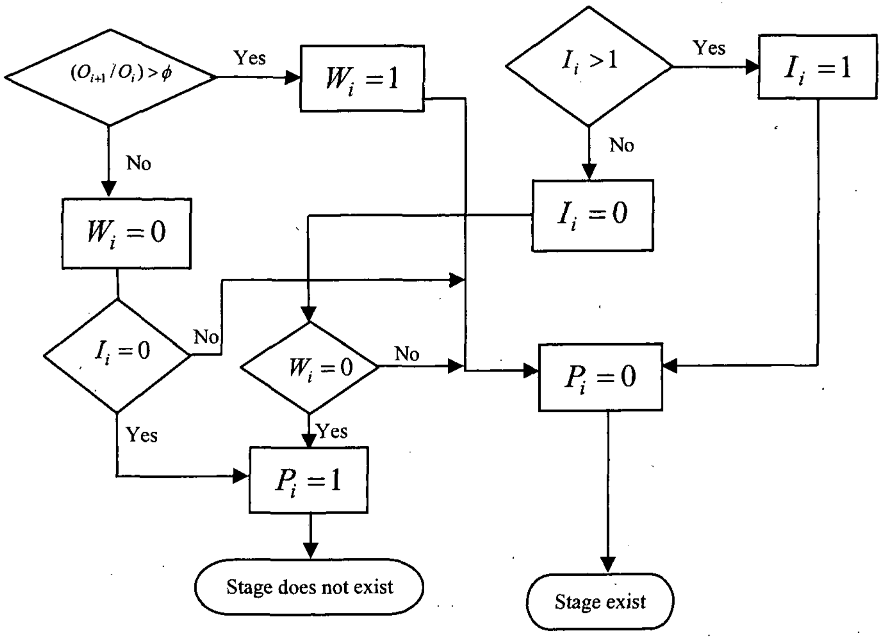


Figure 3.9 Flowchart of the Relationships Between P, I and W

To further highlight the meaning of the flags in Equation 3.14 and 3.15, consider the example architecture with naively inserted 4-pipeline stages as depicted in Figure 3.10. Assuming stage 0 to always exist, the target clock period is 50ns, $\Phi = 1$ and each of the three FEs has a delay and outputs shown in its respective circle. From Equation 3.15, the flags for Stage 1 are:

$I_1 = (T_1 + T_2) / C = (10 + 5) / 50 = 0.3$, and using the rules in 3.16, $I_1 = 0$.

$W_1 = O_2 / O_1 = 20 / 10 = 2$, using the rules in 3.16, if $\{W_1 > \Phi\}$ is true, W_1 is set to 1. For this example, the inequality is $\{2 > 1\}$ is true and therefore $W_1 = 1$

$P_1 = |I_1 - 1| * |W_1 - 1| = |1 - 1| * |1 - 1| = 0$

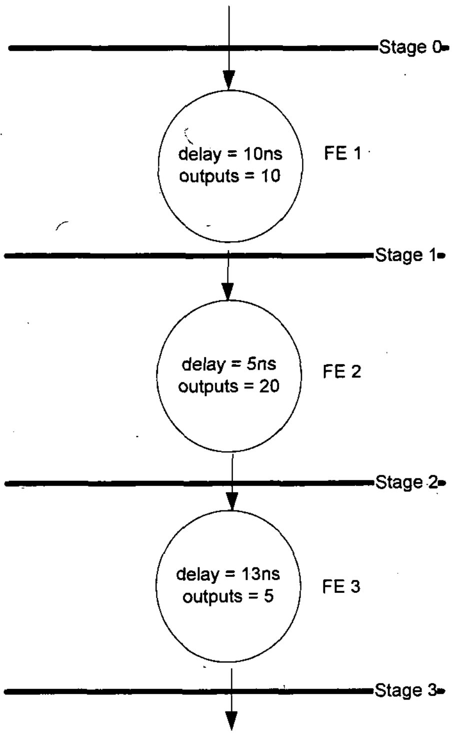


Figure 3.10 An Example Architecture with 4 Naive Pipeline Stages

Since P_1 is 0, stage 1 will have to exist. Similarly, for stage 2 the flags are calculated as follows:

$I_2 = (T_1 P_1 + T_2 + T_3) / C = (10*0 + 5 + 13) / 50 = 0.36$, therefore $I_2 = 0$ (from 3.16)

$W_2 = O_3 / O_2 = 5 / 20 = 0.25$, therefore $W_2 = 0$ (from 3.16)

$P_2 = |I_2 - 1| * |W_2 - 1| = |0 - 1| * |0 - 1| = 1$

From the above calculation, P_2 is 1 and therefore stage 2 does not need to exist, i.e. stage 2 will be removed. For stage 3, the flags are as follows:

$I_3 = (T_1 P_1 P_2 + T_2 P_2 + T_3 + T_4) / C = (10*0*1 + 5*1 + 13 + 0) / 50 = 0.36$, therefore $I_3 = 0$ (from 3.16)

$W_3 = O_4 / O_3 = 0 / 5 = 0$, therefore $W_2 = 0$ (from 3.16)

$P_3 = |I_3 - 1| * |W_3 - 1| = |0 - 1| * |0 - 1| = 1$

From the above, the value of the flag P_3 for stage 3 is set to 1, this means stage 3 can also be removed. The final optimised pipeline stages for this example architecture, only stage 1 has to exist and therefore the total registers are 10 (plus stage 0) and total stages is 2. Timing is also met for this solution, as the clock period is 50ns and the sum of FE 2 and FE3 is smaller than the clock period.

To illustrate the impact of Φ on the final solution, consider $\Phi = 4$. Since Φ value affects the final value of W, the flags P, I and W will all have to be re-evaluated for three stages as shown below:

Stage 1:

$I_1 = (T_1 + T_2) / C = (10 + 5) / 50 = 0.3$, therefore $I_1 = 0$

$W_1 = O_2 / O_1 = 20 / 10 = 2$, the inequality $\{W_1 > \Phi\}$ is false, $W_1 = 0$

$P_1 = |I_1 - 1| * |W_1 - 1| = |0 - 1| * |0 - 1| = 1$

Stage 2:

$I_2 = (T_1 P_1 + T_2 + T_3) / C = (10*1 + 5 + 13) / 50 = 0.56$, therefore $I_2 = 0$

$W_2 = O_3 / O_2 = 5 / 20 = 0.25$, the inequality $\{W_2 > \Phi\}$ is false, $W_2 = 0$

$P_2 = |I_2 - 1| * |W_2 - 1| = |0 - 1| * |0 - 1| = 1$

Stage 3:

$I_3 = (T_1 P_1 P_2 + T_2 P_2 + T_3 + T_4) / C = (10*1*1 + 5*1 + 13 + 0) / 50 = 0.56$, therefore $I_3 = 0$

$W_3 = O_4 / O_3 = 0 / 5 = 0$, the inequality $\{W_3 > \Phi\}$ is false, $W_3 = 0$

$P_3 = |I_3 - 1| * |W_3 - 1| = |0 - 1| * |0 - 1| = 1$

The significant result from the above is that P_1 is set to 1 and therefore stage 1 can be removed. This means when $\Phi = 4$, the final solution has only one stage, that is stage 0. Examining Figure 3.10, this solution is reasonable since the sum of all FEs delay will be less than the target clock period (50ns) and therefore pipeline stages 1, 2 and 3 are redundant. This last experiment highlights the need to examine a number of Φ values in order to generate a solution with the least number of registers. Chapter 4, section 4.3.2, will show how it is possible to examine a limited range of Φ values and still generate a low power solution.

3.5.2 PSI Algorithm Code

Figure 3.11 shows a pseudo-code of the PSI algorithm. Line 1 shows one of three inputs to the PSI algorithm which is the clock period C. Designers can choose to evaluate a single clock period or number of clock periods. If, for example, n clock periods were evaluated then the algorithm will iterate n times for each clock period. Line 2 shows the second input to PSI which is Φ.

```
1   Foreach clock period in C {
2      Foreach value in Φ {
3        Initialise array STAGES_ARRAY with stages from naive insertion
4        Foreach stage in STAGES_ARRAY {
5          if stage = 1 {
6             Calculate I based on current element delay and next element delay
7             Calculate W based current element outputs and next element outputs
8              if { $I > 1} {set I 1} else {set I 0}
9              if { $W > Φ } {set W 1} else {set W 0}
10             Calculate P
11         } else if stage ≠ 1{
12             Calculate I based on current and next element delays
13             Calculate W based current element outputs and next element outputs
14             if { $I > 1} {set I 1} else {set I 0}
15             if { $W > Φ } {set W 1} else {set W 0}
16             Calculate P
17         }
18       }
19    }
20 }
19  Save the value of P for each stage.
```

Figure 3.11 Pseudo-Code of Pipeline Stage Insertion Algorithm

Again designers can choose to use single value or range of values (in chapter 4 it will be shown how to limit the range of values for Φ). For example if Φ was set to m values, then the algorithm will iterate m times for each clock period. Finally, the third input to the algorithm (line 3) is an array that contains a representation of the naive stage insertions. For each stage, two entries exist in the array one for number of outputs, and the other is element delay. Lines 5 to 10 and 11 to 16 show the pseudo-code implementation of equations 3.14 and 3.15 respectively.

3.5.3 Power and Performance Estimation

Section 3.5.1 developed the set of equations to determine if a stage should exist in order to satisfy the clock period requirement. In equation 3.14, P_i flag described whether a stage after element outputs O_i exists or not. In this section the value of P_i will be used in the estimation of power consumption.

First, consider the power consumed by registers (registers are storage elements similar to latches except they are sensitive to the edge of the clock pulse rather than the level of the clock pulse [68, 69]. In industry, registers are preferred to latches because they require single-phase clock). Assuming power consumption of a single register is P_{reg}, the power consumption for Stage1 (Figure 3.9) registers is:

$$\text{Register power consumption for stage 1} = O_1 * P_{reg} * (1 - P_1) \qquad (3.17)$$

The term $(1 - P_1)$ is 0 when Stage 1 is not required, i.e. when $P_1 = 1$. In this case, the term $(1 - P_1)$ will set the above power equation to 0 since no registers are inserted. Similarly for Stage 2:

$$\text{Register power consumption for stage 2} = O_2 * P_{reg} * (1 - P_2) \qquad (3.18)$$

Therefore the total register power consumption is:

$$\text{Total register power consumption} = \sum_{i=1}^{i=\max stages} O_i * P_{reg} * (1 - P_i) \tag{3.19}$$

Now consider power consumption of combinational logic. Assume the power consumption for the combinational logic in the i-th FE to be P_{comb-i}. Note, the value of P_{comb-i} is calculated for the target clock period and then is kept constant during the PSI algorithm iterations. The total power in the architecture would be the sum of total combinational logic power and the total register power:-

$$\text{Total power consumption} = \sum_{i=1}^{i=\max stages} O_i * P_{reg} * (1 - P_i) + \sum_{i=1}^{i=\max element} P_{Comb-i} \tag{3.20}$$

In terms of performance, the equation for I_i (Equation 3.14) has two variables related to performance, namely: C (required clock period) and T_i (delay of element i). For example, increasing the clock period results in increased stage delay, which means more elements can be combined in one stage resulting in fewer registers. Increasing clock period helps reduce power but has impact on performance by reducing throughput. To improve throughput, smaller clock period is needed and generally, a compromise value is chosen such that power is kept within budget. Another way of improving performance is to swap FEs with different delays. Smaller elements delays help reduce clock period and hence improve throughput. By changing these variables (C and T_i) it is possible to develop sets of results that could be employed by the end user to adjust system clock period or choose different implementation of a FE.

3.6 PSI Flow

Section 3.4 described what the PSI method intends to achieve and in this section, the focus is on how the PSI method is actually going to be implemented. So far, this chapter has shown the development of the PSI equations (Equations 3.14 - 3.15) to optimise the naive pipeline stages. In addition, the PSI flag P_i was shown to be used in the estimation of power consumption for a pipelined architecture (Equation 3.20). What has not been mentioned yet, is how the final low power solution comes about, and for this, a flow is required to bring together all that has been developed so far. Figure 3.12 shows the proposed flow - PSI flow - with the PSI algorithm at the centre of the flow and the three steps described in section 3.4 highlighted in this figure. Starting by considering the inputs to PSI algorithm, and as can be seen from this figure, there are three inputs: (1) value(s) for Φ, (2) value(s) for evaluated clock period C and (3) data array. The data array is generated from the naive stages which capture FEs delay and their corresponding outputs (section 4.2.3 shows an example data array). The design architecture will be presented to the PSI flow as DFG, with naive pipeline stages that are generated in an ad hoc manner as will be shown in chapter 4. The output from the PSI algorithm is a value for flag P_i (defines stage existence) for each naive stage and is also used in the calculation of total register count and total stage count.

It is worth highlighting that for complex designs with many FEs, calculating combinational power consumption for each FE is run time intensive, error prone and impractical, especially if designers evaluate large numbers of clock periods (this involves calculating the power for each FE at each clock period). One simple solution for this problem is to generate a structural netlist (pure combinational logic) of the whole design from the synthesis step (Step 2), and to simulate the netlist using commercially available tools to estimate average switching activity, which is then used along with the same netlist in analysing power consumption in standard industry tools

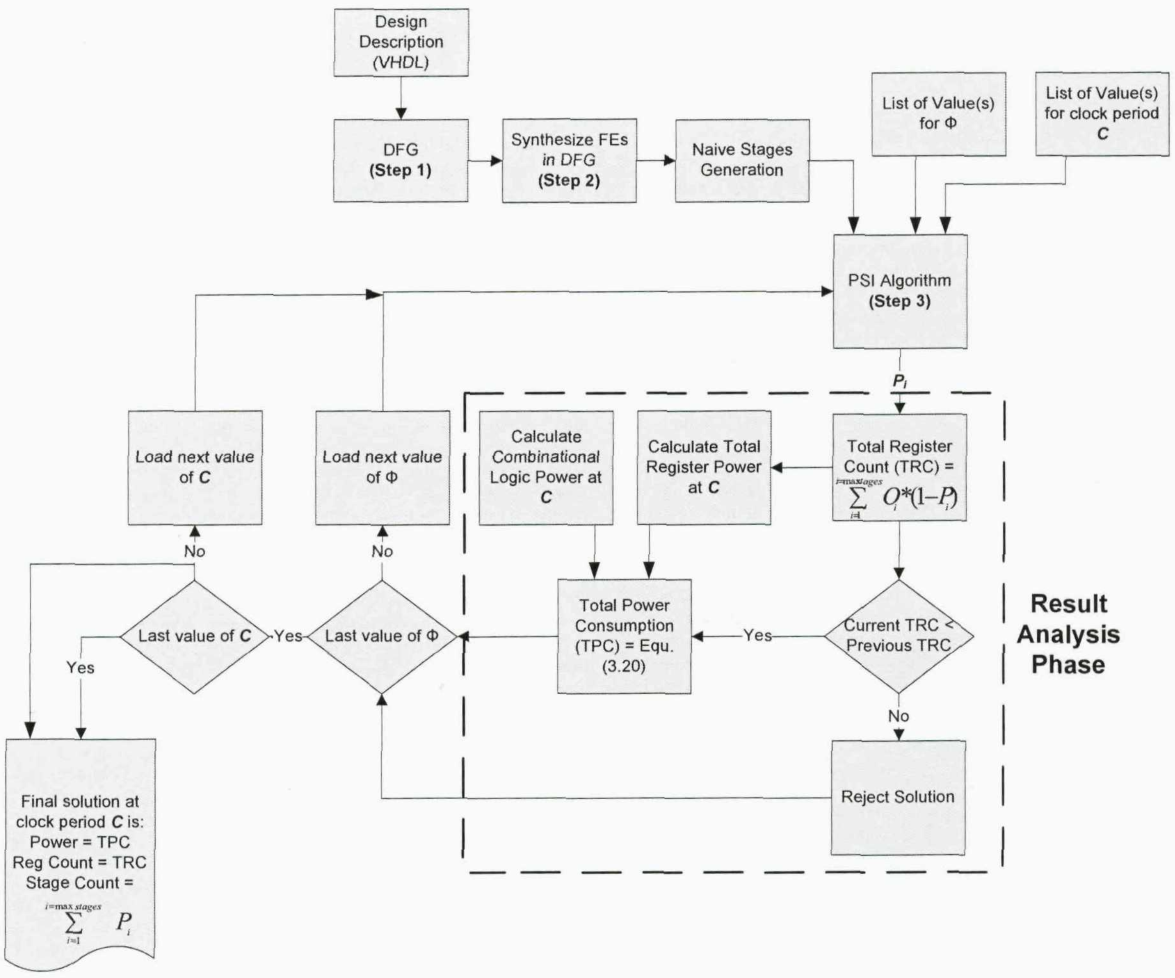

Design Description (VHDL)
DFG (Step 1)
Synthesize FEs in DFG (Step 2)
Naive Stages Generation
List of Value(s) for Φ
List of Value(s) for clock period C
PSI Algorithm (Step 3)
P_i
Load next value of C
Load next value of Φ
Calculate Combinational Logic Power at C
Calculate Total Register Power at C
Total Register Count (TRC) = $\sum_{i=1}^{i=max stages} O_i*(1-P_i)$
No
No
Last value of C
Yes
Last value of Φ
Yes
Total Power Consumption (TPC) = Equ. (3.20)
Yes
Current TRC < Previous TRC
No
Result Analysis Phase
Final solution at clock period C is: Power = TPC Reg Count = TRC Stage Count = $\sum_{i=1}^{i=max stages} P_i$
Reject Solution

Figure 3.12 PSI Flow

This method not only prevents dealing with individual FEs' power consumption but rather the power consumption of the whole design. A further simplification can be made by analysing the structural netlist for different clock periods and obtain an equation which describes the total combinational logic power as a function of clock period. This eliminates the need to know *a priori* what the designer clock period(s) is going to be. This approach – generating power equation – represents a trade-off between run time and power estimation accuracy but, bearing in mind this whole flow is supposed to be operating at the architecture level where the final implementation is still further down the design flow, the author considers any loss in accuracy to be acceptable in the light of the reasons given above. In chapters 4 and 5, it will be shown that the *combinational logic power calculation* step in the

PSI flow is often replaced with a power equation. The same is also true for the step calculating power consumption of a single register.

In Figure 3.12, the result analysis phase is depicted with the dashed rectangle. This is intended to illustrate that the final solution is determined by comparing current register count with the previous solution, and, only if it is less, will the current solution become the best solution so far. In chapter 6, the result analysis phase is modified to include design parameters such as voltage scaling. The PSI flow has been automated using a TCL script to take away the burden of manually configuring and running the steps in the flow and to eliminate user errors. Appendix A.8 shows the PSI flow script used in chapter 4.

3.7 Design Space Exploration

As already shown in the previous section, the PSI flow allows user-defined multiple clock periods with each period analysed with user-defined multiple values of Φ. Clearly, a large number of solutions will be generated from which designers can choose the solution that suits their requirement. It is this ability to generate multiple solutions that is referred to in this thesis as *design space exploration*. The exploration of a design will result in 2-dimensional design space of power consumption versus performance (clock period). At each clock period, the PSI algorithm will run through each Φ value generating a solution that meets the clock period but where possible with reduced power consumption. It is important to note that the PSI algorithm does not compromise performance and therefore the final solution is guaranteed to meet timing. This is because FE delays will be synthesised using the target technology library and a timing margin will applied to the synthesis process to take into account the pipeline stage registers. When performance allows trade-offs between pipeline stages and therefore power consumption, the algorithm will try to exploit these opportunities.

3.8 Concluding Remarks

This chapter has introduced a new pipeline stage insertion method. The three steps underpinning this method have been described in detail in section 3.4. These steps aim to reduce the number of registers needed to implement the pipeline stages. The method generates sets of results from which designers can choose the required power-performance set. Section 3.5.1 detailed the development of analytical equations and discussed their usage in the PSI algorithm. A PSI flow was proposed to bring together the methodology steps and the result analysis facilitating the power-performance trade-offs exploration.

Chapter 4

Case Study 1: Triple Data Path Floating Point Adder

Chapter 3 section 3.4 described the proposed PSI methodology and as part of the method an algorithm was developed to aid the analysis of power-performance trade-offs. This chapter applies the proposed method to a real data-dominated architecture: Triple Data Path Floating Point Adder (TDPFPADD) [48]. Floating-point adders represent one of the most used designs in DSP-oriented systems [21] and since DSPs are found in many portable electronic devices, power reduction while maintaining same performance or keeping the same power consumption while improving performance, is one of the important goals for the designers involved.

The rest of the chapter is organised as follows. Section 4.1 outlines the implementation of TDPFPADD. Section 4.2 describes the application of PSI method to the TDPFPADD. Section 4.3 presents experimental results. Finally, section 4.4 concludes this chapter.

Those readers familiar with the floating-point adder implementation can skip section 4.1 and instead go to the next section. All the information needed from section 4.1 in the subsequent sections is made available so that this chapter is self-contained.

4.1 TDPFPADD Implementation

In section 2.2.1, a number of advantages to using floating-point data representation was presented. On the other hand, a number of challenges exist in implementing such complex hardware. This section outlines the implementation of TDPFPADD [37, 48] and illustrates the different FEs in this design. The TDFPADD is an IEEE-754 compliant, single precision with support for special values: zero, denormals, infinity and Not-a-Number (NaN), underflow and overflow exceptions and round to nearest - even rounding mode. The work carried out here is similar to the implementation presented in [66], but differs in trying to simplify certain design implementation (see Exponent Logic, section 4.1.1.1).

The demand for high performance low power floating-point adders has been on the increase in recent years due to an increase in portable devices and the applications running on those devices such as image processing. As floating-point addition involves operations like pre-alignment shift, addition of significands, normalisation, rounding, correction shift and exponent evaluation, the delay and power consumption of floating-point adders is very significant. The TDPFPADD addresses delay by zero overhead rounding as will be explained in the following sections. Power reduction is achieved through transition activity scaling based on two observations: (1) the leading zero estimation circuit of floating-point adder, that handles variable number of leading zeros generated by signed magnitude addition of significands needs to be operational only during limited set of addition. (2) The operation of floating-point addition can be bypassed during certain situations.

Based on these two observations, partitioning of the floating-point design can be implemented which results in three distinct data paths and a separate activity scaled data path is envisaged for each path. Two of the paths are computing paths while the third is a non-computing path or bypass data path. The bypass path is activated when the result of addition is known *a priori* (for example, adding the floating-point representation of infinity to infinity will always generate infinity). The three paths are distinct and mutually exclusive allowing the implementation of independent clock gating to each path resulting in reduced activity. During any computing cycle, only one of the data paths is active while the states of the other two paths are maintained at their previous states. The algorithm of floating-point addition that is mapped to TDPFPADD can be summarised in Table 4.1. Note this table forms the backbone of the implementation described in the following section.

4.1.1 TDPFPADD Architecture

Figure 4.1 shows the TDPFPADD architecture obtained from [48]. Note the three data paths as highlighted in this figure are Leading Zero Anticipatory (LZA) data path, Leading Zero Bounded (LZB) and Bypass data path; the criteria for enabling each data path is shown in Table 4.2 used in the Control Logic, section 4.1.1.2.

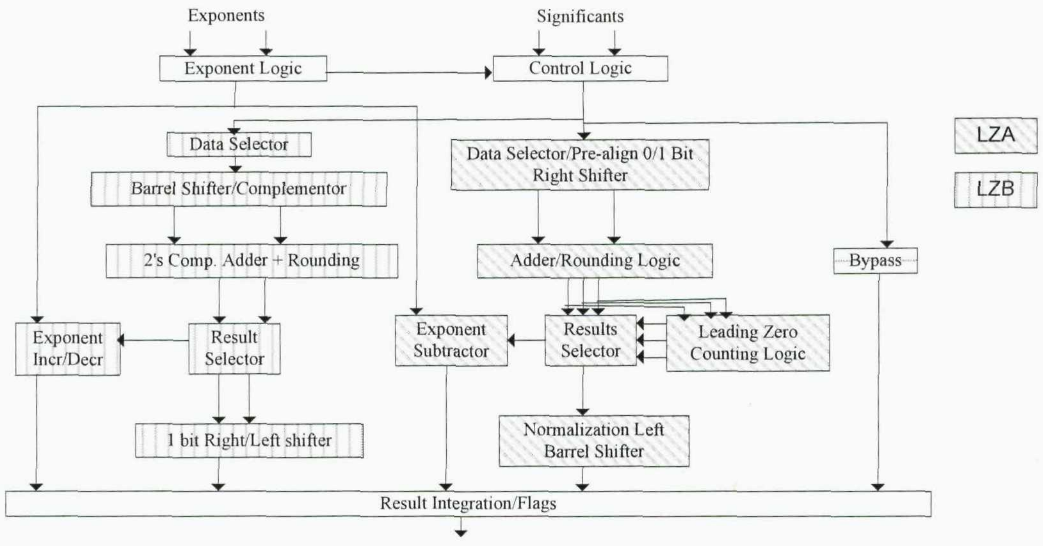


Figure 4.1 Block Diagram of the TDPFADD [48]

The adder is able to perform addition of two floating-point numbers that are: both positive, positive and negative, both negative and negative and positive.

Table 4.1 Floating-Point Addition Algorithm for TDPFPADD [48]

Step 1

Compare the exponents of two floating-point numbers for ($e1>e2$, $e1<e2$, $e1=e2$) and compute exponent difference $|e1-e2|$
Evaluate input numbers for special conditions e.g. $0 \pm$ operand, $\pm\infty\pm$ operand, NaN $\pm$ operand etc.
Select the tentative exponent of the result
Order significands on the basis of the relative magnitudes of exponents
If $|e1 - e2| >$ (significand width (p)) or special conditions, go to step 2
If $|e1 - e2| \leq 1$ and subtraction and neither $|e1 - e2| > p$ nor special conditions, go to step 3.1
If $|e1 - e2| > 1$ or addition and neither $|e1 - e2| > p$ nor special conditions, go to step 4.1

Step 2

Generate default result
Go to step 5

Step 3.1

Align the significands
Perform 1's complement addition of aligned significands
Perform speculative rounding
Count leading zeros of the different copies of result
Select result and corresponding leading zero count
Go to step 3.2

Step 3.2

Normalize significand
Compute the exponent of the result
Evaluate exception conditions, if any
Go to step 5

Step 4.1

Align the significands

Step 4.2

Perform signed-magnitude addition of aligned significands
Perform speculative rounding
Evaluate normalisation requirement, 0/1 bit left or right shift
Select result and perform normalisation
Compute the exponent of the result
Evaluate exception conditions, if any
Go to step 5

Step 5

Select the appropriate copy of result from the relevant data path

Table 4.2 Data Path Enabling Criteria for TDPFPADD [48]

Active data path	Enabling criteria	Activity scaled blocks
Bypass	Either exponent is zero or $e_{max}+1$ or (e1 - e2) > significand width (p)	All the TDPFPADD except Exponent logic, Control logic, Bypass logic and Result Integration/ Flags
LZA	No bypass and subtraction and (e1 – e2) ≤1 (leading zeros ≤ p)	Pre-alignment barrel shifter (large block)
LZB	No bypass and addition or (e1 – e2) > 1 (leading zeros ≤1)	LZA logic and normalisation barrel shifter (large block)

The remainder of this section gives implementation details as follows: section 4.1.1.1 exponent logic, section 4.1.1.2 control logic, section 4.1.1.3 bypass data path, section 4.1.1.4 Leading Zero Anticipatory (LZA) data path, section 4.1.1.5 Leading Zero Bounded (LZB) data path and section 4.1.1.6 result integration/flag logic.

4.1.1.1 Exponent Logic

This block evaluates the relative magnitudes ($e1 > e2$, $e1 < e2$ or $e1 = e2$) of the exponents of input floating-point numbers and calculates the absolute value of difference between them ($|e1 - e2|$). In [48], it is found that the use of 1's complement adder has unique advantage because not only it is able to calculate the absolute value of exponents' difference, but the conditional carry outputs reveal the truth of the conditions $e1 > e2$, $e1 < e2$ or $e1 = e2$ using no extra hardware. The conditional carry outputs from MSB bit position, $C_{out}(0)$ and $C_{out}(1)$ which represent the carry outputs when the adder is anticipating an input carry of 0 and 1 respectively, provide the information needed about those conditions. Table 4.3 summarises the possible values for $C_{out}(0)$ and $C_{out}(1)$ and the meaning that they reflect regarding the magnitude of exponents. Figure 4.2 shows the conditional half adder, which is the fundamental building block of the conditional adder and was first proposed in [79]. Note in [48], they use a modified version of the conditional adder but for the sake of simplicity the original version proposed in [79] is used in this case study. The power consumption of 1's complement adder compared to that of 2's complement adder implementing the same functionality is reduced by half due to less hardware [83]. The absolute value of $|e1 - e2| = e_{dif}$ is passed to the control logic to activate the different data paths and also perform the pre-alignment shift in LZA and LZB data paths. e_{dif} is also used to select the significand of the smaller number for pre-alignment shift by Data Selector. Appendix A.1 shows our implementation of this logic in synthesisable VHDL code. The VHDL implementation is done hierarchically with top-level (entity is labelled as Exp_Sub_8bits_Cout_Ctrl) representing the Exponent Logic that instantiates 2 VHDL components with the whole implementation being 600 lines of code. A VHDL testbench is also provided in the same appendix to verify the functionality of the Exponent Logic.

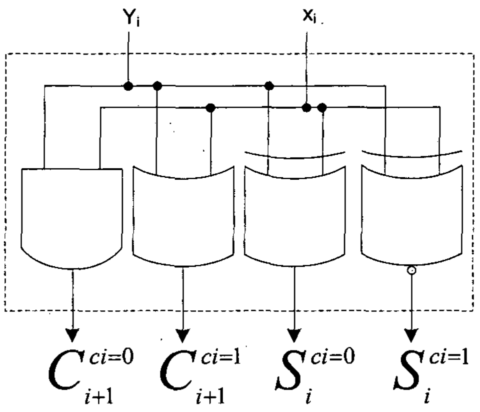


Figure 4.2 Conditional Half Adder

Table 4.3 Carry Ouputs from 1's Comp. Adder and its Interpretation

$C_{out}(0)$	$C_{out}(1)$	Implied Condition
0	0	$C_{out}(1) = 0 \Rightarrow A < B$
0	1	$A = B$
1	0	Impossible condition
1	1	$C_{out}(0) = 1 \Rightarrow A > B$

4.1.1.2 Control Logic

In this block the correct selection of data paths takes place based on exponent difference e_{dif} and the exponents themselves. Control signals are generated by this block which is passed to the appropriate data path signalling if the path should be enabled or remain at previous states. Note the control logic enables the bypass path not only during those special quantity inputs but also when the exponent difference is such that the number of pre-alignment shifts would be greater that the width of significands (p). Appendix A.2 shows our implementation of this block in synthesisable VHDL code. The VHDL implementation is 248 lines of VHDL code with top-level entity labelled as ControlLogic. A testbench is also provided in the same appendix to verify the functionality of the Control Logic.

4.1.1.3 Bypass Datapath

As already stated earlier in section 4.1, this path is a non-computing path, meaning it will only latch the data passed to it by the Control Logic until such a time as the Result Integration/Flags Logic takes this latched value. The path is enabled when e_{dif} is greater than the width of significand. Also during operation of the type 0 ± number, the larger floating-point number is latched. If both the operands are zeros, a zero is latched. For operations of the type $\pm\infty\pm$ number, infinity is latched. If NaN is produced during addition then NaN is latched. As will be shown in section 4.2 the TDPFPADD could be pipelined in which case the result of the bypass data path is not immediately presented to the outputs but rather at an appropriate cycle of the pipeline. The VHDL for this data path is shown in Appendix A.3. As expected from a non-computing path the VHDL implementation is simple with only 14 lines of VHDL code, and due its simplicity, no testbench was required to test the functionality.

4.1.1.4 Leading Zero Anticipatory (LZA) Datapath

This path is enabled when bypass conditions are not met and there is a possibility of generating variable number of leading zeros during signed-magnitude addition of significands which is only possible if the operation is subtraction (e.g. one floating-point number is negative) and the difference in exponents is either zero or one.

Pre-alignment shift is handled by the Data Selector/Pre-alignment logic (Figure 4.1) and the magnitude of the right shift depends on the exponent difference $|e1 - e2|$ while the actual significant to be shifted is determined based on the conditional carry outputs of the 1's complement adder, such that significand with larger exponent is routed directly to the significand adder, while the significand with smaller exponent is right shifted by 0/1 bit. The LSB of the shifted out significand is retained as guard (*G*) bit. This bit, along with round (*R*) and sticky (*S*) bit, ensures that when the computed digits exceed the total number of digits allowed by the floating-point format the result remains within acceptable accuracy using minimum overhead. Figure 4.3 depicts the pre-aligned significands before addition.

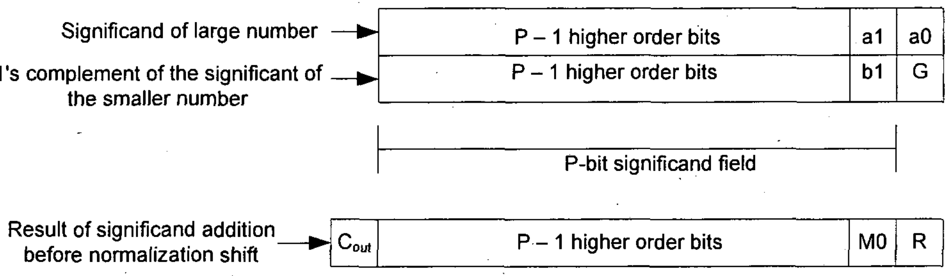

Significand of large number
1's complement of the significant of the smaller number
P – 1 higher order bits
a1
a0
P – 1 higher order bits
b1
G
P-bit significand field
Result of significand addition before normalization shift
C_out
P – 1 higher order bits
M0
R

Figure 4.3 Pre-aligned Significands [48]

G and *a0* become relevant when the exponent difference is 1 and the operation is subtraction in which case *a0* is set to 1. This in effect guarantees 2's complement addition of the aligned significands in which case a positive result is guaranteed as the significands with the smaller exponent are always complemented. When the exponent difference is zero *a0* and *G* are ignored.

Adder/Rounding logic (Figure 4.1) adds the pre-aligned significands without *a0* and *G* bits and generates conditional sum bits for bit position *M0* to MSB and conditional carry outputs by anticipating block carry inputs of 1 as well as 0. Concurrent with addition, rounding is performed so that by the time rounding decisions are known an appropriate copy of the result can be selected for normalisation shift using left barrel shifter. Figure 4.4 shows the outputs from the adder which feed to the selection logic as well as the Leading Zero Counters. Note there are three output results from the adder (*sum1*), (*sum0*) and (*not(sum0)*) which represent conditional sum bits with anticipated carry inputs of 1 and 0 respectively as well as 1's complement of *sum0*. The (*not(sum0)*) output is used during situation when exponents are equal and end around carry of 1's complement adder is zero.

The job of the Leading Zero Counter is to detect and encode the number of leading zeros for each of the three results and the appropriate encoded number is passed by the Result Selector to the Barrel Shifter. An example leading zero counter is presented in [67] and used for the TDPFPADD, as shown in Figure 4.5.

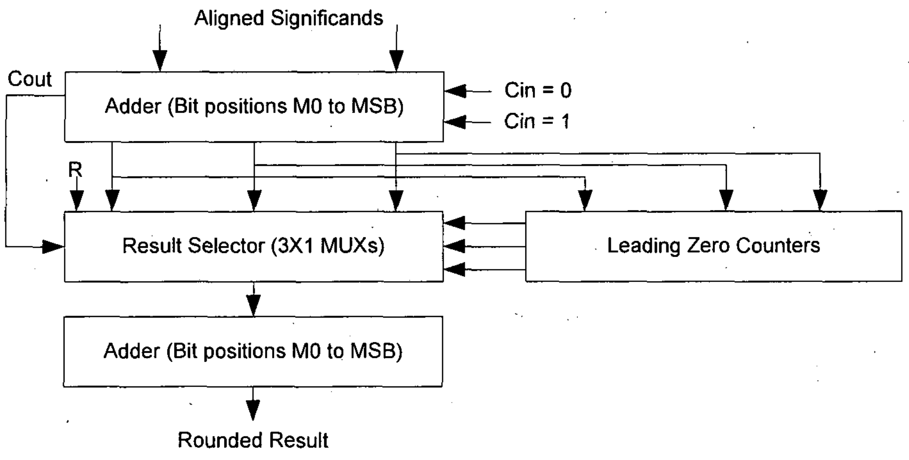

Aligned Significands
Cout
Adder (Bit positions M0 to MSB)
Cin = 0
Cin = 1
R
Result Selector (3X1 MUXs)
Leading Zero Counters
Adder (Bit positions M0 to MSB)
Rounded Result

Figure 4.4 Pre-computation, selection and normalisation of rounded results in LZA data path [48]

This counter is used in our implementation with modification to make it accept 24-bit wide input (the width of single precision significand); the code for the new implementation is shown in Appendix A.4.

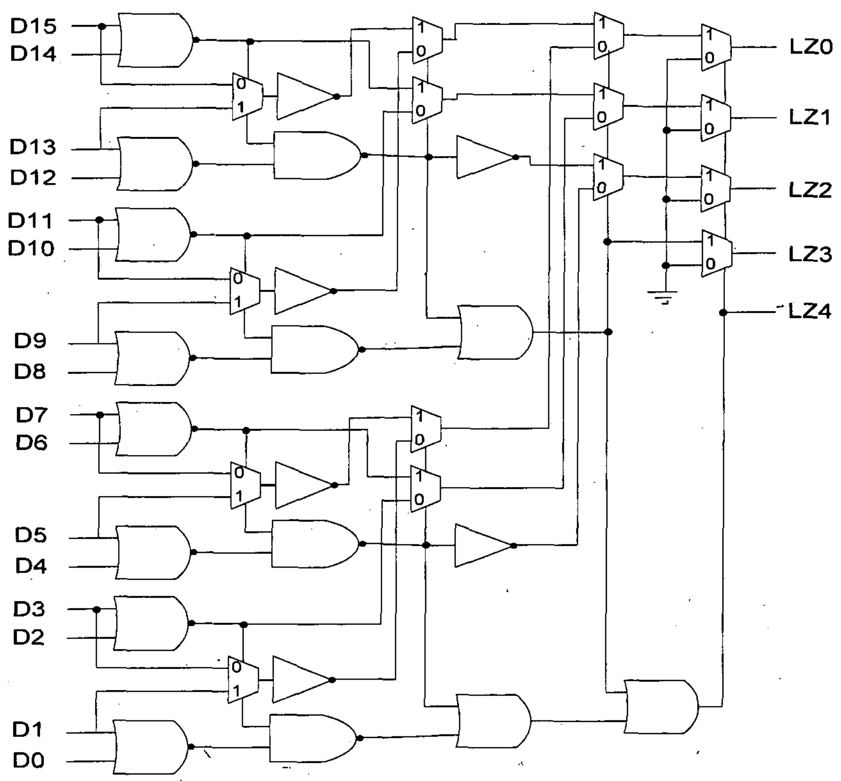


Figure 4.5 Leading Zero Count [66]

The Result Selector logic selects a sum output from the adder and its corresponding leading zero count based on the following criteria. (1) When exponent difference is zero, subtracting one significand from another would produce at least one leading zero due to equality of their leading bits and this situation does not require rounding. Because in this scenario, the exponent difference is zero, the significands were not ordered on entry to the 1's complement adder and therefore it is possible that the larger significand was subtracted from the smaller one, in which case a negative result is produced - or it could be a positive or even zero result. The way to determine which of the three outputs from the 1's complement adder is to be selected, the carry-out from the adder ($C_{out}(0)$and $C_{out}(1)$) is examined (Table 4.3 gives their meaning). If $C_{out}(0) = 1$ then it implies the smaller significand was subtracted from the larger significand and the result is positive. In this case conditional sum with an anticipated carry-in of 1 is selected (*sum1*). This is because in 1's complement addition a carry-out of 1 is then added to the result. If on the other hand $C_{out}(1) = 0$, this means the larger significand was subtracted from the smaller significand and the correct result is from the (*not(sum0)*) adder output. If however $C_{out}(0) = 0$ and $C_{out}(1) = 1$, then both significands are equal (see Table 4.3) and the result is zero. Furthermore, since in this scenario exponent difference is zero and the operation is subtraction, the final result will be zero which could be passed straight to the output without activating the normalisation shifter and therefore saving in switching power.

(2) When the exponent difference is one and the effective operation is subtraction (which is the case for enabling LZA path) and assuming significand subtraction would generate at least one leading zero, in this case rounding is not necessary and the carry input at *M0* in Figure 4.3 would be the complement of *G,* which is the *R* bit. In those cases where there is no leading zero, rounding may be required and the carry injection at *M0* is $G + (a_1 \oplus b_1)$. From these two cases, the injection carry input can be summarised in Equation 4.1.

$$Cin = \mathrm{G} + (a_1 \oplus b_1)\, LB(0) \tag{4.1}$$

Where $LB(0)$ is the leading bit of *sum0* vector and a_1, b_1 and G are the bits shown in Figure 4.2. If $Cin = 1$ then the output from the significand adder with anticipated carry in of 1 (*sum1*) is chosen; otherwise *sum0* is selected.

The normalisation shift is implemented using left barrel shifter which is capable of shifting $(p + 1)$ bits wide. In the IEEE single precision data format the maximum shift is 23 bits and any larger shifts than 23 would constitute the case where activity scaling could be enabled and a result of zero is mapped to the output. This is important since the dominant power component in a barrel shifter is as a result of the switching activity and if scaling of those activities can be made then significant power saving is achieved. A multi-stage 2X1 multiplexer-based left barrel shifter is depicted in Figure 4.6 is used which has advantages in terms of energy delay minimisation when compared to single stage implementation [38, 48, 84]. Note once the shift is complete the leading bit is discarded because in IEEE single precision this bit is guaranteed to be 1 at all times.

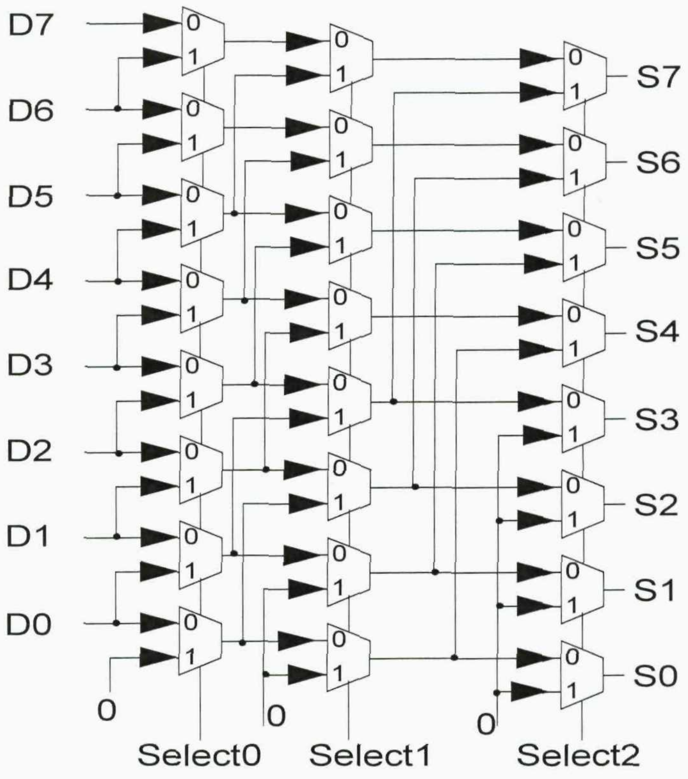


Figure 4.6 Left Barrel Shifter [66]

Finally, the last component in the LZA data path is the Exponent Subtractor and is implemented using a 1's complement conditional adder. The relevant copy of the leading zero count is subtracted from the tentative exponent using this logic. The conditional carry output from the adder based on Table 4.3 determines whether the result will be underflow or not. $C_{out}(0) = 0$ from the adder indicates an underflow condition has occurred.

The synthesisable VHDL code for LZA data path is presented in Appendix A.4. This data path represented a significant effort in terms of implementation and verification. The final implementation is 1770 lines of code and to simplify this, the VHDL architecture for this path instantiated all the TDPFPADD components (ControlLogic, LZA, DataSelect, ResultSelect, etc.). This saves the creation of an extra VHDL architecture to instantiate all the components. The top-level entity is labelled as TDPFPADD and a VHDL testbench to test the whole adder is provided in the same appendix.

4.1.1.5 Leading Zero Bounded (LZB) Datapath

This data path is enabled for all cases of addition where the sign of input floating-point numbers is the same and the significant addition guarantee the result to have ≤ 1 leading zeros. In terms of the subtraction operation, this path is restricted to exponent difference ($|e1 - e2|$) > 1.

The Data Selector in Figure 4.1 selects the significand for pre-alignment, based on the conditional carry outputs from the 1's complement adder used in the evaluation of $|e1 - e2|$ in the Exponent Logic. The significand with smaller exponent is right shifted by an amount equivalent to $|e1 - e2|$ using the barrel right shifter and the other significand is passed straight to the significand adder. Figure 4.7 shows an 8-bit right barrel shifter used in this path. Note if the amount of right shifts is such that it is bigger than the significand width p which for IEEE single precision is 24, the LZB data path is activity scaled and the Control Logic enables the bypass data path. This results in reduced power consumption.

In this data path, the right barrel shifter potentially could shift out more than one bit, and in order to implement the default IEEE round-to-nearest-even rounding scheme, the shifted out bits have to be captured with correct operation executed on them to implement the scheme. The rounding bits guard (G) and round (R) can easily be obtained by retaining the two MSB bits of the shifted out bits. The remaining shifted out bits after G and R form the sticky bit (S). Computation of S bit (this bit ensures that when the computed significand digits exceed the total number of digits allowed by the floating-point format, the result remain within acceptable accuracy using minimum overhead) is quite an involved process. In [78], they proposed a concurrent evaluation of the sticky bit through evaluation of trailing zeros of the significands of input operands for floating-point multipliers. In [48], this technique was extended to apply to the floating-point addition such that the trailing zeros of smaller significand can form the sticky bit. Our implementation makes use of the latter technique.

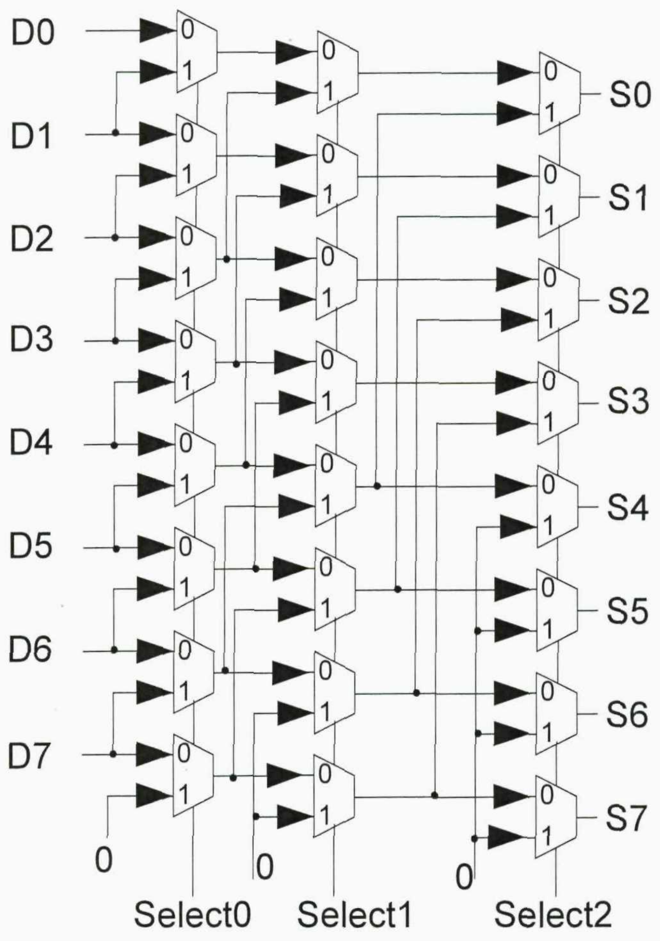


Figure 4.7 Right Barrel Shifter [66]

It was noted that the pre-alignment shift was related to the magnitude of $|e1 - e2| = \mathrm{SH}$ and since two bits are retained as G and R, the effective shift is $\mathrm{SH}_{\mathrm{eff}} = \mathrm{SH} - 2$. If the effective shift is > the number of trailing zeros (TZ) in the aligned significand then the sticky bit is set to 1 otherwise 0. This implies evaluating the condition $\mathrm{SH}_{\mathrm{eff}} > \mathrm{TZ}$ and therefore $\mathrm{SH} - 2 > \mathrm{TZ}$ or $\mathrm{SH} - 2 - \mathrm{TZ} > 0$. To evaluate this condition a 1's complement adder can be used where TZ and SH are encoded to 5-bit number shown in Figure 4.8 (a). Note SH is already available from the Exponent Logic and TZ can be evaluated using the circuit shown in Figure 4.9. The last number in Figure 4.8 (a) represent 2's complement of 2. A 3:2 compressors using Equations 4.2 and 4.3 will compress the three rows in Figure 4.8 (a) to that in Figure 4.8 (b). Using the circuit shown in Figure 4.10, the sticky bit is the output from this circuit [66].

0	SH_4	SH_3	SH_2	SH_1	SH_0
1	$\overline{TZ}_4$	$\overline{TZ}_3$	$\overline{TZ}_2$	$\overline{TZ}_1$	$\overline{TZ}_0$
1	1	1	1	1	0

(a) Input data for the evaluation of SH -2 > TZ

	C4	C3	C2	C1	C0
1	0	S4	S3	S2	S1

(b) Input data after bit compression

Figure 4.8 Data Presentation for the Evaluation of SH - 2 > TZ [48]

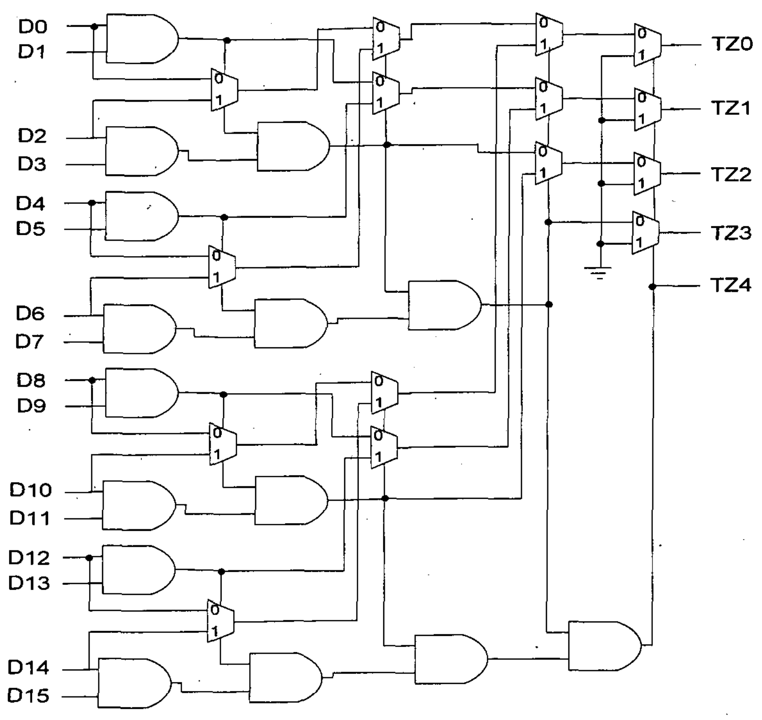


Figure 4.9 Trailing Zero Counter [66]

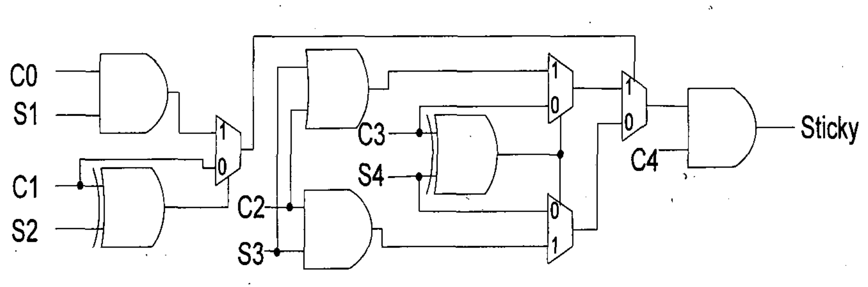


Figure 4.10 Sticky Bit Evaluation [66]

$$S_i = SH_i \oplus \overline{TZ}_i \oplus L_i \ldots\ldots\ldots\ldots\ldots\ldots \forall_i \tag{4.2}$$

$$C_i = SH_i\overline{TZ}_i + L_i(SH_i + \overline{TZ}_i) \ldots\ldots\ldots\ldots\ldots\ldots \forall_i \tag{4.3}$$

The input to the LZB significand adder in Figure 4.1 will have the data structure shown in Figure 4.11. The *G*, *R* and *S* bits are evaluated using the method described in the previous paragraph and in the case of subtraction operation, these bits are complemented and *a0* is set to 1; otherwise during addition operation they take their true value and *a0* is set to 0. The adder adds the block of data from *M1* to MSB with conditional carry input of zero and one as shown in Figure 4.12. This is done because the rounding decision is not known until after normalisation shift and to speed up the operation, both possible carry inputs are evaluated. The bits to the right of *M1* are added conditionally, based on the rounding decision and normalisation shift. It is suffice to say that the injection carrying input to *M1* position can be calculated using Table 4.4 [66] with the Result Selector selecting the correct case and output from the adder. The bits to the right of *M1* can be evaluated using Table 4.5 with M_l bit required during the case where subtraction has produced on leading zero, which means after the normalisation with 1-bit left shifter the M_{-1} bit will be the LSB.

The normalisation shift in this path is implemented using 3X1 multiplexers that facilitate the left/right shifting. The Exponent Increment/Decrement logic is used to conditionally increment or decrement the tentative exponent based on carry-out from significand adder and the generation of leading zero during subtraction.

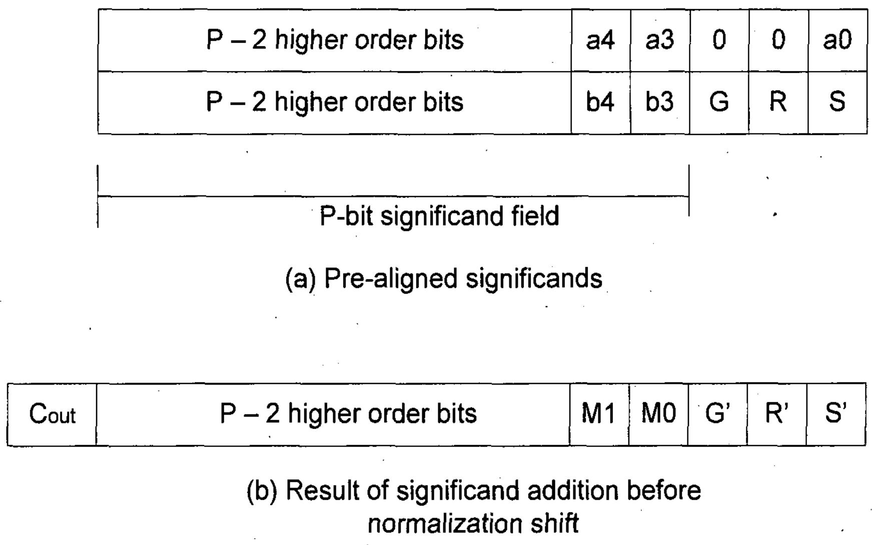


Figure 4.11 Data Representation in LZB Data Path [48]

A 1's complement adder is used (this adder consumes half the power of a 2's complement adder implementing the same functionality [83]) in this logic and the underflow/overflow condition can be easily evaluated from the conditional carry outputs of this adder. A $C_{out}(0)$ = 0 indicates an underflow condition whereas $C_{out}(1)$ = 1 implies an overflow.

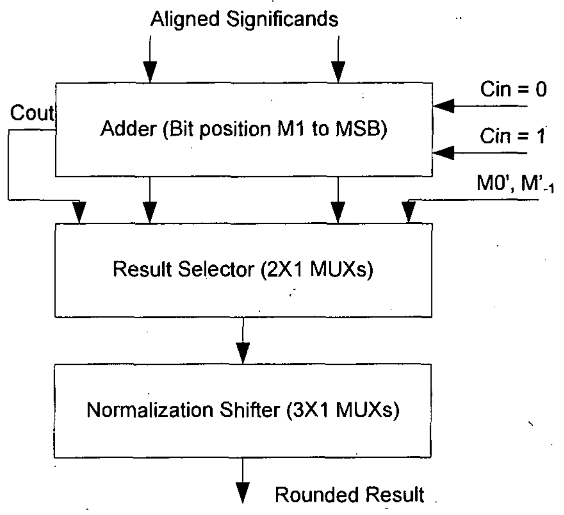

Aligned Significands
Cout
Adder (Bit position M1 to MSB)
Cin = 0
Cin = 1
M0', M'-1
Result Selector (2X1 MUXs)
Normalization Shifter (3X1 MUXs)
Rounded Result

Figure 4.12 Pre-Computation, Selection and Normalisation of Rounded Results in LZB Data Path [48]

Table 4.4 Carry Injection at M1 Bit Position [66]

Case	Rounding condition	Carry injection at M1
1	Addition with Cout = 1	$Cin1 = a3b3 + (a3 \oplus b3)(sum0(0) + G + R + S)$
2	Addition with Cout = 0	$Cin2 = a3b3 + (a3 \oplus b3)G$
3	Subtraction with no leading zero	$Cin3 = a3b3 + (a3 \oplus b3)(G + RS)$
4	Subtraction with one leading zero	$Cin4 = a3b3 + (a3 \oplus b3)G(R + S)$

Table 4.5 Rounded M0 and M_{-1} Bits [66]

Case	Rounding condition	M0'	M'_{-1}
1	Addition with Cout = 1	Not Required	Not Required
2	Addition with Cout = 0	$(a3 \oplus b3)\overline{G} + \overline{(a3 \oplus b3)}\,G(R + S)$	Not Required
3	Subtraction with no leading zero	$(a3 \oplus b3)\overline{G}\;\overline{RS} + \overline{(a3 \oplus b3)}\,G$	Not Required
4	Subtraction with one leading zero	$a3 \oplus b3 \oplus G(R + S)$	$\overline{G}R + \overline{(R + S)}\,G$

Our implementation of the LZB data path using synthesisable VHDL code is shown in Appendix A.5. This data path represented a significant effort in terms of implementation and verification. The final implementation is 1100 lines of code instantiating 8 VHDL components (SignfAdder24bitThreeSums, BarrelRight, underFlowOverFlow, M0Mminus1Generation, LZBFinalShift, PreAdderLZBpath, StickyLogic and Exp_sub_8bits). This saves the creation of an extra VHDL architecture to instantiated all the components. The top-level entity is labelled as LZB and a VHDL testbench to test this path is provided in the same appendix.

4.1.1.6 Result Integration/Flag Logic

The logic in this block enables the relevant result from the three different paths to be presented to the output of the adder with exception flags such as overflow and underflow are also taken care of by this logic. The VHDL code for this block is shown in Appendix A.6. The logic in this block is simple, requiring only 70 lines of code and no specific testbench. The verification of this block is done as part of the verification for the whole adder using the testbench in section 4.1.1.4.

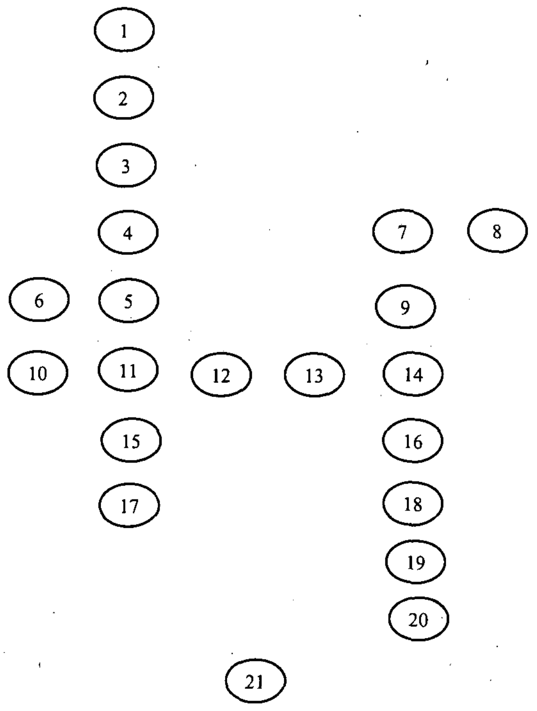
All the above blocks, that make up the TDPFPADD, have been implemented by the author. In addition, the VHDL code shown in the respective appendices has been coded by the author based on the blocks architectures already highlighted in the previous sections. As already indicated earlier in this chapter, the blocks architectures originated from the work done in [48].

4.2 Applying PSI Method to TDPFPADD

In this section, the three PSI steps (section 3.4) are applied to the TDPFPADD architecture detailed in section 4.1. Note these steps are required in the PSI flow detailed in section 3.6 to facilitate the design space exploration.

4.2.1 PSI Step 1

Step 1 is fulfilled by the fact that the chosen architecture to be analysed is TDPFPADD (Figure 4.1). To analyse the architecture in a systematic manner, Figure 4.1 is expanded to show individual Fes, which results in Figure 4.13 and their detail interconnections are shown in Figure 4.14.



No.	Name	Outputs
1	Denormal Check	74
2	Exponent Subtractor	10
3	Control Logic	101
4	Data Select	49
5	Significant Adder	74
6	Final Sign	9
7	Barrel Right	26
8	StickyBit	1
9	Pre-Significant Adder	30
10	Exponent Update	9
11	Result Select	30
12	LZCounter	15
13	Bypass logic	32
14	Significant Adder	74
15	Pre-Barrel Left	44
16	M0Minus1 Generator	39
17	Barrel Left	25
18	Exponent Subtractor	18
19	Underflow/Overflow	10
20	LZB Final Shift	57
21	Result Integrator	32

Figure 4.13 FEs needed in TDPFPADD

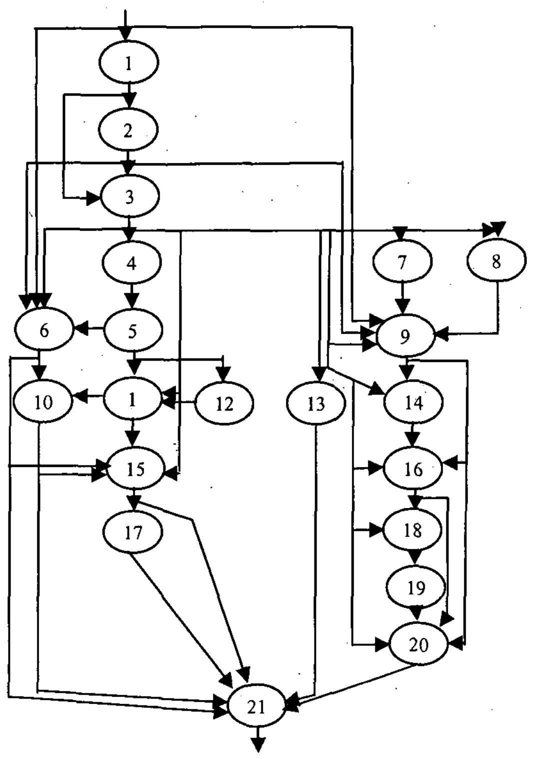


Figure 4.14 Detailed Interconnect Diagram of TDPFPADD

4.2.2 PSI Step 2

In this step, three tasks are executed: elements synthesis, elements power estimation and generating naive stages. Each task is described in the following sub-sections.

4.2.2.1 Elements Synthesis

In this task FEs are synthesised to determine their minimum delay. VHDL is used to describe element functionality and the resulting code is synthesised with Magma BlastRTL [33]. The VHDL code for the TDFPADD was approximately 3000 lines long with a full description given in section 4.1. As part of the synthesis process, target technology library is loaded at the start and in this case a 90nm, 1.3V technology library was used, as illustrated in Figure 4.15. The synthesis involves iterating through number of runs with gradually increasing input constraints (target frequency and estimated input/output delays). Once the element starts to violate the input constraints, the element delay before the violation is noted along with clock period. This period implies the smallest possible execution time for an element and hence its best possible performance. The control of input VHDL files and saving of results for each element is executed using an automated wrapper script shown in Appendix A.7. The script is 150 lines long, TCL-based, with Magma commands embedded in the script to implement the synthesis optimisation.

4.2.2.2 Elements Power Estimation

Estimating power consumption of FEs requires accurate values for interconnect capacitance and resistance. Higher accuracy parasitic is achieved by generating a floorplan for each element and the legalised placement of cells inside the floorplan using Magma BlastFusion tool [33]. Figure 4.16 shows detailed layout schematic of Significant Adder (element 5 in Figure 4.13). From the layout data, verilog netlist and SPEF (Standard Parasitic Exchange

Format: this file contains data about interconnect resistance and capacitance) are written out for each element. Synopsys PrimePower tool [32] is used in estimating power with inputs to the tool consisting of verilog netlist, SPEF, switching activity and operating frequency.

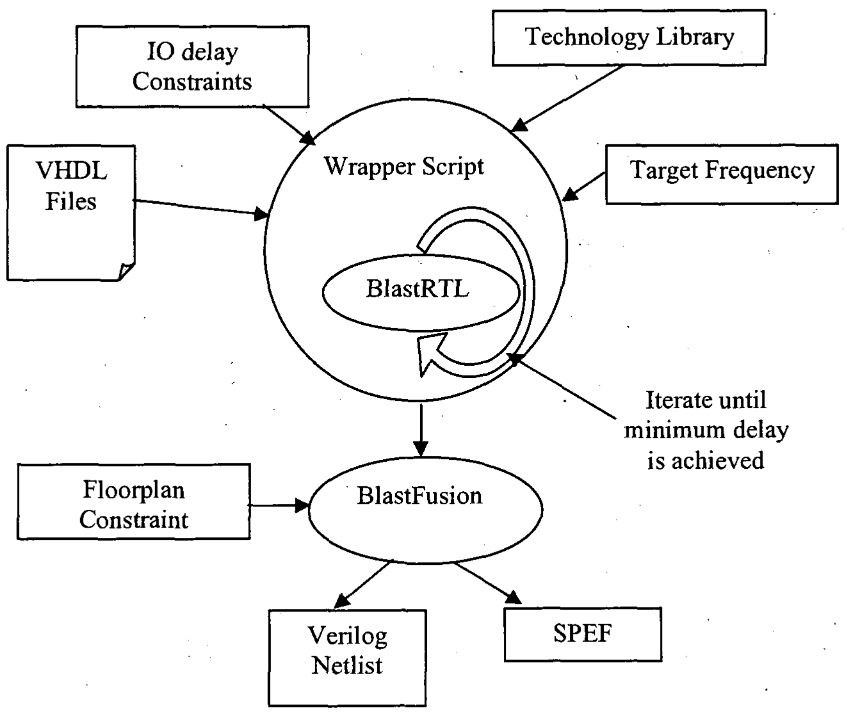


Figure 4.15 Simplified Diagram of Synthesis Process

Note the average switching activity factor was obtained from RTL-level simulation of the adder architecture using 2000 random synthetic data and was found to be 0.4294. Table 4.6 shows the power consumption of the elements when the operating frequency is the same as that used in synthesis to achieve minimum delay.

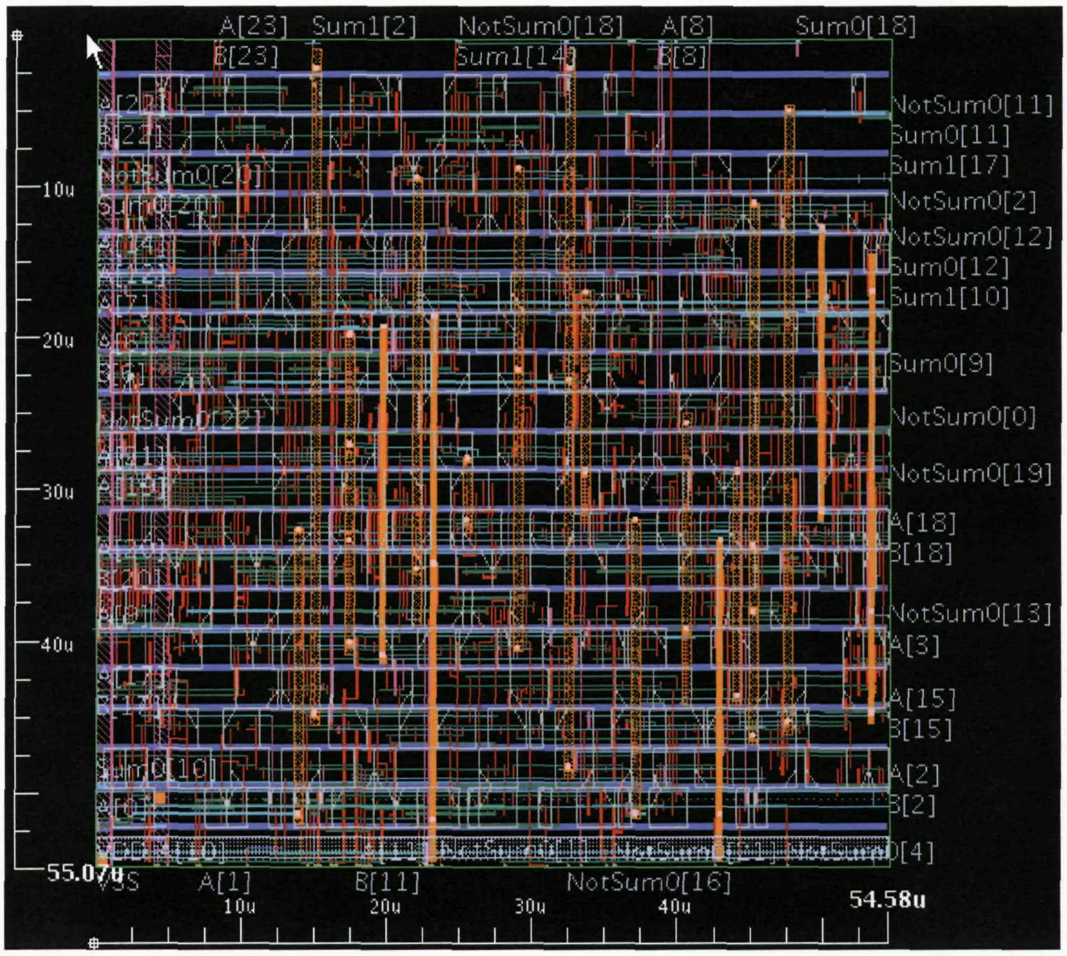


Figure 4.16 Layout Schematic of Significant Adder

Table 4.6 Power Consumption of TDPFPADD Elements

No.	**Element Name**	**Power consumption (mW) @ Best Performance***
1	Denormal Check	0.58
2	Exponent Subtractor	0.75
3	Control Logic	0.88
4	Data Select	0.30
5	Significant Adder	3.55
6	Final Sign	0.09
7	Barrel Right	2.85
8	StickyBit	0.34
9	Pre-SignificantAdder	0.49
10	Exponent Update	0.37
11	Result Select	0.99
12	LZCounter	0.34

No.	Element Name	Power consumption (mW) @ Best Performance*
13	Bypass logic	0.19
14	Significant Adder	3.55
15	Pre-Barrel Left	0.53
16	M0Minus1Generator	1.68
17	Barrel Left	2.76
18	Exponent Subtractor	0.75
19	Underflow/Overflow	0.24
20	LZB Final Shift	0.85
21	Result Integrator	1.74

* Best performance refers to clock frequency used during synthesis to achieve minimum delay.

It is possible for the architecture to operate the elements at a different frequency to that achieved during synthesis. For this reason, power consumption of elements must be estimated at different frequencies resulting in an equation used by the architecture to calculate power at the desired clock frequency. Therefore, the power consumption of the elements is estimated at five different operating frequencies resulting in plots of power verses frequency (Appendix C.1) from which generalised equations were developed. These equations are summarised in Table 4.7 (note, original plots were power verses clock period and hence the reference to clock period rather than frequency in this table).

Table 4.7 Power Equations for Each TDPFPADD FEs

No.	Element Name	Normalised Power Equations (y=power, x= clock period)
1	Denormal Check	y = 0.12e-0.08x
2	Exponent Subtractor	y = 0.12e-0.06x
3	Control Logic	y = 0.18e-0.06x
4	Data Select	y = 0.06e-0.12x
5	Significant Adder	y = 0.72e-0.05x
6	Final Sign	y = 0.02e-0.1x
7	Barrel Right	y = 0.58e-0.07x
8	StickyBit	y = 0.07e-0.05x
9	Pre-SignificantAdder	y = 0.1e-0.12x
10	Exponent Update	y = 0.08e-0.06x
11	Result Select	y = 0.2e-0.07x
12	LZCounter	y = 0.07e-0.07x

No.	Element Name	Normalised Power Equations (y=power, x= clock period)
13	Bypass logic	y = 0.04e-0.12x
14	Significant Adder	y = 0.72e-0.05x
15	Pre-Barrel Left	y = 0.11e-0.1x
16	M0Minus1Generator	y = 0.34e-0.07x
17	Barrel Left	y = 0.56e-0.07x
18	Exponent Subtractor	y = 0.15e-0.07x
19	Underflow/Overflow	y = 0.05e-0.1x
20	LZB Final Shift	y = 0.17e-0.07x
21	Result Integrator	y = 0.35e-0.08x

4.2.2.3 Generating Naive Stages

The naive pipeline stages described in section 1.2 is implemented with the aid of elements delays obtained in section 4.2.2.1 and presented in Table 4.8. These delays give an indication of whether more than one element can be combined in a single stage without making the stage delay too big. For example, implementing naive stages on Figure 4.14 results in Figure 4.17 where stage 6 imbeds five elements. The important point to note here is that, to operate the architecture at high clock frequency it makes sense - with the absence of detailed stage analysis at this point - to insert a stage immediately after each element (i.e. latching the outputs of that element). In the case where more than one element can be assigned to a single stage, the outputs of all the elements at that stage have to be latched (e.g. all elements outputs at stage 6 have to be latched). Note, Figure 4.17 shows that the minimum stage delay is dictated by element 8 because of its large delay compared to other elements - see Table 4.8.

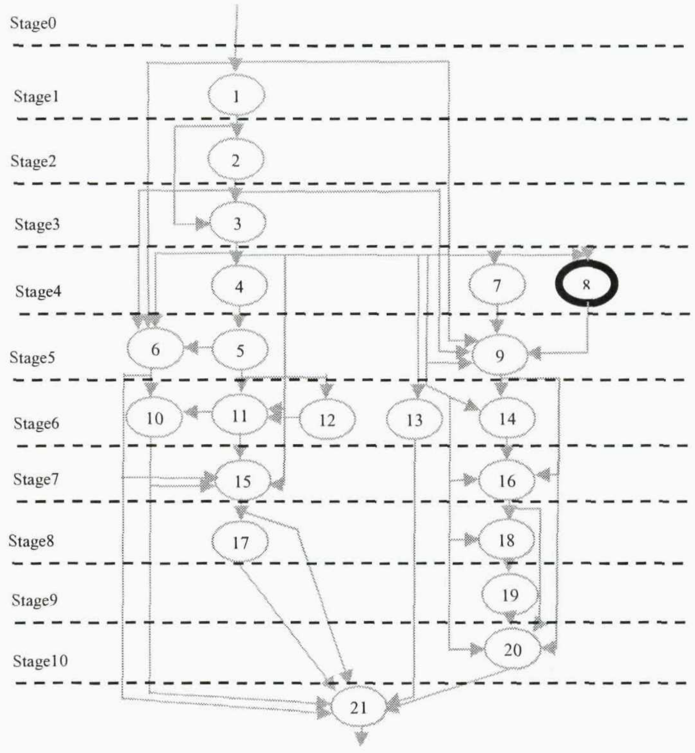


Figure 4.17 Naive Pipeline Stage Insertions for TDPFPADD

Table 4.8 FEs Delays for TDPFPADD

No.	Element Name	Delay (ps)
1	Denormal Check	900
2	Exponent Subtractor	1300
3	Control Logic	1200
4	Data Select	600
5	Significant Adder	1400
6	Final Sign	700
7	Barrel Right	1000
8	StickyBit	1500
9	Pre-Significant Adder	600
10	Exponent Update	1300
11	Result Select	900
12	LZCounter	1100
13	Bypass logic	600
14	Significant Adder	1400
15	Pre-Barrel Left	700
16	M0Minus1 Generator	1100
17	Barrel Left	1000
18	Exponent Subtractor	1100
19	Underflow/Overflow	700
20	LZB Final Shift	1000
21	Result Integrator	900

4.2.3 PSI Step 3

This step executes the PSI algorithm to determine if some of the naive stages could be removed, thus reducing power while maintaining performance. Therefore, the input to the PSI algorithm is a data array representing the naive stages, which captures the information shown in Figure 4.17. This section describes how the data array is generated and presents the results of PSI algorithm in section 4.3.

Analysis of Figure 4.17 reveals two important observations. First, across a stage one element could dominate in terms of delay masking all other elements in that stage – as they have smaller delays – and becomes the critical path of that stage. As an example, in Figure 4.17 element 8 in Stage 4 has bigger delay than element 7 and element 4 and hence dominates in terms of delay. Second, when an interconnect crosses a stage boundary, that interconnect needs to be latched to ensure correct operations. These two observations help in simplifying the naive stages (Figure 4.17) to that of Figure 4.18. In each stage of Figure 4.18 only critical path elements are shown and the total number of outputs to be potentially latched is now the sum of outputs of all elements at that stage (critical path elements or otherwise) and interconnects crossing the boundary of that stage.

From the PSI algorithm point of view, removing non-critical path elements from the diagram does not affect the operation of the algorithm because these elements do not dictate stage delay. However, the outputs of these elements are still considered when deciding if a stage could be removed or not. From now on the term "outputs" with reference to the simplified naive stages does not necessarily refer to a specific element outputs but instead refers to all elements outputs including crossing interconnects at that stage. Table 4.9 presents the data array used by PSI algorithm compiled directly from Figure 4.18. Stage0 is assumed to always exist and hence is not considered by the algorithm.

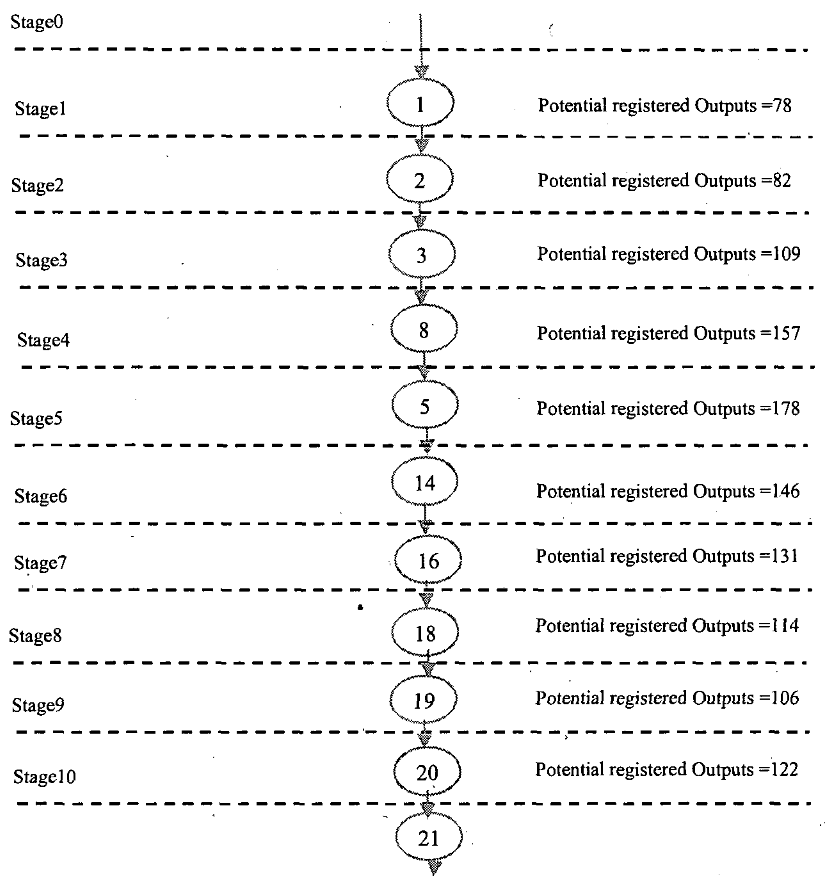

Stage0
Stage1
Stage2
Stage3
Stage4
Stage5
Stage6
Stage7
Stage8
Stage9
Stage10
1
2
3
8
5
14
16
18
19
20
21
Potential registered Outputs =78
Potential registered Outputs =82
Potential registered Outputs =109
Potential registered Outputs =157
Potential registered Outputs =178
Potential registered Outputs =146
Potential registered Outputs =131
Potential registered Outputs =114
Potential registered Outputs =106
Potential registered Outputs =122

Figure 4.18 Simplified Naive Pipeline Stages

Table 4.9 Data Array used as Input to PSI Algorithm

Stage No.	Delay (ps)	Outputs
Stage1	900	78
Stage2	1300	82
Stage3	1200	109
Stage4	1500	157
Stage5	1400	178
Stage6	1400	146
Stage7	1100	131
Stage8	1100	114
Stage9	700	106
Stage10	1000	122

4.2.4 Register Power Estimation

In order to estimate the power consumed by registers used in implementing pipeline stages, an estimate of power for a single register is needed. Since the operating frequency of the architecture could differ from the frequency used in estimating power for a single register, a range of operating frequencies is used during register power estimation. From this, an equation is developed which the architecture uses with the desired operating frequency to calculate power consumption for all the registers.

A typical register from the 90nm technology library was selected and using the data shown in Table 4.10, Synopsys PrimePower tool was set up to estimate power consumption of this register. Note, net capacitance represents the average interconnect capacitance across all the layout data for all elements. Switching activity was calculated using the method described in section 4.2.2.2. PrimePower estimated power consumption for a range of frequencies and from this data, best-fit equation was developed as shown in Figure 4.19.

Table 4.10 Data Used for Power Characterisation of a Typical 90nm Register

Technology Library	90nm
Voltage	1.3V
Average Net Capacitance	1.8 fF
Switching activity Factor	0.4294

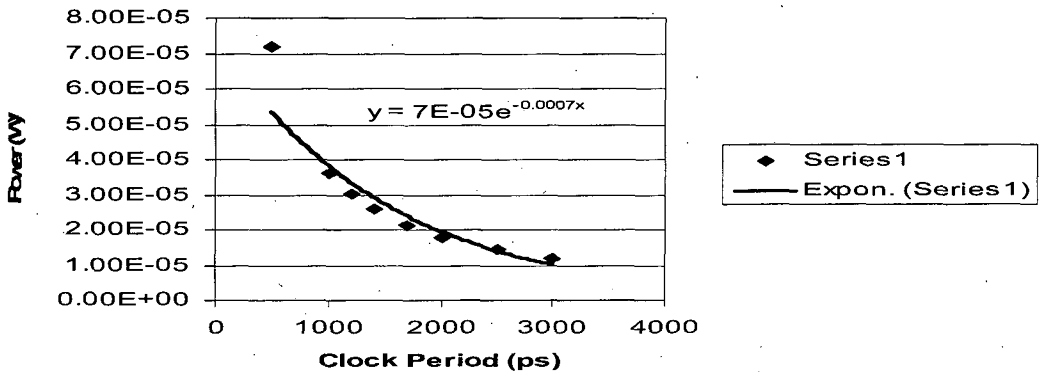


Figure 4.19 Power versus Clock Period of a Typical 90nm Register

4.3 Experimental Results

Now that all the information required by the PSI flow in Figure 3.12 (section 3.6) is available, the flow script (Appendix A.8) is executed and the result is divided into five separate sub-sections. Section 4.3.1 compares PSI results to the naive approach. Section 4.3.2 compares PSI results to 5-stages TDPFPADD [48, 66]. Section 4.3.3 investigates the effect of Φ on power-performance. Current limitation of PSI algorithm is outlined in section 4.3.4. Comparison with exhaustive search is done in section 4.3.5. Note, in all the experiments, the PSI algorithm was implemented with TCL scripting language [89], and executed on a UNIX workstation running Solaris operating system with 8G of RAM.

4.3.1 PSI Compared to Naive Approach

The input to the PSI algorithm is the naive stages represented by the data array shown in Table 4.9, and set of different system clock periods. By changing the clock period, the algorithm is able to determine for each clock period if naive stages could be removed without compromising performance (clock period). The other input to the algorithm is the value of Φ (delayed stage insertion variable), which for this experiment, two values have been evaluated, 1 and 0. Later, it is shown how the PSI algorithm generates the original naive stages, when $\Phi=0$. From Table 4.9, it is clear that the maximum operating clock period cannot be < 1500ps, otherwise element 8 would violate timing requirement; thus clock period starts at 1500ps and increasing thereafter.

Tables 4.11 and 4.12 shows the PSI results when $\Phi=0$ and $\Phi=1$ respectively. The columns of the two tables represent the naive stages, and the rows are the system clock periods under investigation. Boxes marked with 1 indicate that the PSI algorithm has retained that naive stage whereas boxes with 0 means the stage could be removed.

Table 4.11 PSI Algorithm Results for Different Clock Periods and $\Phi = 0$ (Naive)

Clock Period (ps)	Stage1	Stage2	Stage3	Stage4	Stage5	Stage6	Stage7	Stage8	Stage9	Stage10
1500	1	1	1	1	1	1	1	1	1	1
1700	1	1	1	1	1	1	1	1	1	1
1900	1	1	1	1	1	1	1	1	1	1
2100	1	1	1	1	1	1	1	1	1	1
2300	1	1	1	1	1	1	1	1	1	1
2500	1	1	1	1	1	1	1	1	1	1
2700	1	1	1	1	1	1	1	1	1	1
3000	1	1	1	1	1	1	1	1	1	1
3300	1	1	1	1	1	1	1	1	1	1
3600	1	1	1	1	1	1	1	1	1	1
4000	1	1	1	1	1	1	1	1	1	1
4400	1	1	1	1	1	1	1	1	1	1
5000	1	1	1	1	1	1	1	1	1	1

Table 4.12 PSI Algorithm Results for Different Clock Periods and $\Phi = 1$ (Proposed)

Clock Period (ps)	Stage1	Stage2	Stage3	Stage4	Stage5	Stage6	Stage7	Stage8	Stage9	Stage10
1500	1	1	1	1	1	1	1	1	1	1
1700	1	1	1	1	1	1	1	1	1	1
1900	1	1	1	1	1	1	1	0	1	0
2100	1	1	1	1	1	1	1	0	1	0
2300	1	1	1	1	1	1	0	1	1	0
2500	1	1	1	1	1	0	1	0	1	0
2700	1	1	1	1	1	0	1	0	1	0
3000	1	1	1	1	0	1	0	0	1	0
3300	1	1	1	1	0	1	0	0	1	0
3600	1	1	1	1	0	1	0	0	1	0
4000	1	1	1	1	0	0	1	0	1	0
4400	1	1	1	1	0	0	1	0	1	0
5000	1	1	1	1	0	0	0	1	1	0

It is clear from the above tables that when $\Phi=1$ (Table 4.12), there are considerable redundant stages whereas when $\Phi=0$, the PSI algorithm indicates no stage redundancy. This can be explained by examining Equation 3.16 which says W_i (the ratio of the number of next stage outputs over previous stage outputs) is set to 1 if the mathematical inequality $\{W_i > \Phi\}$ is true. When the inequality is true, W_i=1 means P_i is set to 0 (where $P_i = |I_i - 1| * |W_i - 1|$), which means a stage has to exist. Examining Table 4.9 reveals that for any *i*-th stage, W_i is always > 0 because dividing any stage *i*+1 element outputs by stage *i* element outputs results in a value > 0. Therefore, when Φ=0, the mathematical inequality $\{W_i > \Phi\}$ will always be true for any *i*-th stage and hence W_i will always be set to 1 indicating the stage is kept. This explains why in Table 4.11 all the naive stages are kept regardless of system clock period.

Considering the value of W_i, P_i and I_i for all naive stages when the clock period is 1900ps and Φ=0 as shown in Table 4.13. As expected W and P are always 1 and 0 respectively because of the reason given in the previous paragraph. However, an interesting point to note is the value for I at stages 8, 9 and 10, which indicates stages could be removed based on timing (elements delays and clock period). Similar observations are also seen for different clock periods. The conclusion from these observations is that when Φ=0, the PSI algorithm return exactly the same stages as the naive approach.

Table 4.13 P, W and I Values for each Naive Stage when clock period is 1900ps and Φ=0

Stage 1 : I = 1 P = 0 W = 1
Stage 2 : I = 1 P = 0 W = 1
Stage 3 : I = 1 P = 0 W = 1
Stage 4 : I = 1 P = 0 W = 1
Stage 5 : I = 1 P = 0 W = 1
Stage 6 : I = 1 P = 0 W = 1
Stage 7 : I = 1 P = 0 W = 1
Stage 8 : I = 0 P = 0 W = 1
Stage 9 : I = 0 P = 0 W = 1
Stage 10 : I = 0 P = 0 W = 1

For the case when Φ=1, the inequality $\{W_i > \Phi\}$ is true only for stages where W_i is > 1. For example, using Table 4.9 to calculate W_9 for Stage 9 would involve dividing Stage 10 outputs by Stage 9 outputs resulting in W_9=1.06. Thus, the inequality $\{W_9 > \Phi\}$ is true and hence W_9 is set to 1 which indicate Stage 9 is kept regardless of timing (P_9 is always 0). This is not the case for Stage 8 as W_8=0.93 and the inequality is false, resulting in setting W_8 to 0. Whether Stage 8 is kept or removed depends entirely on the clock period and element delays. This explains why in Table 4.12, Stage 9 is always kept regardless of the clock period whereas Stage 8 is kept only for certain clock periods.

4.3.1.1 Power-Performance Analysis

Figure 4.20 shows the dynamic power consumption of TDPFPADD for different clock periods. When the clock period is > 1700ps, PSI always gives better results in terms of power consumption compared to the naive approach. For example, when clock period is 2500ps, PSI stages result in power consumption of 21.3mW compared with 26mW for naive approach, which is a power reduction of 18.1%. In terms of performance, Figure 4.20 shows that PSI is able to perform as good as the naive approach.

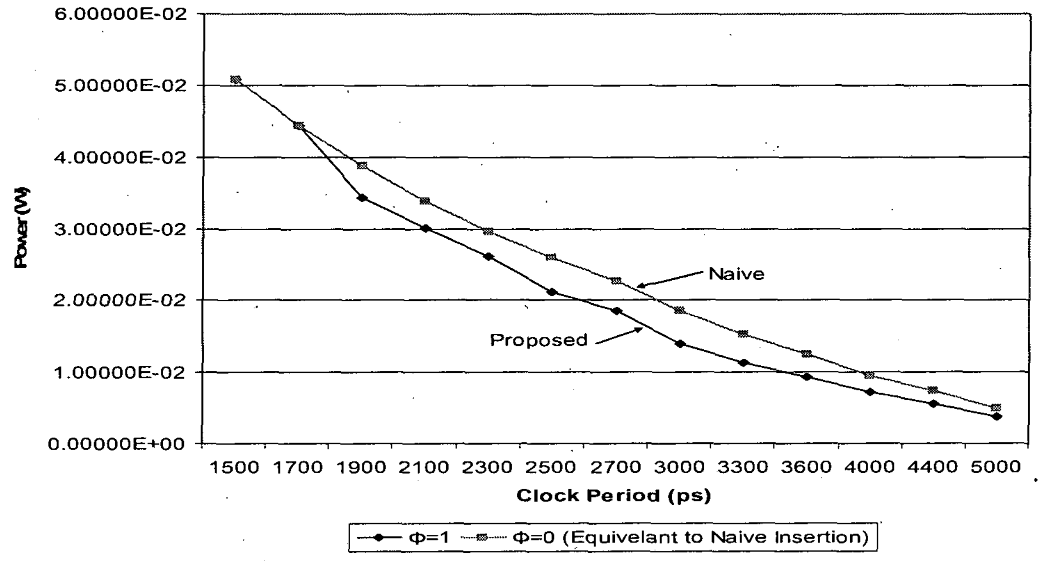


Figure 4.20 Power Versus Clock Period for TDPFPADD

Plotting the difference between the two power curves shown in Figure 4.20, the result is the curve in Figure 4.21. There are a number of observations about this figure. Firstly, the biggest difference in power consumption occurs at clock periods of 2500ps and 3000ps, which both correspond to an 18% power increase in naive stages compared to PSI. Secondly, there is a sharp rise in the curve between clock period 1700ps and 1900ps, which indicates power consumption is significant at such high frequency clocks and adding or removing registers can significantly influence power. Beyond 3000ps (relatively low frequencies), register power is no longer dominant and the difference in register numbers plays less of a part in the overall power consumption. This is confirmed by examining the contribution of registers and combinational logic to overall power in the next section. Note, similar observation was made during the analysis to reduce power by reducing the number of significand bits used in the adder (Appendix B). Thirdly, at clock periods of 1500ps and 1700ps there are no differences between the two approaches, i.e. the same number of stages exists in the naive approach and PSI. However, this is not strictly true when Φ is > 1 as will be shown in section 4.3.3.

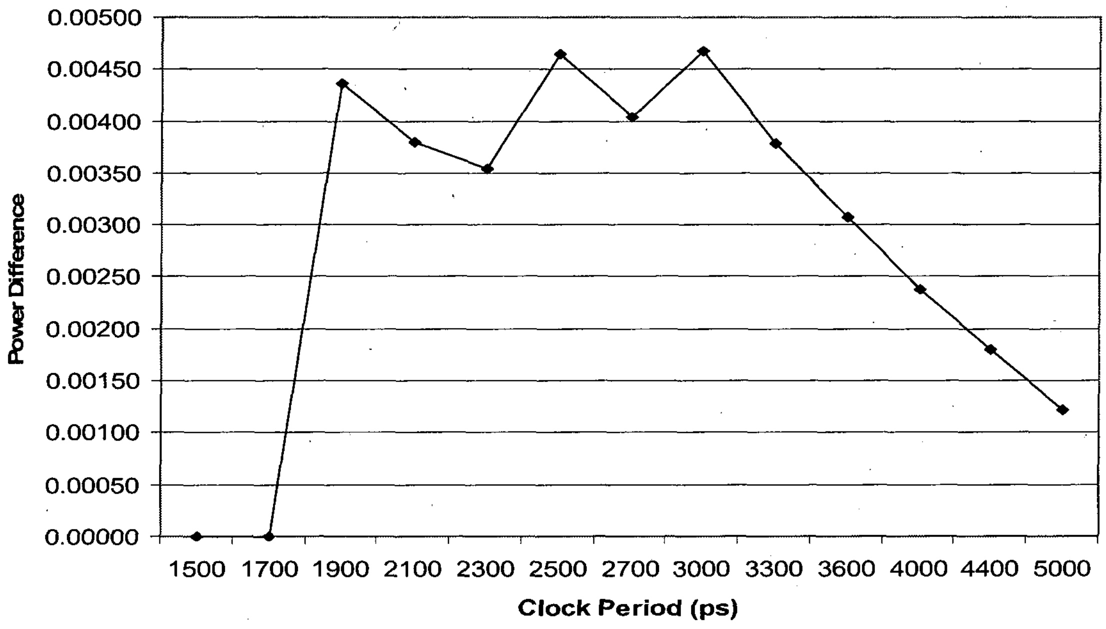


Figure 4.21 Power Consumption Difference between Naive and PSI Approach

Finally, to summarise the trend in Figure 4.21, at smaller clock periods (< 1700ps), there are less opportunities to remove pipeline stages and as the clock period increases (> 1700ps), the PSI finds more opportunities to remove redundant stages. For larger clock periods (> 3000ps), register power becomes less dominant in the overall power consumption and removing registers does not impact on power consumption significantly.

4.3.1.2 Power Consumption of Registers and Combinational Logic

Figure 4.22 show the power consumed by registers and combinational logic for PSI stages. This graph confirms that register power is the dominant part at high frequencies and when clock frequency is reduced, a crossover point occurs where combinational logic starts to consume more power than registers. Note, the crossover point happens at a fairly high frequency range corresponding to clock period range of 2700-3000ps. The significance of this means if the user chooses system clock periods > 3000ps, adding or removing small number of registers would not impact on power consumption significantly.

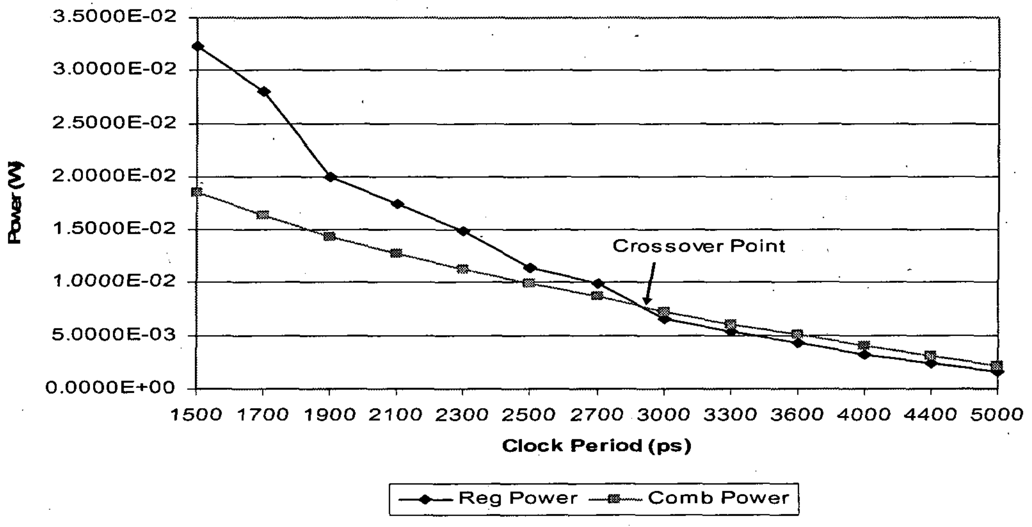


Figure 4.22 Breakdown of Total Combinational Logic Total Register Power Consumption when Φ =1 (PSI)

Therefore, PSI result provides the user with flexibility to alter the number of registers (albeit a small addition or subtraction) without compromising performance or power. Such crossover point does exist in the naive insertion but at much increased clock periods. Figure 4.23 shows the crossover point approximately at 7000ps for naive stages, which compared to PSI result, it is a much-compromised performance.

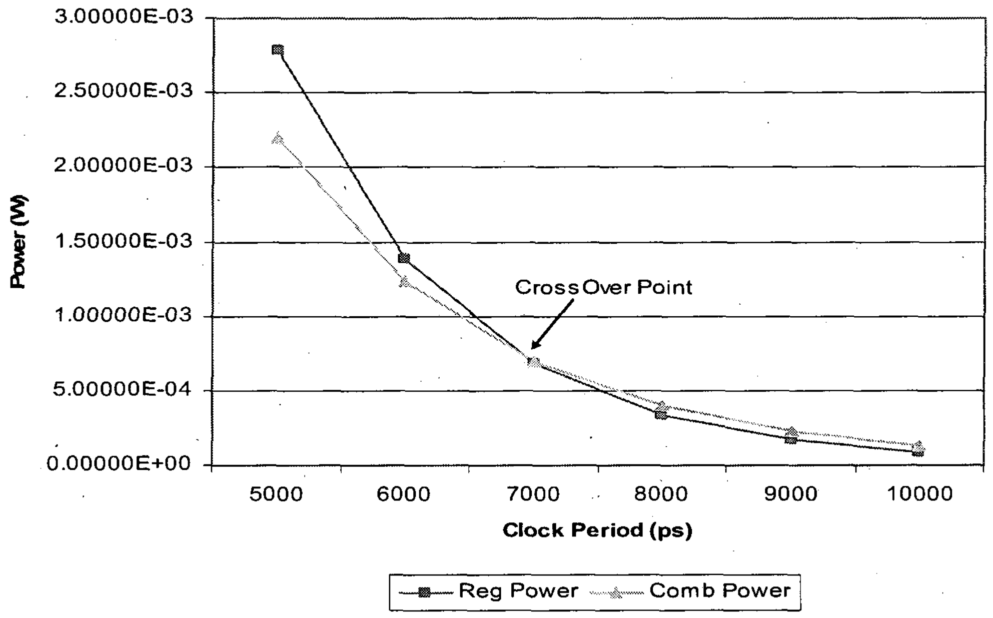


Figure 4.23 Breakdown of Total Combinational Logic and Total Register Power Consumption when Φ =0 (Naive Insertion)

In terms of actual number of stages and register count verses clock period, Figure 4.24 and Figure 4.25 respectively show for the PSI result that these two parameters decrease with increasing clock period. This result is as expected since increasing clock period allows more stages to be removed and hence less register count. Note, although for periods > 3000ps the actual register count for PSI is almost flat (Figure 4.25), power reduction is still possible due to decreased frequency.

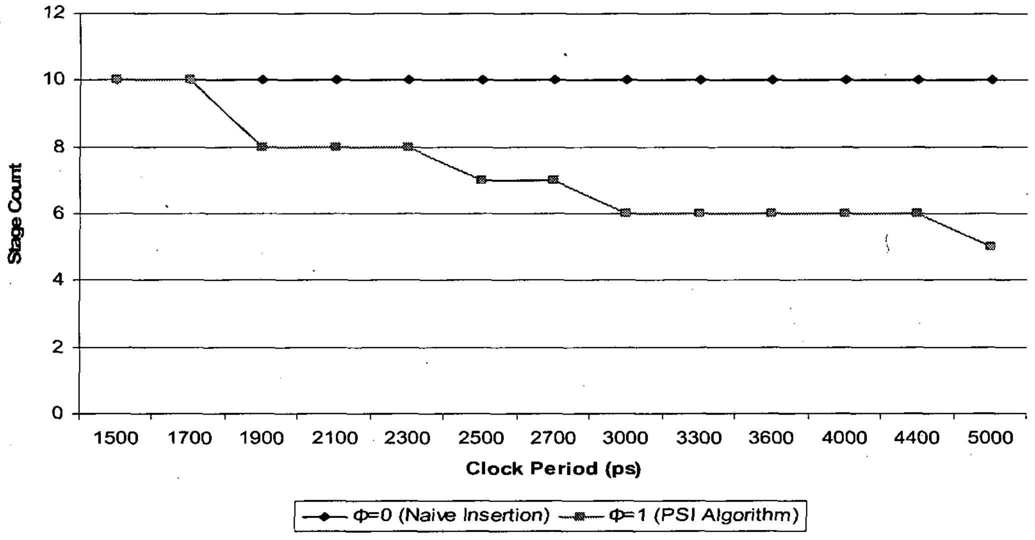


Figure 4.24 Stage Count Trends with Increasing Clock Period

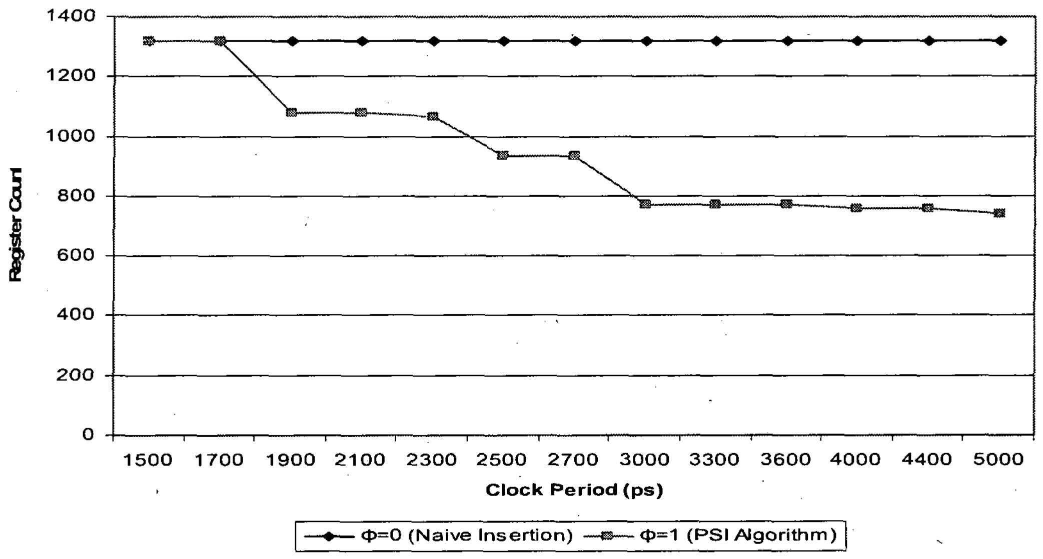


Figure 4.25 Register Count Trends at Increasing Clock Period

4.3.1.3 Energy Efficiency Analysis

In [34, 35] one of the basic energy efficiency metric used for comparing different designs is energy*delay^2. Hence, power is proportional to voltage cubed. This implies power multiplied by delay cubed provides voltage-invariant energy efficiency metric. Therefore, this metric is used for the naive and PSI results to determine the clock period that is the most energy efficient. Figure 4.26 show the optimal clock period that maximises energy*delay^2 is approximately 4400ps for both naive and PSI; however, PSI is 24.5% more energy efficient at that clock period. This metric has less emphasis on performance and more on energy efficiency and therefore performance driven applications may find Figure 4.20 more appropriate.

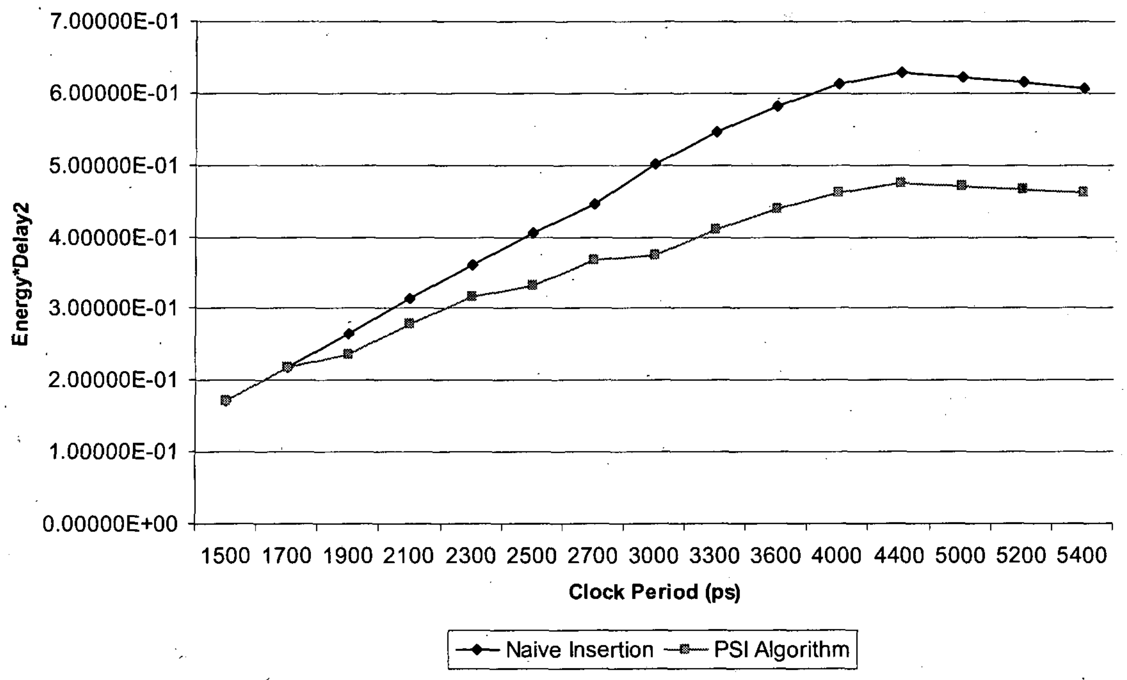


Figure 4.26 Energy Efficiency versus Clock Period

4.3.2 Effect of Φ on Power-Performance Analysis

The value of parameter Φ (delayed stage insertion variable) is important in setting the value of *W* which in turn has implication for the final results as shown in Equations 3.15 and 3.16. It was established in section 4.3.1 that when Φ=0 the final result in terms of power and performance matched that of the naive approach. However, when Φ=1 and Φ=1.4, results in section 4.3.1.1 and 4.3.2 respectively showed significant power reduction while maintaining the same performance. This section investigates the impact of different values of Φ on power and performance.

Equation 3.16, to indicate the final value for W (1 or 0), depends on the mathematical inequality $\{W' > \Phi\}$ where W' is the intermediate value of W. This intermediate value is calculated by dividing the number of outputs of element *i*+1 by element *i*. For example, if $W_i' = 1.09$ and Φ=1, then the inequality is true and hence, W_i is set to 1. This implies a stage will be inserted after element *i*, since this element has less outputs to latch. However, consider the case where a third element *i*+2 has less number of outputs than element *i* or *i*+1, under the current equation for calculating W_i', element *i*+2 outputs are not taken into account and in this example, a stage is still inserted at element *i* outputs. It is possible however to skip the insertion of stage at element i by choosing a value for $\Phi > 1$ which forces the inequality $\{W_i' > \Phi\}$ to be false and hence no stage is inserted at element *i*. This delaying of stage insertion methodology permits skipping of potential stages insertion and hence provides a way of saving on number of latches. If however, it turns out that after skipping a number of potential stages, an undesirable insertion may have to be implemented (due to timing) at an element with outputs greater than some of the previous elements. This penalty might be incurred with current version of the algorithm. For this reason, when Φ is > 1 it is not obvious from the outset if power reduction can be achieved, this depends on the architecture and whether it lends itself to having characteristics that take advantage of the delaying methodology.

Since Φ is defined as a real positive number (Equation 3.16), it can take values from 0 to $+\infty$. In reality, Φ=0 does not reveal a new set of results (known already from the previous discussion) and the upper limit can be found by dividing the largest number of element outputs in the architecture by the smallest number of element outputs, discussed in further detail in section 4.3.3.2.

4.3.2.1 Power-Performance Analysis with Φ=1.5

When PSI algorithm is executed with Φ=1.5, the final result is shown in Table 4.15 where grey boxes indicate redundant stages. Compared to Table 4.12 the first point to note is more grey boxes which are visible at the early stages. For example, stages 1, 2, 3 and 4 in Table 4.15 are shown to be redundant at certain clock periods compared with Table 4.12, which shows no redundancy in these four stages.

Table 4.14 Pipeline Stages Characteristics when Φ =1.5

Clock Period (ps)	Stage1	Stage2	Stage3	Stage4	Stage5	Stage6	Stage7	Stage8	Stage9	Stage10
1500	1	1	1	1	1	1	1	1	1	1
1700	1	1	1		1	1	1	1	0	1
1900	1	1	1	1	1	1	1	0	1	0
2100	1	1	1	1	1	1	1	0	1	0
2300	0	1	1	1	1	1	0	1	0	1
2500	0	1	1	1	1	0	1	0	1	0
2700	0	1	0	1	1	0	1	0	1	0
3000	0	1	0	1	0	1	0	0	1	0
3300	0	1	0	1	0	1	0	0	1	0
3600	0	0	1	0	1	0	0	1	0	0
4000	0	0	1	0	1	0	0	1	0	0
4400	0	0	1	0	0	1	0	0	0	1
5000	0	0	0	1	0	0	0	1	0	0

The power consumption when Φ=1.5 is less than that of naive stages and when Φ=1 as shown in Figure 4.27. For example, when the clock period is 3000ps Φ=1.5 solution consumes 33.8% less than naive solution and 11.5% less than when Φ=1.

In terms of energy-efficiency this solution has its maximum efficiency when the clock period is in the range of 4000ps-4400ps as shown in Figure 4.28. Compared to the naive stages, Φ=1.5 solution is 36.9% more energy-efficient at clock period of 4400ps.

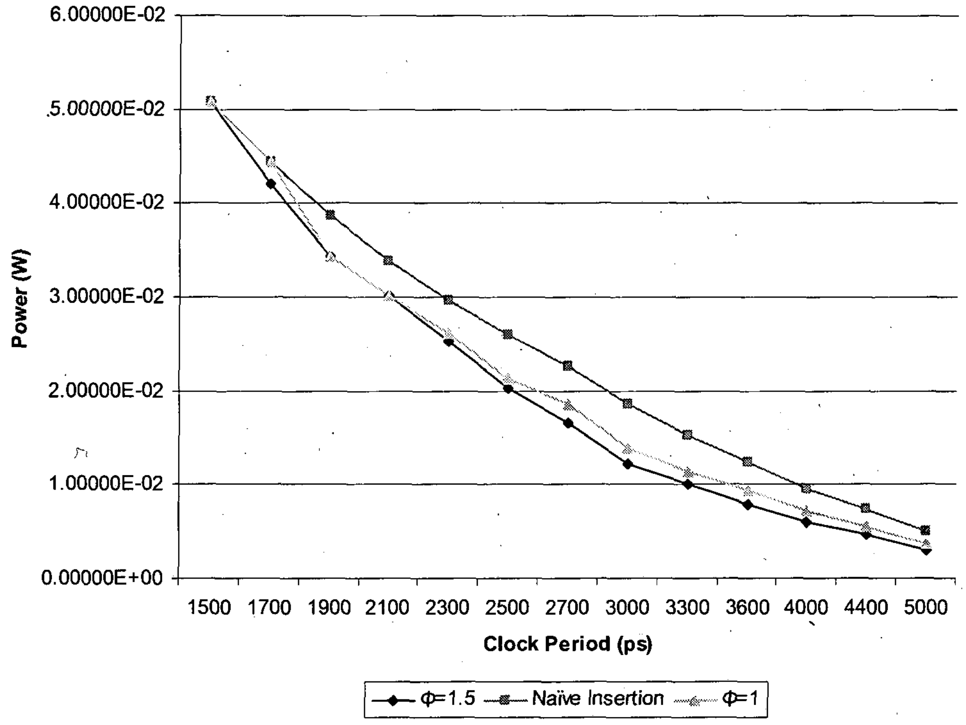


Figure 4.27 Power versus Clock Period for Φ =1.5

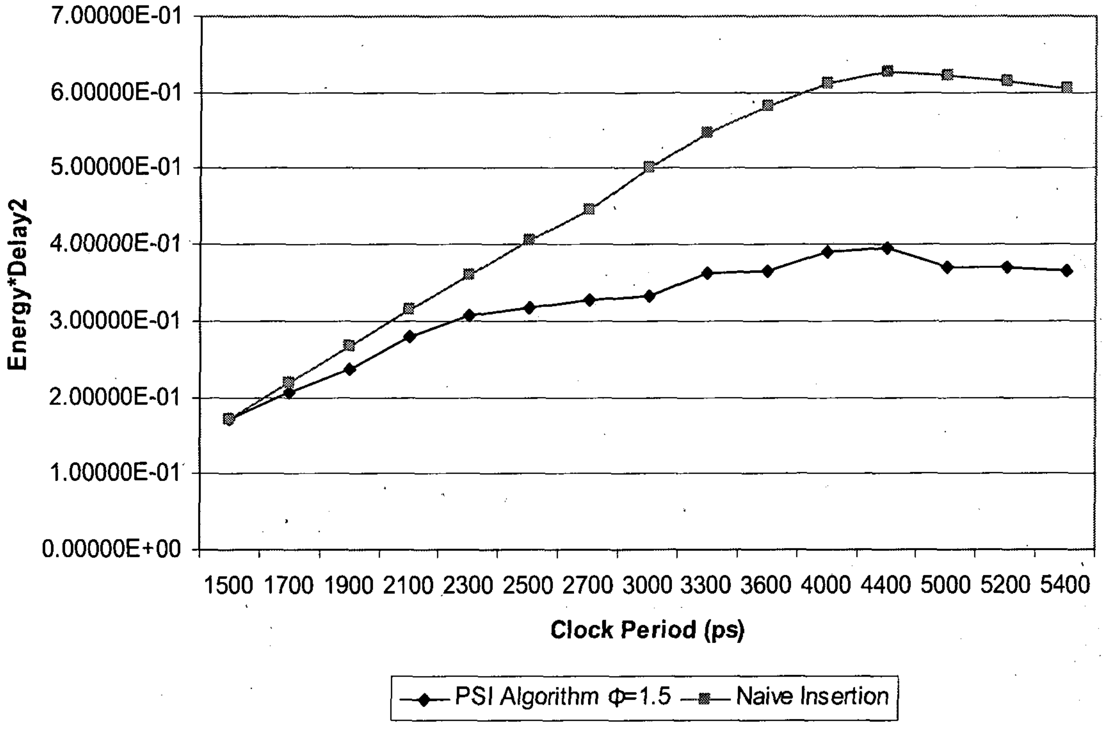


Figure 4.28 Energy-Efficiency Curve when Φ =1.5

4.3.2.2 Power-Performance Analysis with Φ ranging from 0 to 3

In this section, PSI algorithm is executed with Φ set to values ranging from 0 to 3, increasing at an interval rate of 0.1. Note, the maximum limit for Φ (chosen as 3) is based on the observation that to make the inequality $\{W > \Phi\}$ false for any stage, it is necessary to calculate the largest value W can have based on dividing the largest number of element outputs by the smallest number of outputs. For TDPFPADD and using Table 4.9, these values are 178 and 78 which result in W =2.28, and because the delaying strategy is used, Φ is set to be > 2.28, hence 3 is chosen. Setting Φ to values > 3 gives the same result as Φ=3.

Using UNIX workstation running Solaris operating system with 8G of RAM, the PSI algorithm was executed with clock period in the range of 1500ps to 5000ps. For each clock period, power consumption was estimated with Φ ranging from 0 to 3. The final results took 21 seconds to generate and the row data is presented in Appendix C.2.

Figure 4.29 depicts a range of power dissipation for each clock period depending on the value of Φ. Different values of Φ at the same clock period could generate results with different number of stages. For example, at clock period of 5000ps when Φ=1.4 and Φ=0.2, the number of stages is 2 and 10 respectively. In case of Φ=1.4 the solution consumes 41.6% less power than the solution with Φ=0.2. Note, increasing values of Φ does not necessarily generate results that consume less power. For example, at clock period of 5000ps, the solution with Φ=1.4 consumes less power than when Φ=3 as shown in Figure 4.30. Figure 4.31 shows the trend of Φ values for the least power consumption solution for each of the evaluated clock periods. It is clear from Figure 4.31 and Figure 4.30 that the trend of Φ values for a particular clock period does not match the trend across multiple clock periods. This is because the value of Φ will influence the value of W, providing a way of delaying the decision to insert a pipeline stage further down the pipeline. In doing so, opportunities might arise such that a less costly pipeline stage in terms of registers may be found further down the pipeline stages. However, sometimes it is not possible to delay this decision too deep down the pipeline because the clock period requirement would be violated. Therefore, the value of the clock period will influence the value of I (stage should be inserted or not) and this in turn will influence whether the stage is inserted or not. Table 4.15 shows for the clock period of 5000ps, a comparison of the values for *I*, *W* and *P* for two values of Φ. The first Φ value is 1.4 (representing the low power solution for the clock period 5000ps), and the second value is 0.6 (representing a high power consumption solution). In this table P = 0 means a stage is inserted. There are three interesting observations from Table 4.15. Firstly, the low power solution has two stages compared with the high power solution, which has nine stages; this represent a significant reduction in the number of stages. Secondly, looking at the value of *I* in the Φ=1.4 and Φ=0.6 columns, both indicate by the fact that *I*=0 for most stages, that the clock period is large enough to embed several stages. Thirdly, the value of *W* for Φ=1.4 is zero for most stages, which indicates that the inequality $\{W > \Phi\}$ is false for most stages. This implies that the decision to insert a stage based on the number of outputs is being delayed when Φ=1.4 compared to Φ=0.6. This delaying strategy has helped in eliminating redundant stages leading to a low power solution with only two pipeline stages.

Table 4.15 Values of *I*, *P* and *W* when Φ = 1.4 and 0.6 and the Clock Period is 5000ps

Stage No.	Φ=1.4			Φ=0.6		
	I	***P***	***W***	***I***	***P***	***W***
Stage 1	0	1	0	0	0	1
Stage 2	0	1	0	0	0	1
Stage 3	0	0	1	0	0	1
Stage 4	0	1	0	0	0	1
Stage 5	0	1	0	0	0	1
Stage 6	1	0	0	0	0	1
Stage 7	0	1	0	0	0	1
Stage 8	0	1	0	0	0	1
Stage 9	0	1	0	0	0	1
Stage 10	0	1	0	0	0	0

In terms of energy-efficiency, Φ=1.4-3 at clock period of 4400ps is the most energy efficient solutions as shown in Figure 4.32.

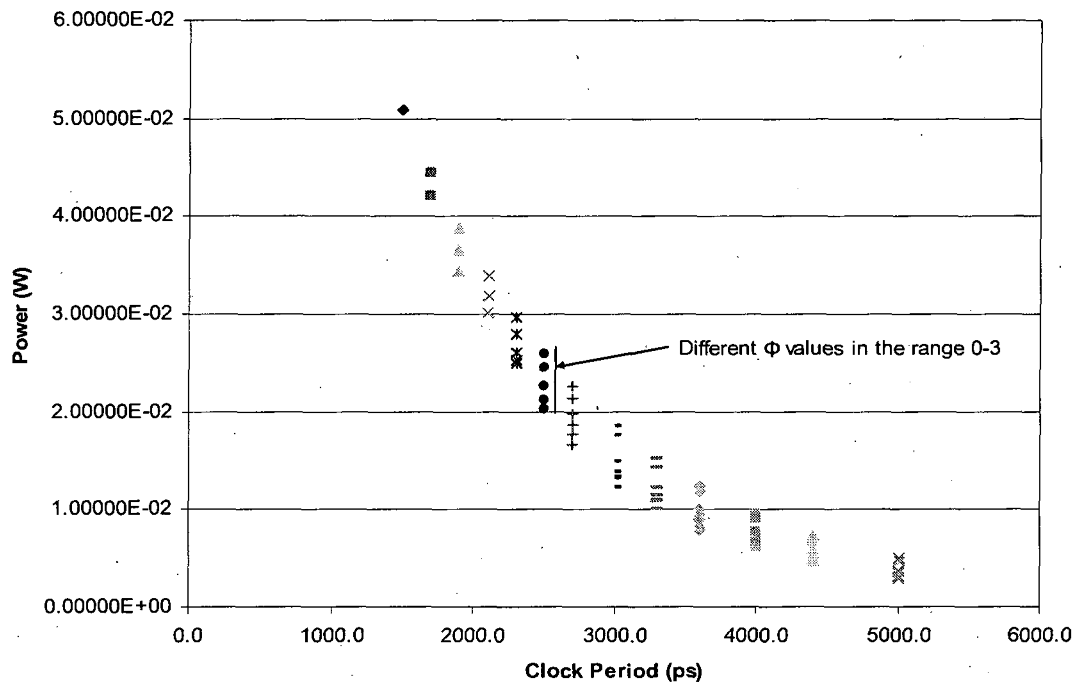


Figure 4.29 Distribution of Power Dissipation when Φ in the Range of 0-3

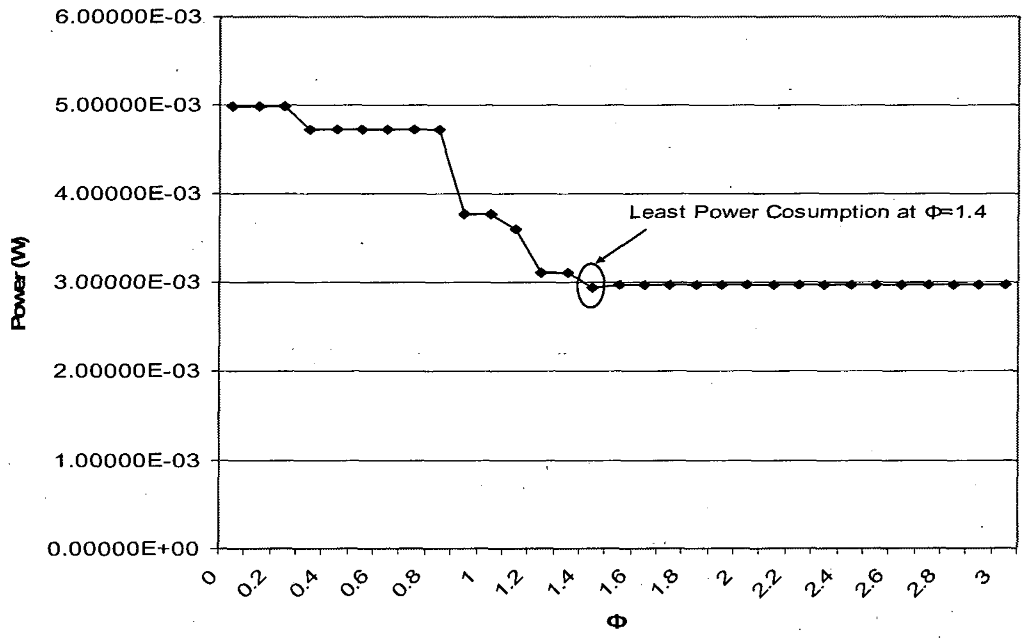


Figure 4.30 Φ =1.4 Generate Stages with Least Power Consumption at Clock period of 5000ps

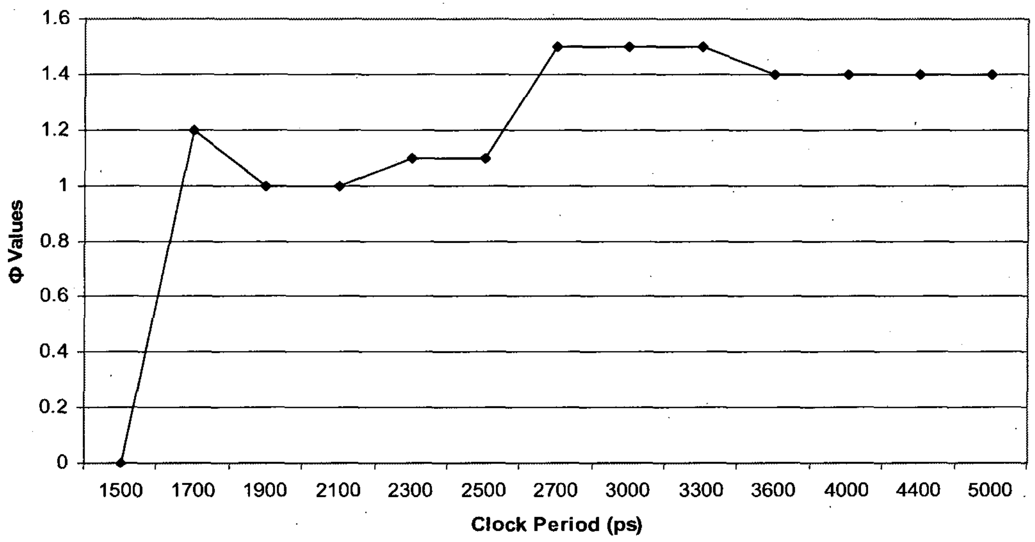


Figure 4.31 Φ Trend for the Least Power Consumption Solution

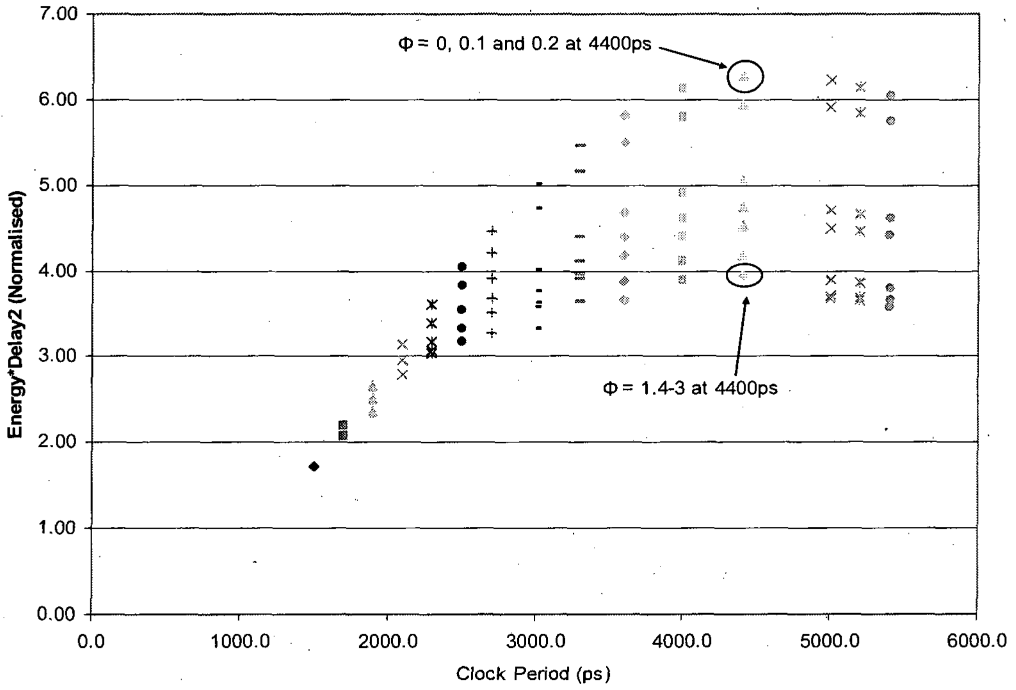


Figure 4.32 Energy-efficiency for different values of Φ

4.3.3 Limitation of PSI Algorithm

Using Φ to delay stage insertion is a powerful mechanism that allows PSI to skip potential stage insertion points and hence save on register count. It is possible however, that after a number of skips, one may end up with an element that has a larger number of outputs compared to the previously skipped outputs. Under the current PSI algorithm this expensive outputs would be latched if clock period does not permit the skipping of current element outputs. What would be better under these circumstances is to go back, check the skipped outputs and consider the element that has the least number of outputs to latch. This eliminates the need to make expensive decisions and instead uses the previously skipped outputs to reduce latch count. This improvement is proposed as part of future work.

4.3.4 Comparison with Exhaustive Search Algorithm

This section aims to compare the PSI algorithm with a general-purpose exhaustive search. The goal is to run the exhaustive search with a specific clock period constraint, identify the solution that generates the least number of registers and compare it to the PSI at the same clock period. Implementation of the exhaustive search algorithm is described in section 4.3.5.1 and comparisons of results are illustrated in section 4.3.5.2.

4.3.4.1 Exhaustive Search Algorithm Implementation

In general, Exhaustive Search (ES) algorithms tend to have a longer execution time compared with more tailored algorithms but they guarantee an optimal solution as they evaluate every possible combination in the problem, which makes them attractive from this point of view [102]. The ES is used to analyse pipeline stages of the TDPFPADD and generate results that meet user specified clock period while using the least number of latches to implement the stages. The input to the algorithm is the data array shown in Table 4.9. Since the number of stages in this architecture is 10, ES must evaluate $2^{10} = 1024$ cases representing the different combination of stages. During the evaluation, if a case violates timing, this case is discarded and the next case is evaluated. Once a case is identified as meeting timing, the case is saved to a best solution variable and if a subsequent case also passes timing but with less total number of registers it becomes the current best solution. At the end of the run, best solution variable should have a result that meets timing with the least number of registers. The ES pseudo-code is illustrated in Figure 4.33. Note line 1 to 4 initialises the variables used in the algorithm. Line 5 iterates through the 1024 cases one at time evaluating each case using the *for* loop shown in lines 8 to 22.

```
1  Set C to user specified clock period
2  Initialise array STAGES_ARRAY with data from Table 4.9
3  Initialise best_solution to 100000
4  Initialise total_delay 0
5  for {K=1, K<1024, K=K+1} {
6      convert decimal K to binary number B
7      assign each bit of B to a stage in STAGES_ARRAY
8      for {i =1, i<=10, i=i+1} {
9          if {stage{$i} exists} {
10             total_delay = 0
11           } else {
12             total_delay = total_delay + stage{$i}_delay
13           }
14         if {total_delay > } {
15               Timing is violated, exit current case16
16             } else {
17               Timing ok, Continue to next stage
18             }
19             if {i = 10} {
20                 register_count = total latched outputs from enabled stages
21             }
22     }
23     if {register_count < best_solution} {
24         best_solution = register_count
25     }
26 }
```

Figure 4.33 Pseudo-Code of ES Implementation

If during the evaluation the total delay of unlatched stages represented by *total_delay* variable becomes > required clock period as shown in line 14, current case is abandoned. When Stage10 is evaluated and timing is still meet (line 19), the total register count from all the stages that exists is assigned to the variable *register_count* (line 20) and if this is less than *best_solution* (line 23), *best_solution* is assigned the new total register count as shown in line 23.

Total delay of unregistered stages is represented by the variable *total_delay*. When a stage exist, this delay is set to 0 (line 10) otherwise it is updated with delays of unlatched stages (line 12). When *total_delay* is > required clock period the current case is abandoned because timing is violated as shown in line 15. After evaluating stage 10 and timing is still met, total number of registers for latched stages is saved in the variable *register_count* (line 20). If current case generates least number of latches compared to previous cases, this case is now the best solution (lines 23 to 25). ES was implemented using TCL programming language [89].

4.3.4.2 Experimental Results

ES algorithm was executed on TDPFPADD architecture using clock period ranging from 1500ps to 5000ps. The result is then compared to that of PSI algorithm for the same clock periods. Figure 4.34 shows the total power consumption of each algorithm along with the naive approach, which represents no stage optimisation. From this figure, it is clear that for this design the PSI results are closely correlated to that of ES. There is a slight improvement in power consumption using ES at 4000ps however; it is so small it is practically insignificant.

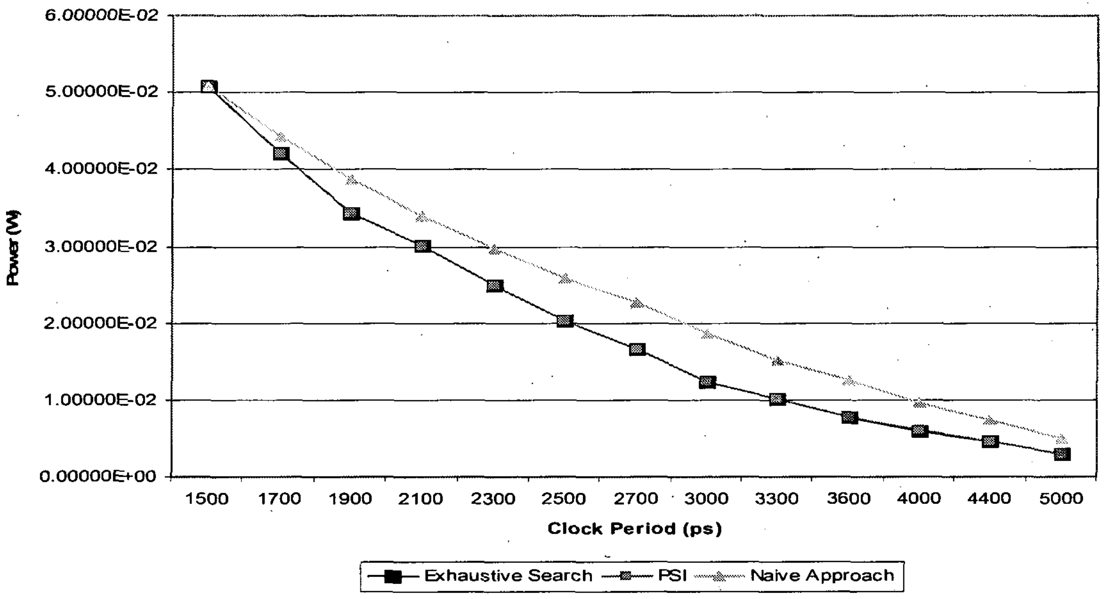


Figure 4.34 Power Consumption Estimate for ES and PSI algorithms

In terms of register count and number of stages, Figure 4.35 and 4.36 respectively illustrate the fact that register count is practically identically between the two algorithms with a small reduction in register count at 4000ps, hence the small improvement in power at that frequency. This reduction would have been significant at higher clock frequencies because register power consumption is dominant at those frequencies, compared with low frequencies where combinational logic power is more dominant, as was highlighted in section 4.3.1.2. In terms of number of stages, the two algorithms generate the same number of stages for all clock periods as shown in Figure 4.36.

What is significant though is that ES generated the results in 13 seconds compared with the PSI, which took only 9 seconds. This improvement in run time may seem small but as will be shown in chapter 5, when the number of stages increases, ES run time becomes prohibitive.

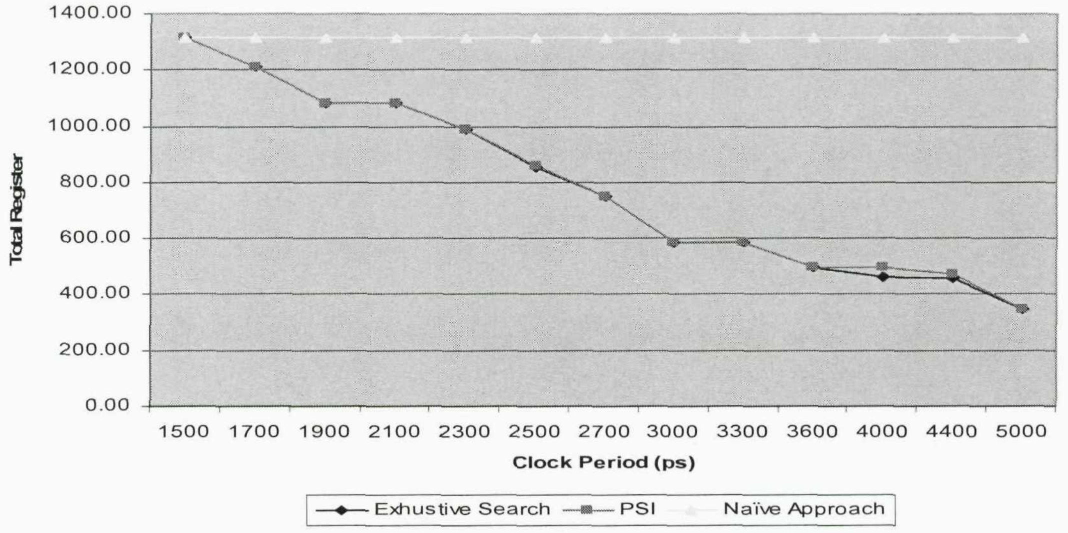


Figure 4.35 Total Registers for ES and PSI

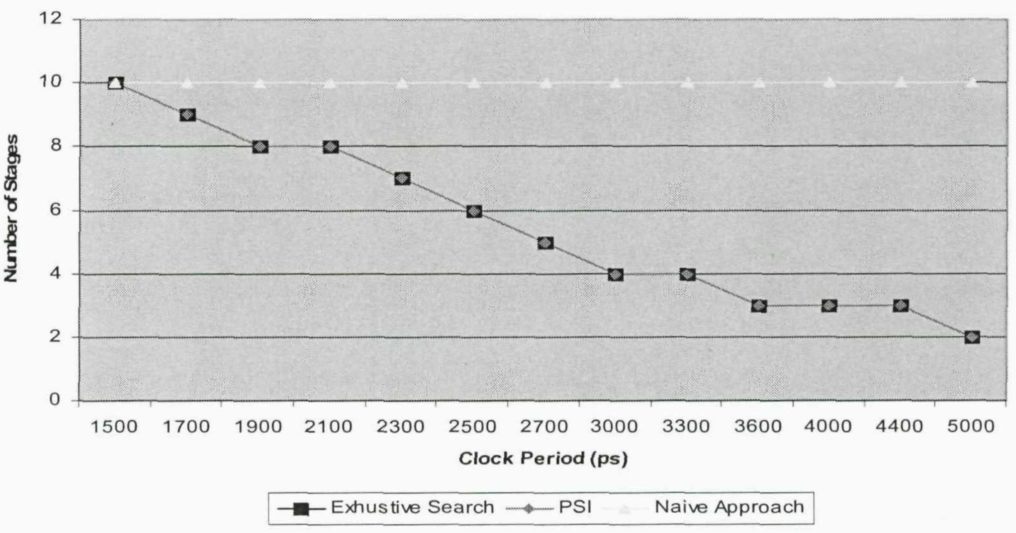
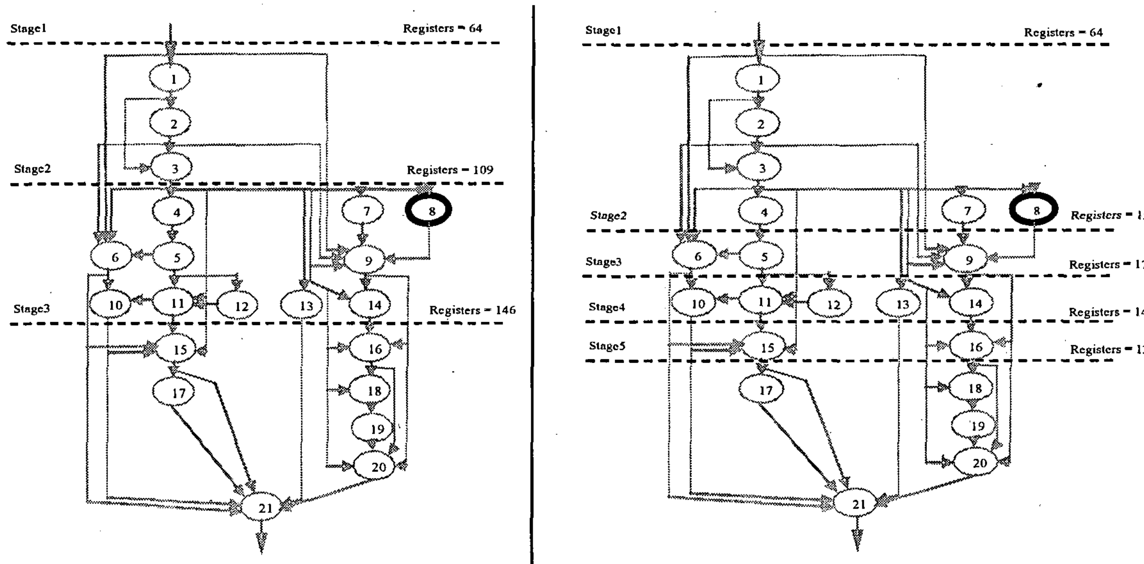


Figure 4.36 Total Stages for ES and PSI

4.3.5 Analysis of the Impact of Pipelining on Interconnect Capacitance and Dynamic Power Consumption

Wilton *et al* [149], showed that for a maximally pipelined design targeting 130nm FPGA, the interconnect capacitance between the stages was reduced and this reduction contributed to a significant lowering of dynamic power. For example, experiments with 64-bit unsigned multiplier showed that the difference in dynamic power between the most pipelined variant and the least pipelined variant is 81%. The paper by Keane and Woods [153] also highlighted that the layout of the circuit can have significant impact on interconnect lengths. It was shown that preserving the regular structure of a carry-save multiplier had kept the interconnect lengths to minimum and this in turn lowered the dynamic power consumption by 10.4% compared with a flat (no regularity preserved) carry-save multiplier. The experiments carried out in [153] used a 350nm standard cell CMOS technology and an operating speed of up to 80MHz.

This section investigates the impact of increasing pipeline stages for the TDPFPADD. Two TDPFPADD designs are implemented using 90nm standard cell library and operating at 200MHz frequency: 3-pipeline stages and 5-pipeline stages. These two pipeline implementations were solutions from the PSI method using input clock period of 200MHz and Φ values between 0 and 3, as shown in Figure 4.37. To study the impact of regular cell layout on interconnect lengths, two physical layouts are investigated for each of the pipelined design.

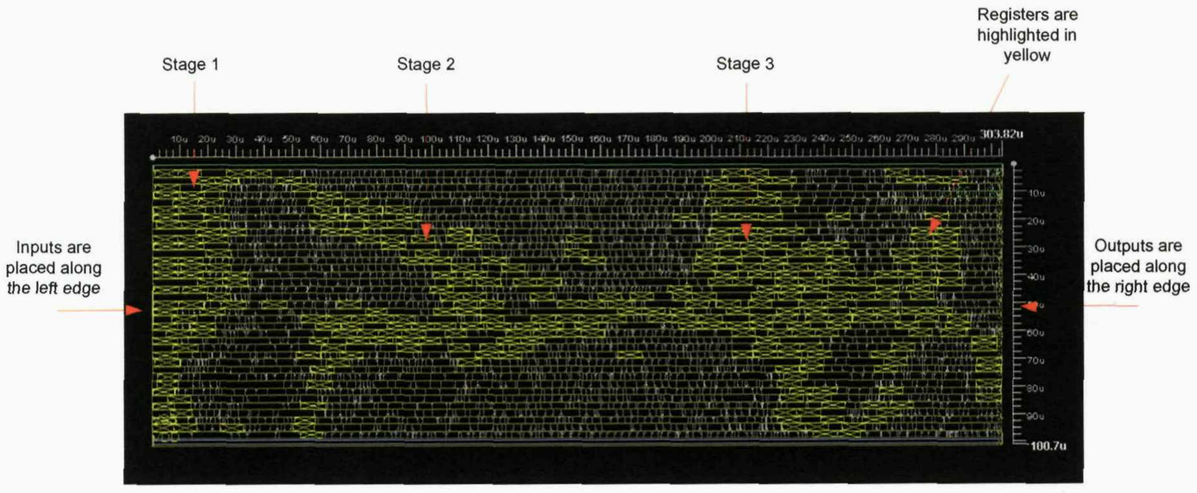


a) 3-Stages Solution with total of 349 registers

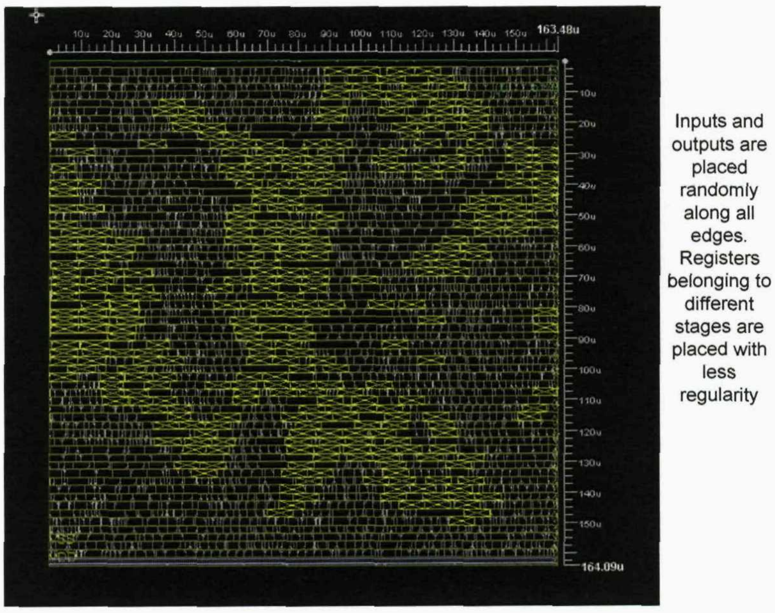
b) 5-Stages Solution with total of 676 registers

Figure 4.37 Two Solutions from the PSI Method when the Clock Period is 200MHz

The first physical layout will be in the shape of a square with input/output pins placement done randomly around the periphery of the floorplan by the layout tool Magma BlastFusion [33]. The second physical layout is the shape of a rectangle with input pins placed manually on the left hand side edge and the output pins are on the right hand side edge of the rectangle. Figure 4.38 and 4.39 depict the two physical layouts for the 3-stages and 5-stages pipelines.


Registers are highlighted in yellow
Stage 1
Stage 2
Stage 3
Inputs are placed along the left edge
Outputs are placed along the right edge
303.82u
100.7u

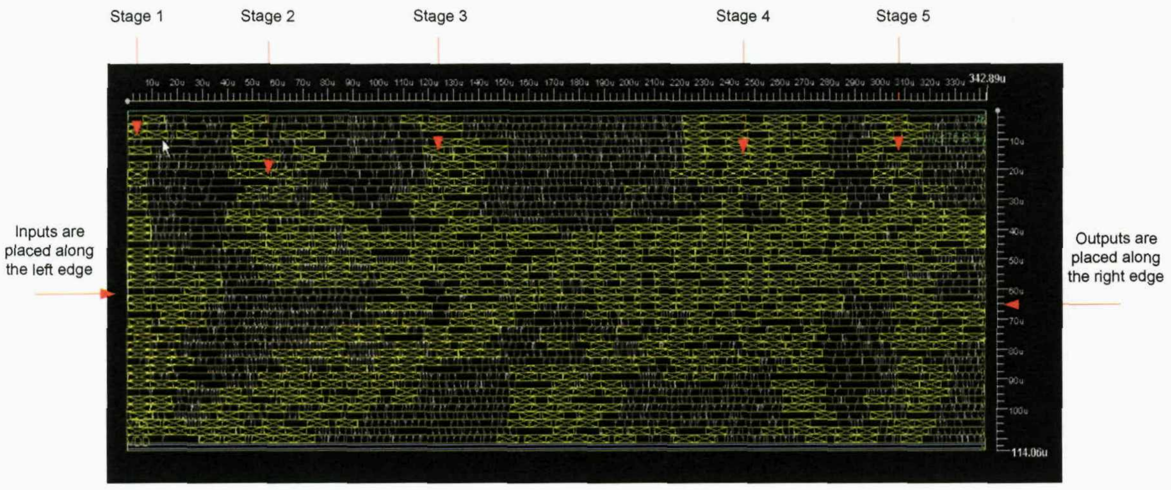
a) 3-stages pipeline rectangular layout


163.48u
164.09u
Inputs and outputs are placed randomly along all edges. Registers belonging to different stages are placed with less regularity

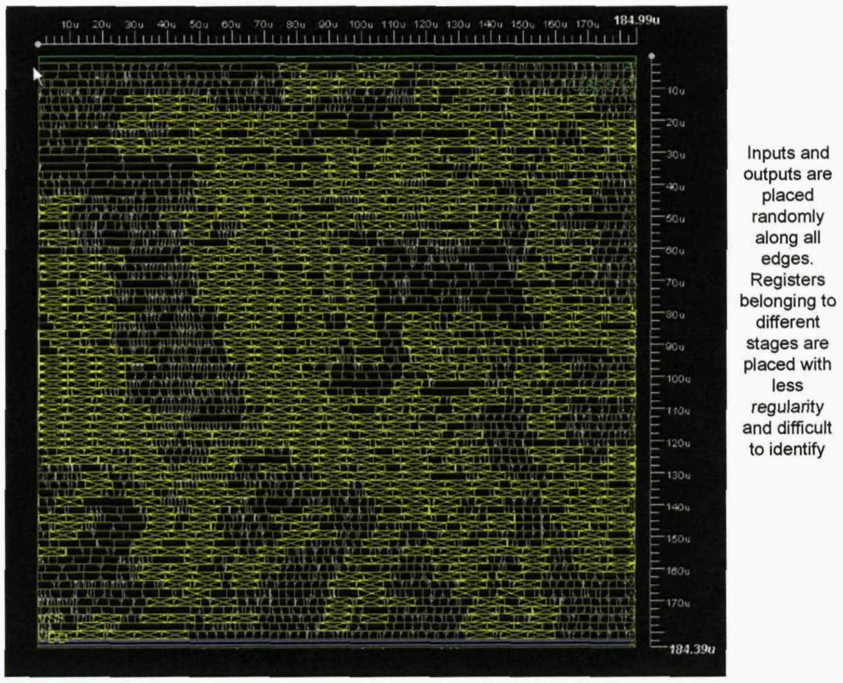
b) 3-stages pipeline square layout

Figure 4.38 3-Stages TDPFPADD Pipeline with Two Physical Layouts

The square physical layout represents a flat layout while the rectangular shape promotes structures that are more regular. For the rectangular shape, Magma BlastFusion tool is able to place the standard cells in a sensible pattern helped by the natural shape of a pipeline and the location of pins placement. Figure 4.38 highlights the register placement and as can be seen, they are more regular and resemble a pipeline for the 3-stages rectangular layouts than for the square layout.


Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Inputs are placed along the left edge
Outputs are placed along the right edge
342.89u
114.06u

a) 5-stages pipeline rectangular layout


184.99u
184.39u
Inputs and outputs are placed randomly along all edges. Registers belonging to different stages are placed with less regularity and difficult to identify

b) 5-stages pipeline square layout

Figure 4.39 5-Stages TDPFPADD Pipeline with Two Physical Layouts

In this experiment we also consider the clock tree power consumption and therefore a full clock tree is done for each of the pipelined designs using the clock tree synthesis functions available in Magma BlastFusion. Synopsys Primepower is used to estimate accurately the power consumption of the gate level representation of the designs with toggle activity obtained from gate-level simulation of 2000 semi-random input stimulus. The gate level netlist is extracted from the physical layouts database and each of the pipelined designs has two netlists, one for the rectangular layout and the other for the square layout. Table 4.16 shows the results for both 3-stages and 5-stages TDPFPADD designs and for the different layout shapes. In the first column of Table 4.16, the different comparison matrices are listed. *Switching power* refers to the power consumed through charging and discharging of all the interconnect capacitance (i.e. the capacitive loads seen by the driving cells).

Table 4.16 Results for the 3-stages and 5-stages TDPFPADD

Matrices	**3 stages square**	**3 stages rectangle**	**5 stages square**	**5 stages rectangle**
Total cells Area (mm2)	0.021	0.020	0.029	0.030
Switching power (mw)	1.166	1.214	1.458	1.493
Internal Power (mw)	2.421	2.280	3.536	3.534
Leakage Power (uw)	0.127	0.122	0.180	0.183
Total Power (mw)	3.588	3.494	4.995	5.027
Clock tree Power (mw)	0.763	0.663	1.212	1.104
Register Power (mw)	1.269	1.274	2.153	2.166
Total Nets	2522	2442	2986	2980
Total net capacitance (fF) (excluding clock and reset)	24066	25021	26549	28458
Average net capacitance (fF) (excluding clock and reset)	9.54	10.2	8.89	9.55
Clock tree capacitance (fF)	1753	1675	3137	2785
Reset line capacitance (fF)	1131	1135	2299	2484
No. of Registers	349	349	676	676
Total cells	2496	2410	2998	3035

Internal power is the sum of the short circuit power and internal switching power (charging and discharging of the driving cells internal capacitance), consumed by all the logic cells. *Leakage power* is the total power dissipation of all logic cells when their inputs are static. *Total power* is the sum of *switching power*, *internal power* and the *leakage power*. *Clock tree power* is the total power consumption of the clock tree, which includes the power consumption of the clock buffers and the switching power of the clock tree interconnects. *Register power* is the total power consumption (leakage power plus internal switching power) in registers only. Note, the *clock tree power* and *register power* have already been included in the *total power* metric shown in Table 4.16; however, they have been added here as separate matrices to aid the analysis of the impact of register and clock tree power consumption for the different pipeline stages. *Total nets* is the sum of all interconnects in the design excluding clock and reset nets. *Total net capacitance* is the sum of all the interconnect capacitance in the design excluding clock and reset nets. *Average net capacitance* is calculated by dividing the *total net capacitance* by *total nets*. The reason for not including the clock and reset nets as part of the three net matrices described earlier is because clock and reset nets connect to all the registers in the design. These nets have significant capacitance associated with them and therefore isolating the clock and reset nets helps to identify their contribution to the overall net capacitance in the design. The details of the remaining matrices are clear and require no further elaboration.

From Table 4.16, comparing the two layouts for the 3-stages pipeline implementation, it is clear that the *total power* consumption of the 3-stages rectangular shape consumes 2.6% less power than the 3-stages square layout. This reduction in power consumption is small, and looking closely at the *switching power* and *internal power* figures, the rectangular layout consumes more switching power but has less internal power compared with the square layout. The increase in *switching power* for the 3-stages rectangular layout can be explained by the fact that the *total net capacitance* has increased from 24066fF to 25021fF. The increase in *total net capacitance* for the 3-stages rectangle could be because the shape of the layout is thinner and longer and therefore nets could be covering a longer distance than the square shape. The reduction seen in internal power for the 3-stages rectangular shape is because in this layout the *total cells* were 2410 compared with 2496 for the square layout.

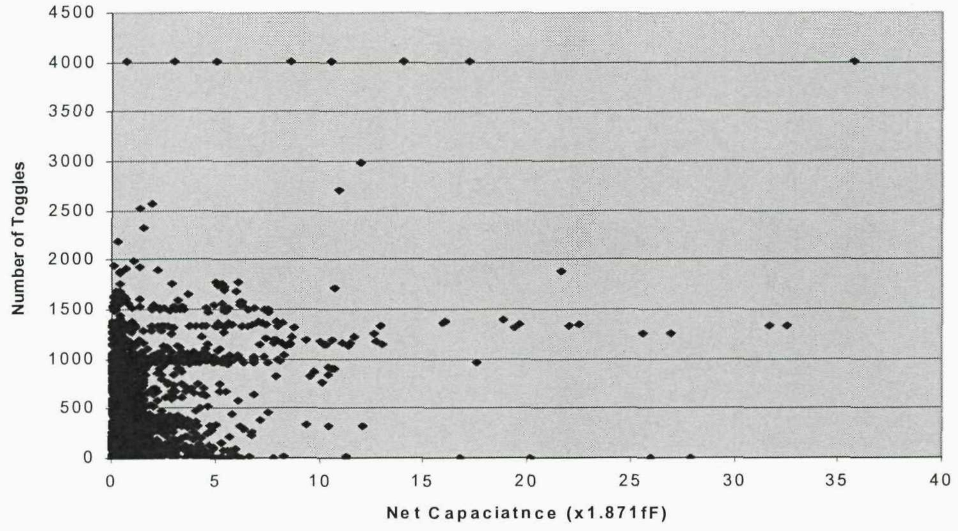
Reduction in the number of cells is a good indication that the layout tool is able to exploit the regularity of the design structure by implementing the nets on metal tracks, that avoids excessive net jogging (frequent changes in metal layers and switching between horizontal and vertical directions). Net jogging can be costly in terms of buffers required to adequately maintain the timing of the net and generally is an indication that the design is congested and metal track resources are difficult to find. This does not appear to be the case for the 3-stages rectangular layout and is reflected in the smaller number of *total cells* and therefore reduced *internal power*.

For the 5-stages rectangular layout, the *total power* is only 0.6% higher than the 5-stages square layout. This increase is insignificant and practically identical. However, it is interesting to establish why the 5-stages rectangular layout has not contributed to a decrease in the *total power* as seen with the 3-stages rectangular layout. This could be explained by the fact that the 5-stages design, in general has more cells to place and this may have resulted in routing resources being scarce and difficult to find. Note, the increase in the number of cells for the 5-stages rectangular layout compared to the square layout is largely due to buffering of nets that are now travelling a longer distance than in the square layout. This is confirmed by the increase in the *average net capacitance* from 8.89fF for the square layout to 9.55fF for the rectangular layout.

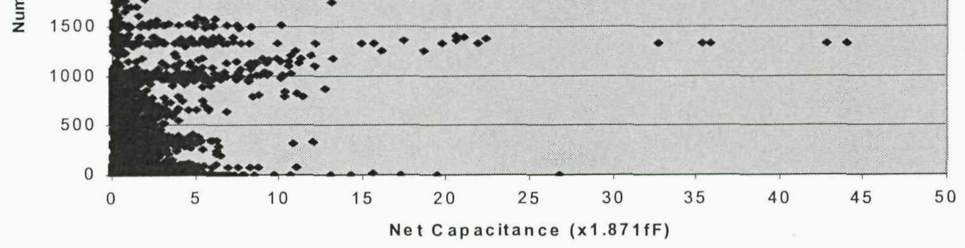
Overall, although the rectangular layout for the 3-stages implementation has reduced the total power consumption slightly, the *switching power* due to net capacitance has not decreased. For the 5-stages implementation, both layouts generated practically identical power results with the rectangular layout having an area increase of 3.5%. These results lead to a significant conclusion, that is for the 90nm CMOS technology library and using commercial layout tool, the difference in power consumption between the regular layout (represented by the rectangular shape) and the flat layout (represented by the square shape) is small. This implies that the layout shape does not influence the interconnect capacitance significantly and therefore interconnect switching power consumption is not altered significantly. Another important finding is the large contribution of *internal power* consumption to the *total power*. For example, with the 3-stages rectangular and square layout, the *internal power* consumption was 65.3% and 67.5% respectively of the *total power* consumption. The *internal power* consumption for the 5-stages rectangular and square layout was 70% and 71% respectively of the *total power* consumption. The percentages for the internal power consumption are large, and therefore minimising the number of cells will help to decrease overall power consumption.

The interesting comparison to be made from Table 4.16 is between the 3-stages and the 5-stages implementation of the TDPFPADD. First, consider the figures for the rectangular layout of both designs. Considering the *total cells area*, it is noted from Table 4.16 that the 5-stages design is 0.030mm2 compared with 0.020mm2 for the 3-stages rectangular implementation, which is an increase in area of 50%. The interconnect *switching power* has increased by 23% from 1.214mW to 1.493mW whereas the *internal power* has more than doubled to 55% for the 5-stages rectangular implementation. The *total power* consumption for the 5-stages implementation compared to the 3-stages implementation has increased by 44% from 3.494mW to 5.027mW. Furthermore, the *clock tree power* and *register power* has increased by 67% and 70% respectively for the 5-stages implementation. These results show clearly that increasing the pipeline depth has an adverse effect on overall power consumption. The main increase in power consumption comes from the increase in *internal power* with *register power* being a large contributor. This investigation shows that *clock tree power* is also an important contributor to power consumption and having a larger number of registers means the clock tree will be bigger with more clock buffers and therefore more power consumption. The latter point has not been analytically investigated in the previous work [149], and the result shown here indicates the importance of understanding the contribution of the *clock tree power* to overall power consumption. Table 4.16 shows that the 5-stages rectangular layout had a *leakage power* increase of 50%, which represents a significant increase in leakage power leading to shorter standby time. Leakage power is set to get worse as technology scaling continues (highlighted in Figure 1.7), and reduction in the number of cells including reduction in total registers should help in keeping leakage power under control.

As already highlighted in the previous paragraph, the *switching power* for the 5-stages implementation has increased by 23% compared with the 3-stages, which suggests that the increase in the number of stages has not led to a decrease in interconnect length and therefore interconnect capacitance. To investigate this point, Figure 4.40 shows the number of toggles verses interconnect capacitance for both 3-stages and 5-stages rectangular layout.



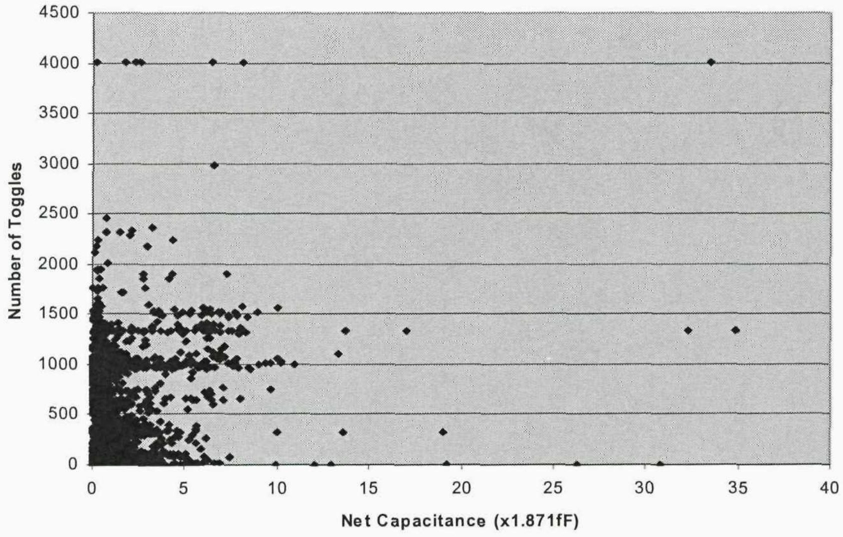
(a) 3-stages rectangular layout



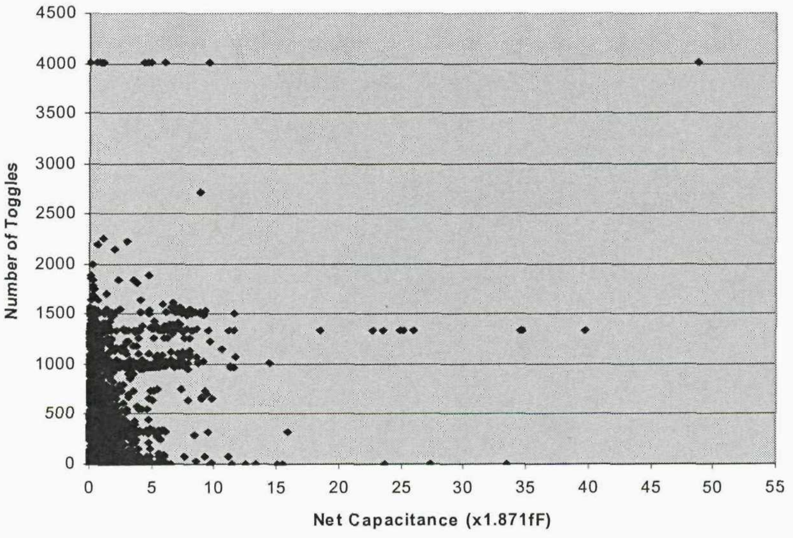
(b) 5-stages rectangular layout

Figure 4.40 Toggle Count versus Net Capacitance for the 3 and 5-Stages Rectangular Layout

In Figure 4.40, the interconnect capacitance was chosen instead of interconnect length [153], because in a multi-metal layer routing, a net could travel a long distance in a higher metal layer yet have less capacitance than a shorter net on a lower metal layer. This is because the higher the metal layer the bigger the gap between it and the substrate, and therefore net capacitance is reduced. In addition, generally higher metal layers tend to have less wire-to-wire capacitance due to less routing congestion.



(a) 3-stages square layout



(b) 5-stages square layout

Figure 4.41 Toggle Count versus Net Capacitance for the 3 and 5-Stages Square Layout

In Figure 4.40, the difference between the two graphs is very small and both designs seem to cluster the capacitance of nets between the ranges 0 to 5; the number of toggles is also contained within the range of 0 to 2000. The conclusion from Figure 4.40 is that for 90nm CMOS technology library, increasing the number of pipeline stages does not alter significantly the distributions of capacitance among the signal nets. Similar results are also seen for the square layout for both 3-stages and 5-stages implementation as shown in Figure 4.41.

Comparing the 3-stages and 5-stages square layout, the result in Table 4.16 shows slightly less increase for the different matrices to that seen for the rectangular layout. For example, the 5-stages square layout had a *total cells area* increase of 38% and *switching power* increase of 25% compared to the 3-stages square layout. *Internal power* and *leakage power* increased by 46% and 42% respectively while *total po*wer increased by 39%. The *clock tree power* for the 5-stages square layout increased by 59% and register power increased by 70%. These percentage increases confirm the same trend seen already with the rectangular layout comparison; that is, increased pipeline stages will impact on the different components of power consumption.

In summary, the overall results presented in this section have shown that for 90nm CMOS technology the regular layout (square layout) of the design does not significantly impact on interconnect capacitance. In fact, the results from Table 4.16 shows that the average net capacitance was less for the flat layout (square layout). This highlights the fact that excessive effort invested in the physical layout may not result in reduced interconnect capacitance. The results also showed that internal power was more significant than switching power and therefore reducing the number of cells and in particular registers would contribute to a larger reduction in overall power consumption. Leakage power, clock tree power and register power have all been shown to increase significantly as the number of stages increases. It is worth mentioning here that Shah *et al* [66] proposed a similar 5-stages pipeline TDPFPADD to that shown in Figure 4.37 and as far as we can establish the author inserted the 5 pipeline stages without considering the different outputs of FEs. The results in this section have indirectly compared Shah's implementation to the PSI method and as it can be seen the PSI 3-stages solution will consume between 39-44% (depending on the layout) less dynamic power than the 5-stages solution offered by Shah while still meeting the clock period requirement.

4.4 Concluding Remarks

This chapter verified the PSI methodology on TDPFPADD design. The results show that PSI can significantly reduce power consumption while maintaining the same performance. For example, compared to the naive stages, the PSI stages consumed 18.1% less dynamic power and were 24.5% more energy-efficient. Compared to Shah's 5-stages [66], up to 44% power reduction was observed. The new methodology is capable of comparing different pipeline stages in an efficient way and can present a set of solutions from which designers can choose the one that fulfils their requirements. It is shown how the value of Φ can affect the final solution and provided a way of limiting the range of values to be used.

A comparison between PSI and exhaustive search demonstrated that PSI can generate results almost as good as the exhaustive search but with run time advantage. The results correlate very well across a large range of clock periods and demonstrated that stage count and register count are almost identical.

The final experiment in this chapter has shown that for 90nm CMOS standard technology library different physical layout had limited impact on interconnect length and therefore interconnect capacitance. The experiment also showed that adding more pipeline stages does not reduce interconnect switching power and more importantly internal power is made worse and the latter component of power has significant impact on overall power consumption. Clock tree power analysis showed that increasing the pipeline stages would increase the number of clock tree buffers and this would lead to the clock tree consuming more interconnect switching power and internal power (consumed within the clock buffers). These results are significant and highlight the care that needs to be taken when adding a large number of pipeline stages. Leakage power is set to get worse as technology continues to scale beyond 100nm and every effort is needed to keep this component of power consumption under control. Reducing the number of cells in the design including reduction in total registers should help in reducing leakage power and extending standby time.

In summary, the PSI algorithm with its fast execution time has demonstrated its capability at presenting a range of solutions with differing merits. In addition, it has been demonstrated that going for the easy approach of naively inserting stages could lead to expensive penalties in terms of power consumption. On the other hand using computationally expensive algorithms such as the exhaustive search may not yield significantly better results than PSI. Maximally inserting pipeline stages while operating at the original clock period could lead to significant overall dynamic power consumption with an increase in interconnect switching power.

Chapter 5

Case Study 2: MPEG-1 Motion Compensation

This chapter aims to further validate the PSI method introduced in chapter 3 by applying it to another challenging design: MPEG-1 motion compensation. The motion compensation module has numerous implementation complexities such as a large number of arithmetic units, and many logical operations, and it consumes significant power [93]. The motion compensation module aims to compress the video frames to decrease the requirement for large data storage media. Improving the video compression is achieved by pipelining the motion compensation module leading to a higher throughput, and therefore better compression rate. Since many of today's handheld devices and in particular mobile phones employ digital video technology, the motion compensation module power consumption needs to be reduced in order to be within the allocated power budget for these devices. To reduce power consumption in a pipelined motion compensation module, a thorough analysis using the PSI method is needed.

The remainder of this chapter is organised as follows. Section 5.1 outlines the background to the MPEG-1 standard. A general description of video compression is given in section 5.2. Section 5.3 defines the motion compensation and section 5.4 describes the motion compensation implementation. Section 5.5 applies the PSI method to the motion compensation and discusses the experimental results. Run time complexity analysis is outlined in section 5.6. Finally, section 5.7 concludes this case study.

5.1 MPEG-1 Background

The Moving Picture Experts Group (MPEG) is the group in charge of developing standards for coded representation of moving pictures, audio and their combination. They are under the umbrella of the International Organisation for Standardisation (ISO) and the International Electrotechnical Commission (IEC) [94]. The goal of the group is to define a standard for coding moving pictures and the associated audio for digital storage media at approximately 1.5 Mbits/s so that a movie can be compressed and stored in a CD-ROM. The result is the MPEG-1 (Moving Picture Experts Group – Phase 1) standard, which consists of five parts: Systems (11172-1), Video (11172-2), Audio (11172-3), Conference Testing (11172-4) and Software Simulation (11172-5). The video standard is the main focus of this case study. The motion compensation offered by the video standard is implemented in both encoders and decoders.

5.2 Video Compression

Compression is the process of compacting data into a smaller number of bits [95]. A moving video image is made of a sequence of frames sampled at regular intervals to produce a moving video signal. Video compression or video coding is the process of compacting or condensing a digital video sequence into a smaller number of bits. Raw or uncompressed digital video typically requires a large bit rate (approximately 216 Mbits for 1 second of uncompressed TV-quality video) [95].

A video sequence has both temporal and spatial characteristics as shown in Figure 5.1. The temporal characteristic is defined as a series of still frames sampled at regular intervals. The spatial aspect of a scene is the information captured by the sampling points (pixels) themselves.

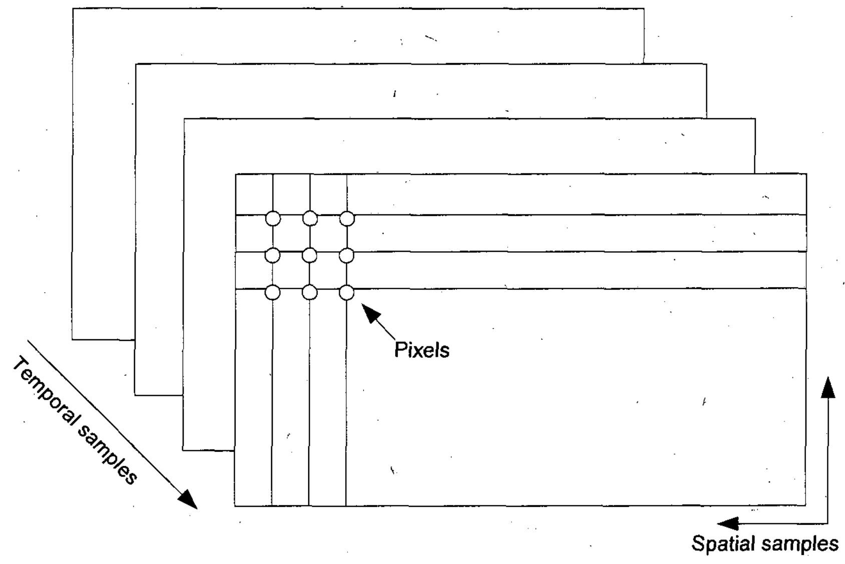


Figure 5.1 Video Sequence Characteristics [95]

Applying compression to temporal and spatial samples implies removing redundant components in these samples while maintaining enough information for faithful reproduction of the scene. In the temporal domain, there is usually a high correlation (similarity) between video frames that were captured at around the same time. Successive frames in time are highly correlated especially if the temporal sampling is high. In the spatial domain, there is usually a high correlation between pixels that are close to each other, i.e. the values of neighbouring samples are often very similar. Compression involves a complementary pair of systems, a compressor (encoder) and a de-compressor (decoder). The encoder converts the source data into compressed form (occupying a reduced number of bits) prior to transmission or storage and the decoder converts the compressed form back into representation of the original video data. The encoder/decoder pair is often described as CODEC (enCOder/DECoder) as shown in Figure 5.2.

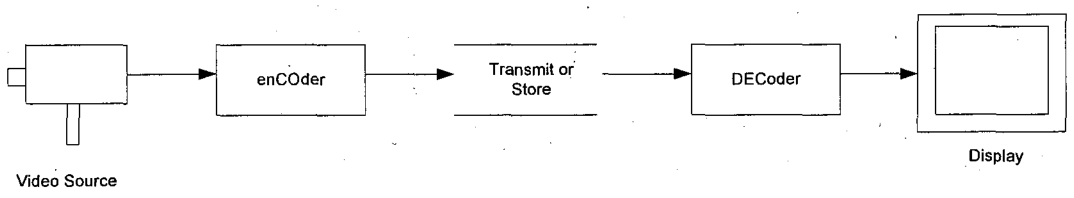


Figure 5.2 Encoder/Decoder [95]

Spatial compression is normally done using Discrete Cosine Transform (DCT). On its own, DCT offers limited compression performance, and to achieve significant scope in improving compression, motion compensation operating at the temporal domain is required. This is because motion compensation reduces the data required to transmit or store a video scene by comparing current frame with previous or future frames.

5.3 Motion Compensation

Motion compensation is the whole process of motion estimation and motion vector generation. Motion estimation creates a model of the current frame based on available data in one or more previously encoded frames known as the reference frames. These reference frames may be past frames (i.e. earlier than the current frame in temporal order) or future frames (i.e. later in temporal order). The design goals for a motion estimation algorithm are to model the current frame as accurately as possible (since this gives better compression) whilst maintaining acceptable computational complexity [97]. Motion compensation is normally done on a block of 16x16 pixel region known as a macroblock. The macroblock is used in many popular video coding standards (MPEG-1, MPEG-2 and MPEG-4). Motion estimation involves two operations. Firstly, a macroblock in the current frame is selected and a search is done in the reference frame (past or future frames) to find the best match. The best match is the one that minimises the energy of the difference between the current 16x16 macroblock and the match 16x16 macroblock. The area in which the search is carried out may be centred on the position of the current macroblock because (a) there is likely to be a good match in the immediate area of the current macroblock due to high similarity between frames and (b) it would be computationally intensive to search the whole of the reference frame. Figure 5.3 illustrate this first operation. The second operation involves subtracting the chosen candidate macroblock in the reference frame from the macroblock in the current frame to produce a residual block. The residual block is then encoded and transmitted along with the offset between the current macroblock and the position of candidate region known as the motion vector.

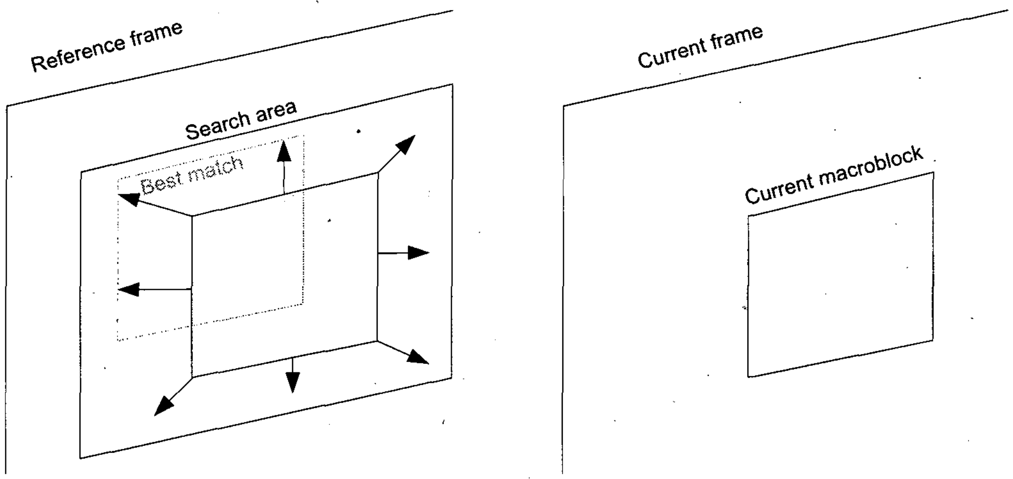


Figure 5.3 Motion Estimation [95]

The decoder then decodes the residual block and motion vector and adds to the residual block the candidate macroblock pointed to it by the motion vector. Figure 5.4 and Figure 5.5 shows an MPEG-1 encoder and decoder respectively.

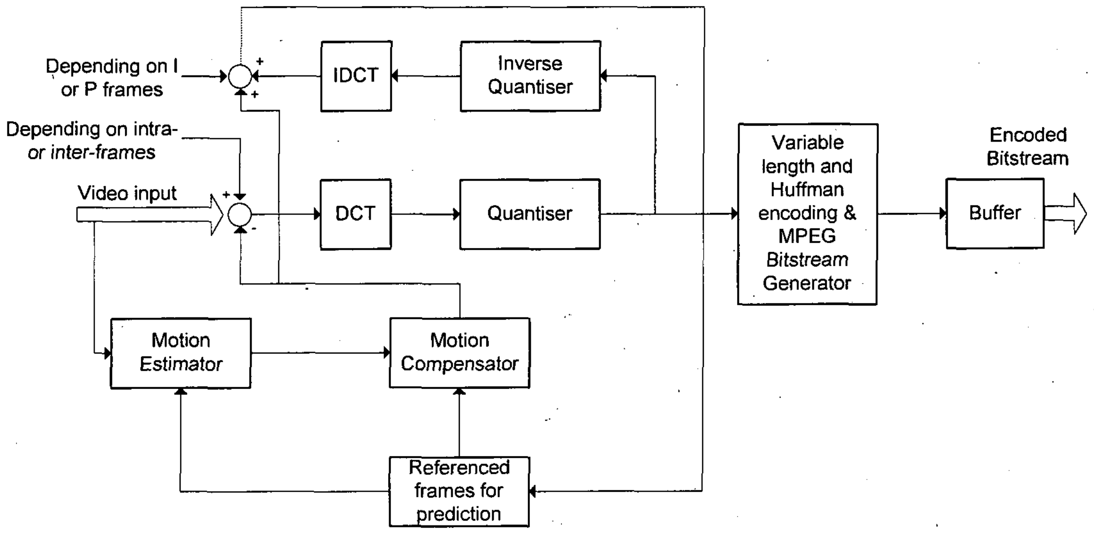

Depending on I or P frames
Depending on intra- or inter-frames
Video input
IDCT
Inverse Quantiser
DCT
Quantiser
Variable length and Huffman encoding & MPEG Bitstream Generator
Encoded Bitstream
Buffer
Motion Estimator
Motion Compensator
Referenced frames for prediction

Figure 5.4 Block diagram for the MPEG encoding scheme [98]

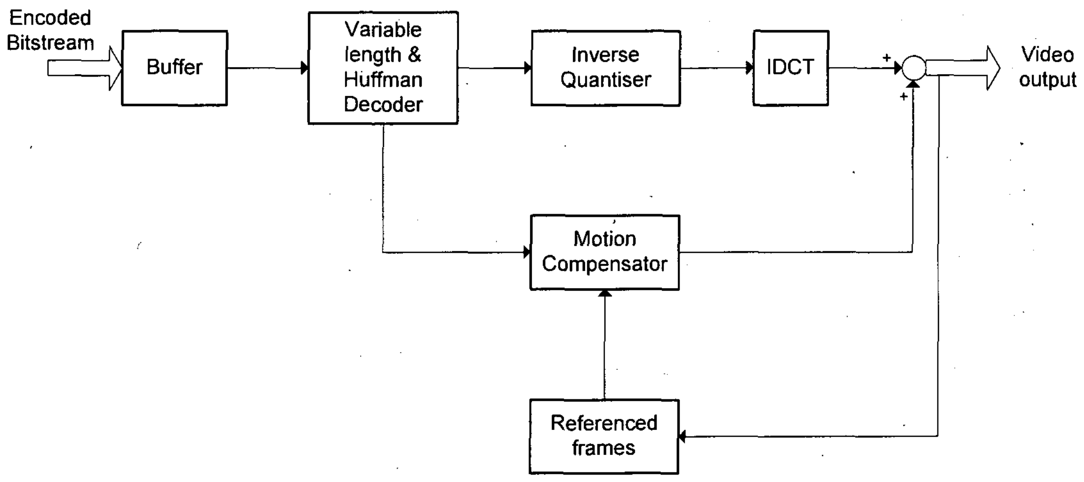


Figure 5.5 Block diagram for the MPEG decoding scheme [98]

5.3.1 Motion Compensation and Macroblock Size

As mentioned in the previous section, the most popular video coding standards use a macroblock size of 16x16 pixels. Reducing the block size to 8x8 or even 4x4 reduces the energy in the residual block and hence the data needed to be transmitted or stored. This is because smaller block sizes allows for finding better matches in the reference frames and hence smaller residual energy (residual energy is the subtraction of current macroblock from chosen candidate macroblock in the reference frame). However, smaller block sizes leads to increased search operations and increase in the number of motion vectors. Nevertheless, a smaller block size is necessary for detailed and complex motion scenes and therefore, to solve the increase in motion vectors, block sizes must be chosen such that transmitted/stored motion vectors do not outweigh the benefit of reduced residual energy. In terms of search operations, pipeline stages can help increase throughput and therefore more searches can be done. However, pipeline stages must be implemented in such a way as to minimise power consumption without affecting throughput (performance) and this is the focus of this chapter.

5.4 Motion Compensation Implementation

There are many low-power motion compensation designs [139-143] and while they have good power saving features, there is no systematic analysis for the impact of pipeline stages on power-performance trade-offs. This section details the implementation of the motion compensation algorithm based on the Berkeley MPEG player [99] as this algorithm is one of the most computationally expensive parts of the MPEG decoder [100]. The aim is to be able to add the necessary pipeline stages to meet the performance requirement, and any stages that do not affect performance must be identified and removed to save power consumption. The PSI method of chapter 3 is used to identify redundant stages and provide a set of results from which the users can choose the one that meets their performance requirement. The Berkeley motion compensation algorithm was implemented in C and is shown in Appendix D.1. In order to synthesise the algorithm the first step was to translate the C code into VHDL. The author translated the C-code motion compensation algorithm into 1040 lines of VHDL code with arithmetic and logical operations instantiated in the code (see Appendix D.2). Figure 5.6 highlights the major arithmetic and logical units in the motion compensation (note, not all connections are shown in the diagram for reasons of clarity).

As part of PSI step 1 (section 3.4), a data flow diagram of the FEs is needed; this is fulfilled by Figure 5.6. In PSI step 2, the delays of each FE must be calculated. To do this, each element was synthesised using a 90nm technology library at 1.3V supply voltage, a floorplan (see Appendix A.7 for example script) was generated for each element and full cell placement was done using Magma BlastFusion [33] with the final result being a post-layout database. The FEs delay was extracted from the post-layout database using Magma commands [33], shown in Table. 5.1. With the aid of elements delay, the naive pipeline stages (this is the least-effort approach of introducing pipeline stages by adding a stage after each FE without considering the number of elements output) were inserted in an ad hoc manner as shown in Figure 5.7.

The elements delays shown in Table 5.1 present an interesting design challenge. As can be seen, these delays are relatively large (for example the multiplier res_mul_inst_0 has a delay of 9.6ns) and if the delays are not reduced, performance of the whole motion compensation design could be compromised because stage delay will be too big, and therefore the system frequency will have to be reduced. In order to address this problem, elements with large delays are identified and examined to see if pipelining could be applied within the elements themselves.

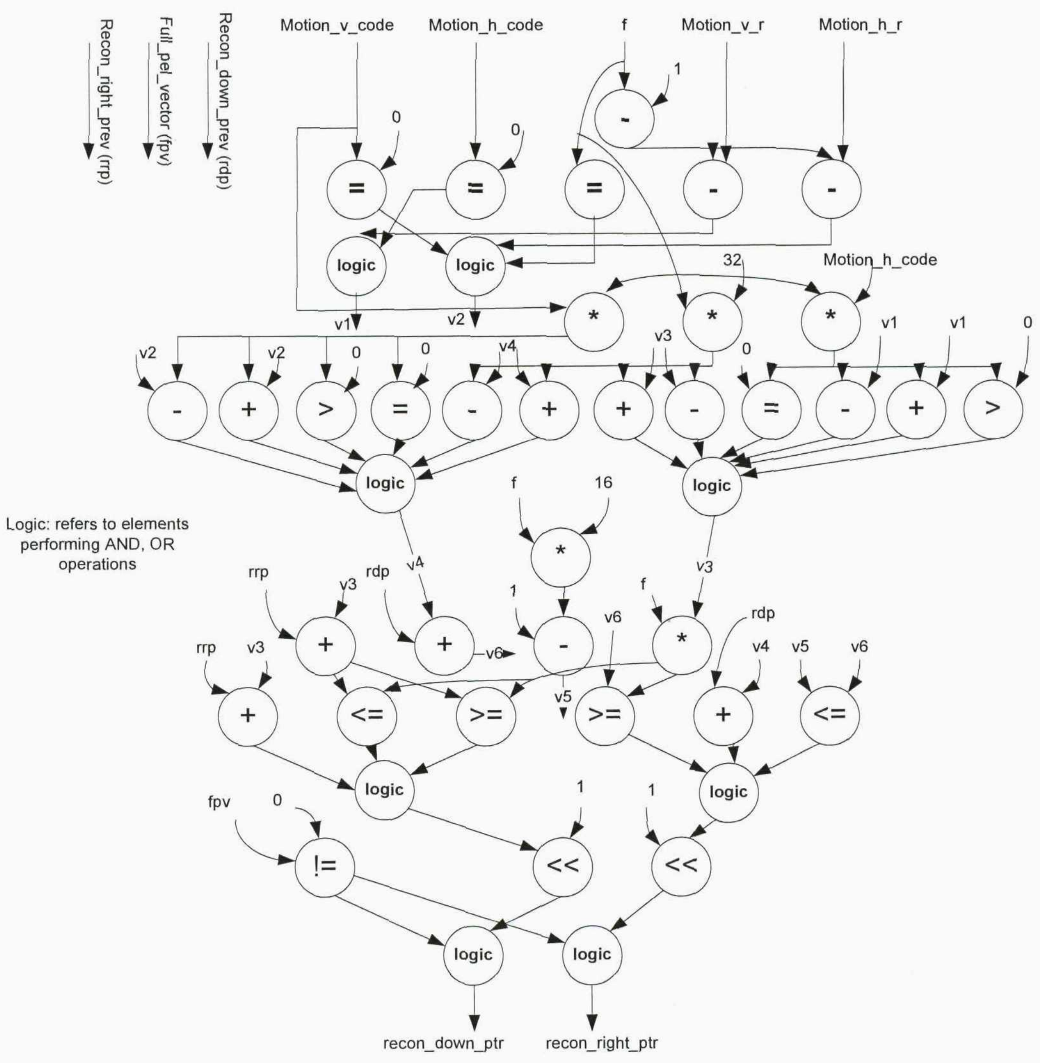


Figure 5.6 DFG of Motion Compensation

Table 5.1 Motion Compensation Element Delays

No.	Element Name†	Operation	Delay (ps)
1	Res_minus_inst_0	-	2600
2	Res_minus_inst_2	-	3800
3	Res_minus_inst_1	-	3800
4	Res_equal_inst_0	=	2800
5	Res_equal_inst_1	=	2800
6	Res_equal_inst_2	=	2800
7	Res_mul_inst_0	*	9600
8	Res_mul_inst_1	*	8200
9	Res_mul_inst_2	*	8200
10	Res_comp_v_r_inst	Logic	1000
11	Res_comp_h_r_inst	Logic	1000
12	Res_greaterthan_inst_0	>	4100
13	Res_add_inst_1	+	3800
14	Res_minus_inst_3	-	3800
15	Res_equal_inst_3	=	2800
16	Res_minus_inst_4	-	3800
17	Res_add_inst_0	+	3800
18	Res_add_inst_3	+	3800
19	Res_minus_inst_5	-	3800
20	Res_equal_inst_4	=	2800
21	Res_greaterthan_inst_1	>	4100
22	Res_add_inst_2	+	3800
23	Res_minus_inst_6	-	3800
24	Res_right_inst	Logic	1200
25	Res_down_inst	Logic	1200
26	Res_mul_inst_3	*	8200
27	Res_mul_inst_4	*	8200
28	Res_minus_inst_7	-	2600
29	Res_add_inst_6	+	3800
30	Res_add_inst_4	+	3800
31	Res_lessequal_inst_5	<=	4100
32	Res_add_inst_7	+	3800
33	Res_greaterequal_inst_6	>=	4100
34	Res_greaterequal_inst_7	>=	4100
35	Res_lessequal_inst_2	<=	4100
36	Res_add_inst_5	+	3800
37	Res_down_ptr_inst	Logic	1000
38	Res_right_ptr_inst	Logic	1000
39	Res_shfl_inst_0	<<	5400
40	Res_shfl_inst_1	<<	5400
41	Res_notequ_inst_4	!=	4100
42	Res_right_shift_inst	Logic	1000
43	Res_down_shift_inst	Logic	1000

† Elements in this table are listed from Figure 5.9 starting from top to bottom, right to left.

The strategy adopted in this scenario is to add a pipeline stage through elements that have delays bigger than 2600ps. This effectively divides the elements in two and provides a more reasonable delay without introducing too much complexity to this task (i.e. each candidate element was examined to identify the best pipeline stage insertion point that would divide the delay in half without adding too many stage registers).

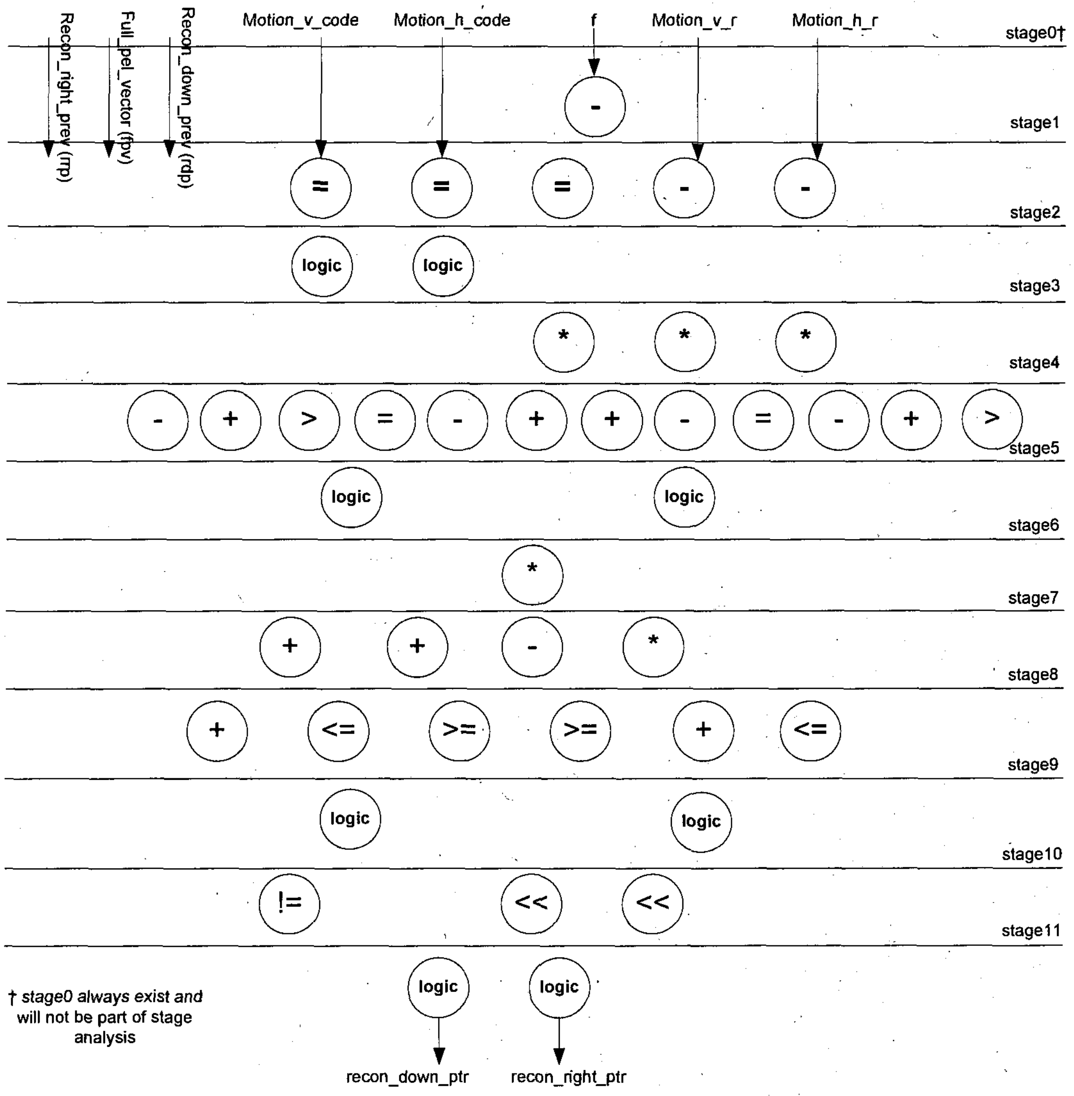


Figure 5.7 Ad hoc Naive Pipeline Stages for Motion Compensation

Figure 5.8 shows an example element with a pipeline stage. Pipeline stage within elements would manifest itself as being part of a bigger pipeline stage at the motion compensation level. Figure 5.9 shows the new naive pipeline stages, and how element stages fit in the whole picture. It is important to take care of elements with small delays when adding the new stages.

If they can be executed within the time it takes one half of a divided element to execute, these small delay elements are not divided. For example, in Figure 5.12 res_comp_h_r_inst takes 1000ps to execute and could easily fit in the stage that executes half of res_mul_inst_0 which takes 4800ps. In this case, there is little advantage in dividing res_comp_h_r_inst into two stages.

Comparing Figure 5.7 to Figure 5.9, it is clear that Figure 5.9 has four additional stages and potentially this means an increase in run time when analysing the pipeline stages. However, these stages are necessary in order to guarantee that when the PSI is run there are

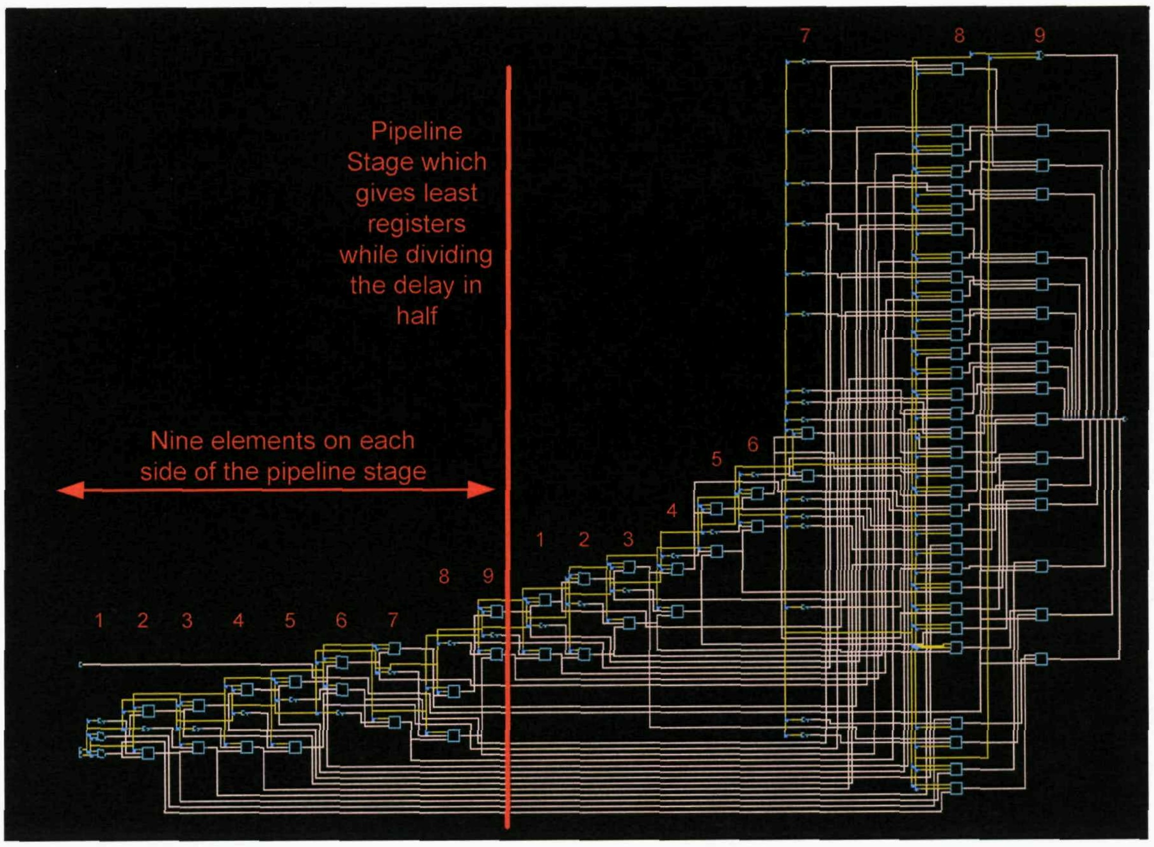

Pipeline Stage which gives least registers while dividing the delay in half
Nine elements on each side of the pipeline stage
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9

Figure 5.8 Pipeline Stage inside Adder Element

reasonable stage delays to meet user performance. If the PSI can achieve performance without using these additional stages, then the stages are not implemented and power consumption is reduced. On the other hand, if these four stages are not present in the initial PSI run and user performance requirement is high, then the PSI cannot deliver the performance required because stage delays are too large. As will be shown in section 5.5, from the PSI point of view, run time is not significant compared with other stage analysis methods (e.g. exhaustive search) and could easily absorb a relatively large increase in the number of stages without excessive increase in run time.

The input to the PSI algorithm is the data array shown in Table 5.2 representing the naive stages, which captures the information in Figure 5.9.

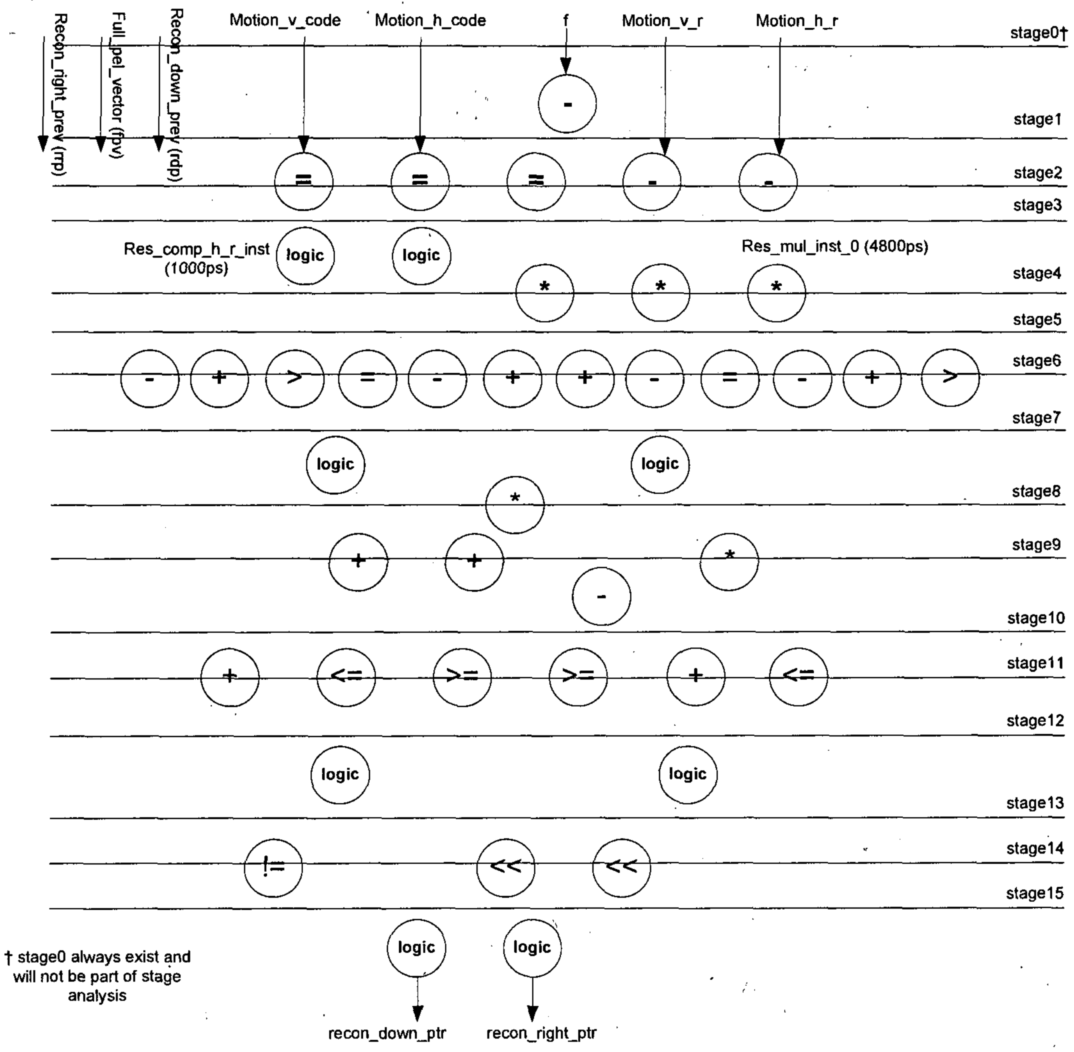


Figure 5.9 Naive Pipeline Stages with Element Stages

Table 5.2 PSI Input Data Array for Motion Compensation

Stages	Delay (ps)	Outputs
1	2600	119
2	1900	335
3	1900	171
4	4800	474
5	4800	153
6	2050	864
7	2050	242
8	4100	260
9	4100	383
10	4100	153
11	2050	567
12	2050	89
13	1000	51
14	2750	250
15	2750	85

5.5 Experimental Results

This section compares the experimental results from the PSI method versus those from naive approach and exhaustive algorithm. The following sub-sections are organised as follows: section 5.5.1 describes the estimation of switching probability and section 5.5.2 outlines register and combinational logic power equations and their uses in the following experiments. Section 5.5.3 presents Experiment 1 which deals with power consumption results, Experiment 2 showing register count results is presented in section 5.5.4, an energy efficiency graph is obtained in Experiment 3, section 5.5.5 and finally Experiment 4, section 5.5.6 shows the changes in number of stages as clock period is increased.

In all the above-mentioned experiments, the algorithms were executed on Linux 64bit system with 2 CPUs running at 2.6GHz and 15G RAM. Synopsys PrimePower is the tool used for power estimation.

5.5.1 Switching Probability Estimation

Stimulus data were obtained from running five real videos downloaded from the Berkeley web site [99]. The videos were played on the original C-code Berkeley decoder and the data to the motion compensation module was captured using a monitor block. The monitor is a C-code block added to the decoder and writes the entire input stimulus to the compensation module to a file; as a result, five files were generated - one for each video.

In order to estimate power consumption at the FEs level, the next task is to work out the stimulus for each FE since what has been captured so far is the data at the motion compensation boundary. There are two options available for this task: 1) a monitor block would have to be written for each FE in the original Berkeley decoder, and since there are 43 FEs, 43 monitors would have to be created. Furthermore, PrimePower uses a Value Change Dump (VCD) file for power analysis generated from gate level simulation and in order to analyse the power for each element, 45 VHDL gate-level test benches would have to be created. Since there are five videos, the total number of VCDs files generated is 45x5=225 files. 2) The other option is to create a one gate-level test bench for the motion compensation module, from which a VCD file is generated. Using this VCD file the average switching probability for the whole design can be worked out. Clearly, option 1 is more accurate than the second option; however, it is far more cumbersome to implement, is prone to manual errors and generates a large amount of data. Option 2 is by far more practical to implement, widely used in industry and the inaccuracy introduced in the power estimation due to averaging of the switching probability is relatively small. This is the method adopted in this case study for power estimation.

The simplest way to estimate the average switching probability is by simulating the gate-level netlist of the motion compensation algorithm with the five stimulus data files and writing out a VCD file for each stimulus. In PrimePower, the same gate-level netlist is loaded along with each of the five VCD files, and from this database, a simple function is executed to calculate the average probability throughout the design. It was found that for the motion compensation module signals on average had a switching probability of 22.8%.

5.5.2 Register and Combinational Logic Equations

Having worked out the average switching probability in the design, it is now possible to generate equations that represent power consumption as a function of frequency for both a single register and overall combinational logic. These two equations are needed because when the PSI algorithm is run, a range of frequencies will be evaluated and instead of running PrimePower to estimate power at each of the frequencies, a quicker way is to estimate power using these equations. Appendix E.1 shows the data used in generating the equations and Table 5.3 states these equations.

Table 5.3 Combination Logic and Register Power Equations

Combination Logic Power Equation	$y = 0.0185e^{-0.0001x}$
Register Power Equation	$y = 2\times10^{-5}e^{-0.0002x}$
Where y is power in watts and x is the clock period in ps.	

The power consumption of all the registers is simply the power consumed by a single register at the desired frequency (calculated from the register equation) multiplied by the total number of registers needed to implement the pipeline stages, which is given by the PSI algorithm. The total power consumed in the design is the sum of total register power and combinational logic power calculated at the desired frequency.

5.5.3 Experiment 1: Power Consumption Results

The aim of this experiment is to confirm that PSI can generate low power pipeline stages for the motion compensation design without impacting performance. The PSI was run with the naive data array shown in Table 5.2 and different clock periods ranging from 4800ps to 10400ps. At each clock period the PSI algorithm examined the naive pipeline stages and using values for Φ ranging from 0 to 17, and incremented by 0.5 (the maximum value for Φ is the result of dividing the largest number outputs by the smallest, 864/51 which is approximately 17, see section 4.3.3.2 for further details). Those stages that do not affect performance are eliminated leaving only necessary stages. The total dynamic power consumption is therefore the sum of combination logic power and total register power as shown in Equation 5.1 derived from chapter 3, where O_i and P_i are obtained from PSI for each stage and P_{Comb} and P_{reg} are the combinational logic and register power values determined from the equations in Table 5.3 at the required clock period.

$$\text{Total Power Consumption} = \sum_{i=1}^{i=\max stages} O_i * P_{reg} * (1 - P_i) + \sum_{i=1}^{i=\max element} P_{Comb-i} \qquad (5.1)$$

Figure 5.10 shows the power consumption of the motion compensation algorithm versus clock period. Clock period starts at 4800ps, this choice is governed by the slowest element delay from Table 5.2 and this happens to be an element at Stage 4. As a comparison, the exhaustive search (ES) algorithm (see section 4.3.5 for more details about ES) was also run and the results are shown in the above figure. These results clearly demonstrate that the PSI tracks the exhaustive search algorithm very well with only a slight improvement in the results generated by the exhaustive search when clock period is between 8800ps and 9200ps. To all intents and purposes, the improvement is very small and the two results are practically identical. As for the naive approach, the power consumption is a lot higher than with the other two algorithms because no stage optimisation has taken place. At 4800ps, the PSI generates a result that consumes 30% less power compared with the naive approach.

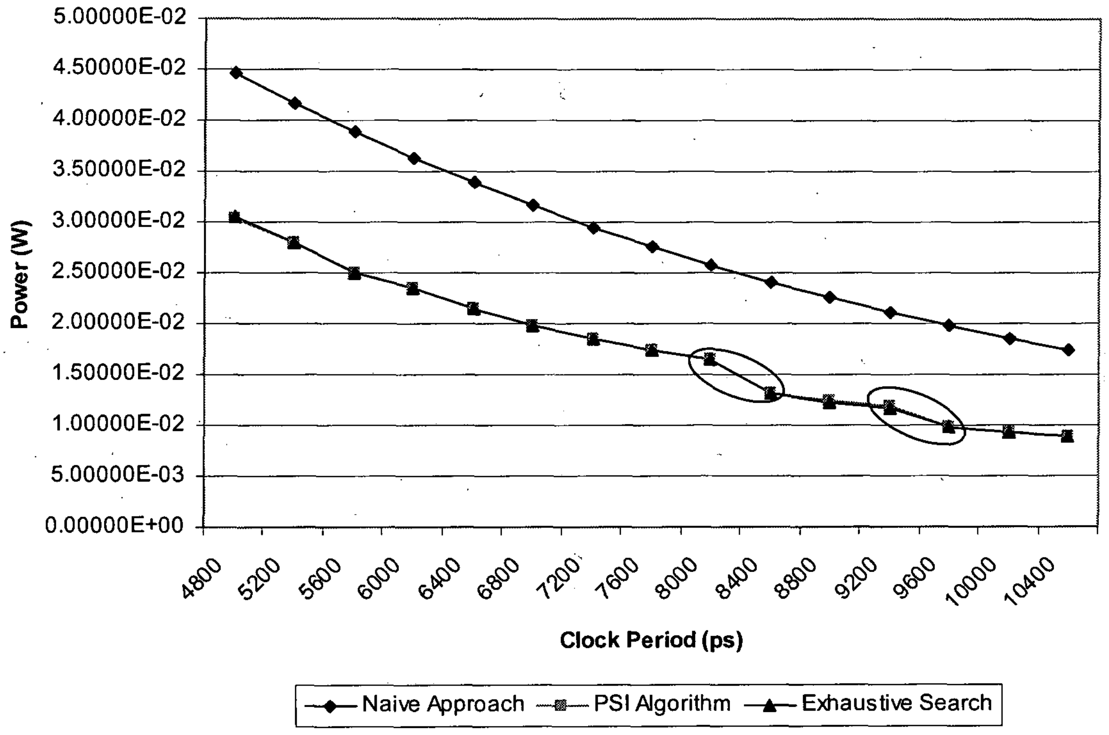


Figure 5.10 Power Consumption for Different Clock Periods

There is a noticeable power drop highlighted by the oval shapes in the above figure when the clock period was changed from 8000ps to 8400ps and from 9200ps to 9600ps. This is because a significant reduction in register count was observed, and as a result, power consumption was reduced. Experiment 2 explains the details of the reduction in register count at those clock periods.

In summary, as expected, the PSI generates better results than naive stages, which are as good as those of ES. Furthermore, this experiment confirms the expected trend of decreasing power consumption as clock period is increased.

5.5.4 Experiment 2: Register Count Results

Having examined the power consumption results in the previous section it is no surprise to see that, in this experiment, PSI and exhaustive search register count tracked each other very well. Figure 5.11 shows the register count versus clock period for both PSI and exhaustive search. As expected, the overall trend of falling register count as frequency decreases is confirmed by this figure. This is because as the frequency decreases the clock period becomes larger and opportunities for removing stages increases, which results in smaller register count. However, what is noticeable from Figure 5.11 are the two significant decreases in register count as highlighted by the oval shapes. In order to understand these two scenarios both PSI and exhaustive search were re-run at these four clock periods with a debugging mode enabled in the two algorithms to report every decision made by the algorithms.

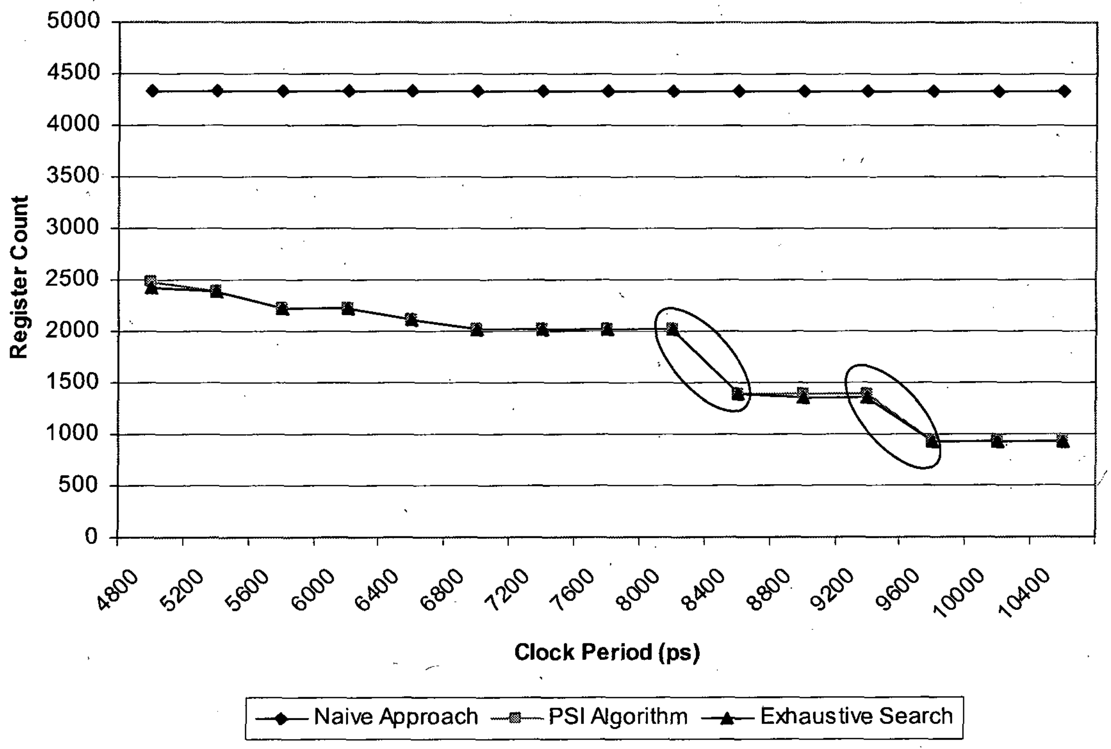


Figure 5.11 Register Count for Different Clock Periods

Table 5.4 shows the stages that the PSI did optimise away (represented by 0) at the four critical clock periods: 8000ps, 8400ps, 9200ps and 9600ps.

Table 5.4 PSI Results at the Critical Clock Periods: 8000ps, 8400ps, 9200ps and 9600ps

Stages	**Element Delay (ps)**	**Outputs**	**Clock Periods**			
			8000ps	8400ps	9200ps	9600ps
1	2600	119	0	0	0	0
2	1900	335	0	0	0	0
3	1900	171	1	1	1	1
4	4800	474	1	1	1	0
5	4800	153	1	1	1	1
6	2050	864	0	0	0	0
7	2050	242	1	0	0	0
8	4100	260	1	1	1	1
9	4100	383	1	0	0	0
10	4100	153	1	1	1	1
11	2050	567	0	0	0	0
12	2050	89	0	0	0	0
13	1000	51	1	1	1	1
14	2750	250	0	0	0	0
15	2750	85	0	0	0	0

It is observed from Table 5.4 that changing the clock period from 8000ps to 8400ps has resulted in the optimisation of two stages (stage 7 and 9). This has been made possible by the fact that the elements in stage 7 or stage 9 can now be allocated to one stage without violating the clock period. For example, elements at stage 7 have maximum delay of 2050ps and it is now possible to combine stages 6, 7 and 8 and still be with in the clock period of 8400ps. Relating the optimisations to the outputs column, it is clear that stages 7 and 9 greatly contribute to the reduction in register count. In summary, the result indicates that if the system can operate with a clock period running at 8400ps; this period would represent a tipping point in terms of register count reduction and therefore lower power consumption is to be expected. Similar optimisation mechanism is taking place at 9600ps but this time stage 4 has been optimised away compared to the clock period of 9200ps and represents a saving of 474 registers.

Since the results of the PSI and the exhaustive search correlate very well, one expects to see similar register count reduction for the exhaustive search. Table 5.5 shows the optimised stages for exhaustive search and as can be seen, they are identical to PSI (Table 5.4) with only two differences, represented by the highlighted boxes.

Table 5.5 Exhaustive Search Results at the Critical Clock Periods: 8000ps, 8400ps, 9200ps and 9600ps

Stages	**Element Delay (ps)**	**Outputs**	**Clock Periods**			
			8000ps	8400ps	9200ps	9600ps
1	2600	119	0	0	1	0
2	1900	335	0	0	0	0
3	1900	171	1	1	0	1
4	4800	474	1	1	1	0
5	4800	153	1	1	1	1
6	2050	864	0	0	0	0
7	2050	242	1	0	0	0
8	4100	260	1	1	1	1
9	4100	383	1	0	0	0
10	4100	153	1	1	1	1
11	2050	567	0	0	0	0
12	2050	89	0	0	0	0
13	1000	51	1	1	1	1
14	2750	250	0	0	0	0
15	2750	85	0	0	0	0

5.5.5 Experiment 3: Energy Efficiency Results

Figure 5.12 shows that the optimal clock period that maximises the energy efficiency metric energy*delay^2 is approximately 30ns for both PSI and exhaustive search. What is also shown in Figure 5.12 is the naive approach; even though it has a higher peak than the other two approaches, it is in fact consuming more power at its optimal clock period (~25ns). Actually at 30ns clock period the naive approach is 24% less energy efficient than the PSI and exhaustive search. It is interesting to note that the energy efficiency for the PSI and exhaustive search happens at a low frequency 33MHz (clock period of 30ns). This is because of the observation that combinational logic power is the dominant part of total power consumption for clock periods from 10ns to 29ns. Clock periods > 29ns show a rapid decline in combination logic power contribution and, as a result, the rapid decline in the energy efficiency curve as shown in Figure 5.12. More details on this can be found in Appendix E.1.

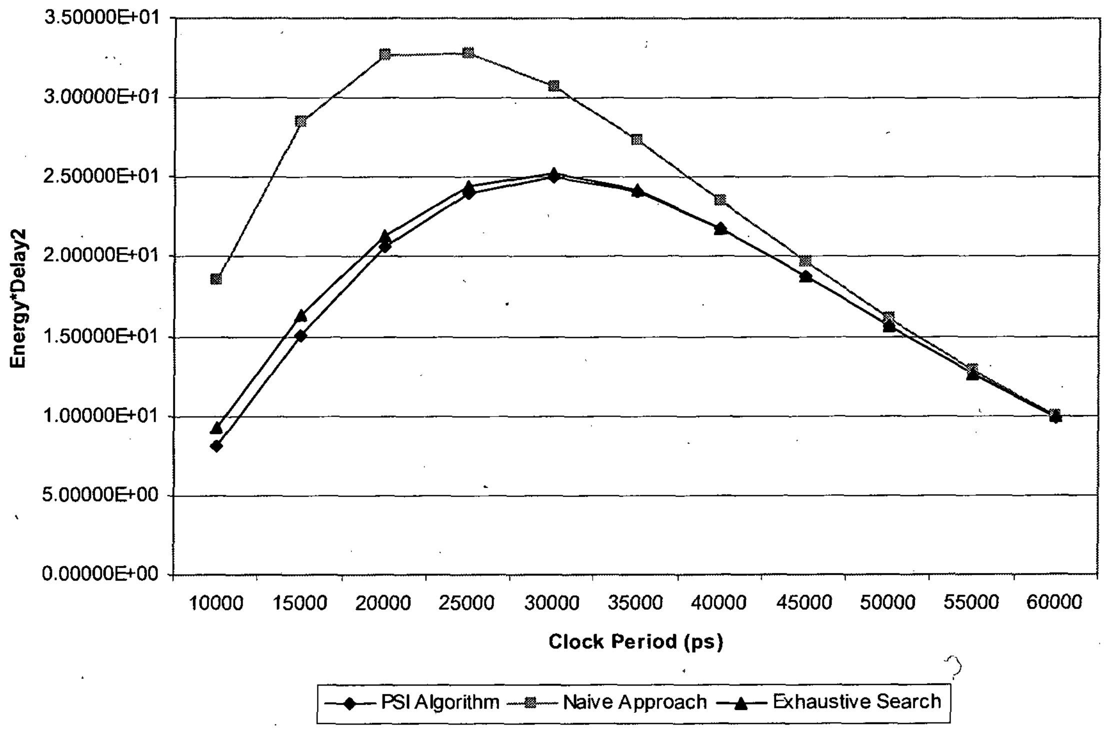


Figure 5.12 Energy Efficiency Curves

5.5.6 Experiment 4: Stage Count Results

Figure 5.13 shows the changes in the number of stages versus clock period for the PSI, naive approach and exhaustive search. Apart from the result at clock period 4800ps, the rest of the data are identical for both PSI and exhaustive search. The naive approach is fixed at 15 stages because no stage optimisation is done in this approach. At 4800ps the exhaustive search has one less stage than PSI; however, the reduction in terms of power consumption is limited because the optimised stage is stage 13 (Table 5.2) which has only 51 outputs to register. In terms of latency, the exhaustive search has a shorter latency than the PSI at this frequency.

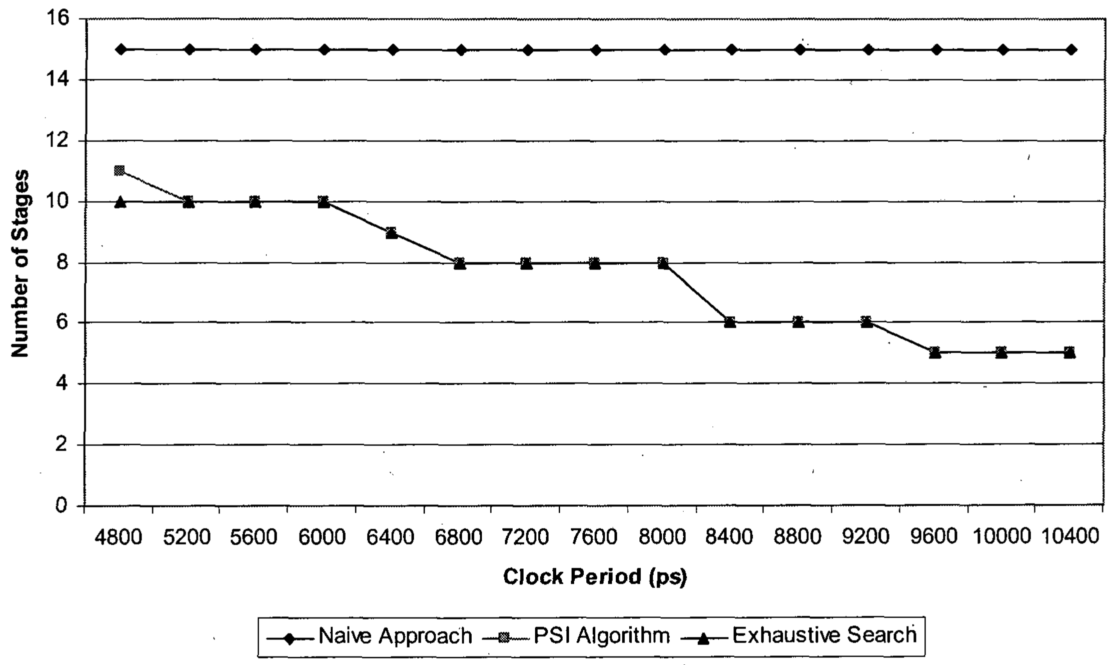


Figure 5.13 Stage Count for Different Clock Periods

5.6 Run Time Complexity

The results shown in the previous experiments confirm that the PSI generates results as good as the exhaustive search. What is also important to establish is how well the two algorithms perform in terms of run time. This section aims to derive, empirically, equations estimating run time (given the number of stages in a design) for both ES and PSI. To achieve a meaningful comparison, the computations executed by each algorithm need to be quantified, and for this case study Table 5.6 summarises the computations for the two algorithms. Note, the timing results were obtained using Linux 64bit system with two CPUs running at 2.6GHz and 15G RAM.

Table 5.6 Run Time Comparison between PSI and ES

PSI	Exhaustive Search
• 15 stages to test • Φ ranged from 0-17 (34 values incrementing by 0.5) • 15 operating frequencies evaluated • Total iterations: 15x34x15 = 7650	• 15 stages represent 2^15=32768 combination to test • 15 operating frequencies evaluated • Total iterations: 32768x15=491520
Run Time: 4.29 seconds	**Run Time: 2 minutes and 17 seconds**

It is clear from the above table that the PSI is 97% faster than the exhaustive search. This is a significant run time saving considering the power results are practically identical between the two algorithms. From these results, it is possible to make an approximate estimation as to how long it would take both algorithms to run if, for example, a design had 30 stages instead of 15. In this case study the exhaustive search took approximately 279 microseconds per iteration, and similarly the PSI took 561 microseconds per iteration. Assuming that in the future the exhaustive search could be run on a faster computer and can run an iteration in 100 microseconds, the 30 stages example results in 2^30 which is approximately 1G iterations, and these would take 28 hours to complete (assuming only one clock period being examined). For the PSI, the run time will be a function of the number of stages and the value of Φ. Assuming Φ will take a range similar to the motion compensation, 34 values, then the PSI would take 34x30x561us = 0.6 seconds to run (also assuming only one clock period under evaluation). From this simple estimation, it becomes apparent that the larger the number of stages the longer the exhaustive search will take to run. Thirty stages are not uncommon especially in graphic accelerators where the rendering pipeline could have hundreds of stages [101]. Using the actual motion compensation (Table 5.6) and TDPFPADD (section 4.3.5.2) run times along with the above estimated run time, it is possible to empirically predict for any number of stages what the corresponding execution time will be and this prediction is shown in Figure 5.14.

To formulate run time, the trend line function in Microsoft Excel was used to generate from Figure 5.14 the two equations shown in Table 5.7.

Table 5.7 Run Time Estimation Equations for PSI and ES (Assuming One Clock Period is Analysed)

PSI run time equation	$y = 8 \times 10^{-5} x$
ES run time equation	$y = 7 \times 10^{-5} e^{0.409x}$
Where y is run time in hours and x is number of stages	

In summary, the PSI run time has a linear relationship with respect to the number of stages whereas the exhaustive search has an exponential trend, and therefore, as the number of stages increases, the run time for certain designs will become excessively long. Since both algorithms generate practically identical power results, the PSI is the preferable method when considering complex and large designs. Furthermore, the run time estimate in Table 5.7 assumed only one clock period being analysed. If on the other hand a larger number of clock periods are used, the run time for ES would increase even further (run time would be measured in days rather than hours), while the PSI would only incur a very small run time hit (PSI run time would still be in minutes).

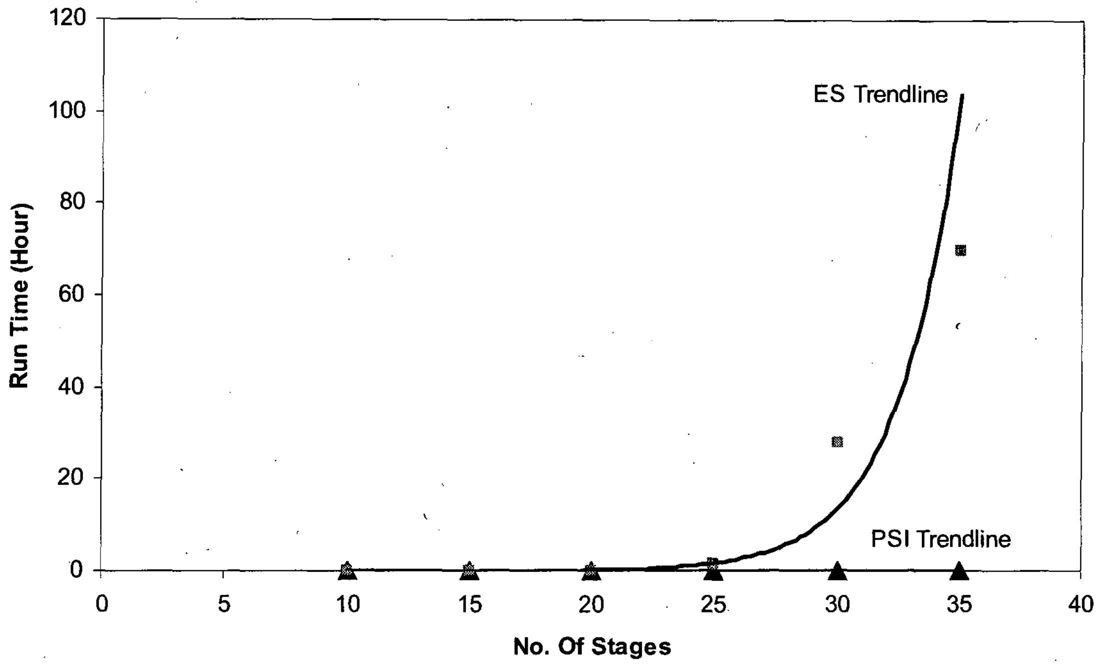


Figure 5.14 Run Time Curves for PSI and ES

5.6.1 The Effect of Φ Values on Run Time and the PSI Flags *I*, *W* and *P*

In chapter 3, section 3.5.1, Φ is defined as a variable that takes a positive value ≥0 and this variable influences the optimisation of naive stages. In addition, chapter 4, section 4.3.3 described how the range of Φ is worked out and observed in section 4.3.3.2 the impact different values of Φ can have on the power consumption of PSI solutions. This section analyses the impact of Φ on run time.

As mentioned in section 5.5.3, the range of Φ was set from 0 to 17 incrementing by 0.5 which gives 34 values to be tested; it would be interesting to know what values of Φ gave the best power results. Figure 5.15 shows the values of Φ that resulted in low power solutions for the motion compensation at each evaluated clock period.

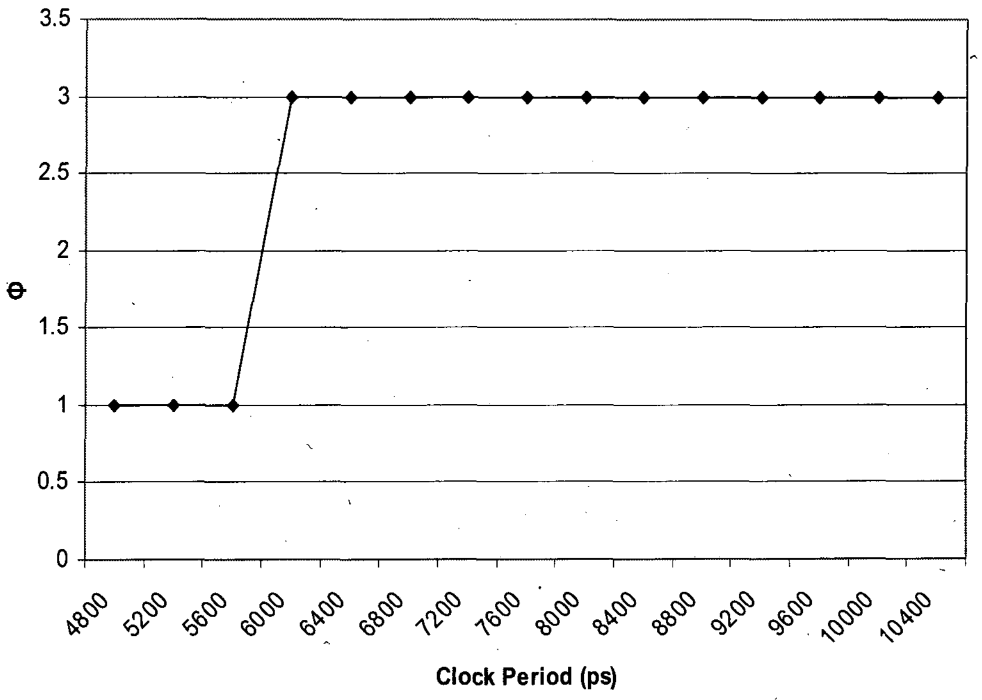


Figure 5.15 Values of Φ Giving Least Power Consumption

Figure 5.15 indicates that most of the low power results are at Φ = 3, and therefore running the PSI with Φ values beyond 3 (3.5 to 17) does not provide low power solutions. If the values above 3 are eliminated then run time will decrease (see Table 5.6 on how run time was calculated for PSI). This in turn should help obtain a solution quicker for designs that have a large number of stages and many clock periods to be examined. However, it is not easy a priori what the actual values of Φ are that give the best results; this is very much design dependant. However, one can make a guess based on a random selection of outputs, and from those an upper limit of Φ is established. For example, in Table 5.2 choosing stages 2 and 3 randomly and calculating the upper limit for Φ, which will be two (dividing the largest number by the smallest and rounding the final number), is a good starting point for the PSI algorithm. If there are low power solutions in this range (0 to 2) then it will take less run time to find them and the solution may well be within the power budget allocated by the designer. If however the solutions are not within budget then a larger range needs to be tested.

To better understand why, when $\Phi = 3$, the PSI method generates the smallest number of registers and therefore lowest power consumption solution, the effect of Φ on the PSI algorithm flags, *I*, *W* and *P* needs to be investigated. The value of P, which will ultimately decide if a stage should exist or not, depends on the value of *W* and *I* as described in section 3.5. For this investigation, the PSI method is applied to the motion compensation module with a target clock period of 8000ps and two values for Φ: 3 and 1. For each value of Φ, the flags *I*, *W* and *P* will be noted for each of the 15 naive stages. Note, as indicated earlier, we know that $\Phi = 3$ will generate a low power solution and it is interesting to compare this solution against a more power consuming solution resulting from setting $\Phi = 1$. Table 5.8 shows how the values for *I*, *W* and *P* changes when Φ value is changed. Note, in Table 5.8, $P = 0$ means a stage has to exist. The solution using $\Phi = 3$ has eight stages compared with nine stages for the solution using $\Phi = 1$, and this explains why $\Phi = 3$ generates a low power consumption solution. The value of *I* for both solutions is identical as shown in Table 5.8, and this indicates that at least from timing point of view, the value of Φ is not influencing *I* (see Equation 3.14, on how Φ could indirectly influence the value of *I*). Therefore, the reason why the solution using $\Phi = 1$ has gained an extra stage, must be as a result of the value of *W*. This is confirmed by looking at stage 1 in Table 5.8 which shows that this is the extra stage in the $\Phi = 1$ solution because $W_1 = 1$. The reason why $W_1 = 1$ is because $W_1 = O_2/O_1 = 335/119$ (from Table 5.2), evaluating to 2.82, which then leads to the inequality $\{W_1 > \Phi\}$ being true, setting $W_1 = 1$ and a stage is inserted. This inequality for the solution using $\Phi = 3$ is false, leading to $W_1 = 0$ and hence no stage is inserted.

Table 5.8 Values of *I*, *P* and *W* when Φ = 3 and 1 and the Clock Period is 8000ps

Stage No.	Φ=3			Φ=1		
	I	*P*	*W*	*I*	*P*	*W*
Stage 1	0	1	0	0	0	1
Stage 2	0	1	0	0	1	0
Stage 3	1	0	0	1	0	1
Stage 4	1	0	0	1	0	0
Stage 5	0	0	1	0	0	1

Stage 6	0	1	0	0	1	0
Stage 7	1	0	0	1	0	1
Stage 8	1	0	0	1	0	1
Stage 9	1	0	0	1	0	0
Stage 10	0	0	1	0	0	1
Stage 11	0	1	0	0	1	0
Stage 12	0	1	0	0	1	0
Stage 13	0	0	1	0	0	1
Stage 14	0	1	0	0	1	0
Stage 15	0	1	0	0	1	0

In summary, for designs that are complex with many pipeline stages and clock periods to be examined it may be worth considering choosing a smaller range for Φ to get a solution quicker which may well be within the targeted power budget.

5.7 Concluding Remarks

In this chapter, the motion compensation algorithm was used to further demonstrate the low power solutions offered by PSI. This design was challenging in terms of the number of FEs and requirements to reduce power consumption. Another challenge was encountered regarding large element delays, and if these were not addressed, they could have affected the overall performance. A viable solution of pipelining within elements was proposed and implemented and the result was more sensible element delays without complicating the overall task. A synthesisable VHDL implementation of the design was generated and, using the PSI method, pipeline stages were examined for a number of clock periods. The final results show that the PSI still outperforms the naive approach with up to 30% dynamic power reduction adding further credibility to the PSI method. Further, the PSI method was able to generate low power solutions as good as the exhaustive search but with significant run time advantage. The PSI solutions were generated for each clock period and in terms of register count, power consumption and number of stages all were similar to the exhaustive search solutions. This adds confidence to the PSI underlying algorithm that the routines to generate low power solutions are correct and can compete very well with other algorithms such as the exhaustive search. Even though the PSI was significantly faster than the exhaustive search, it was suggested that further run time improvement could be made by reducing the range of values of Φ based on the trend of stage outputs. This should help run time for designs with a large number of stages. Using run time figures from this case study and in conjunction with figures from TDPFPADD design in chapter 4, empirical equations were developed to estimate the run time for any given number of stages. These equations state that the PSI run times have a linear relationship with an increasing number of stages whereas the exhaustive search has an exponential relationship.

Chapter 6

PSI and Voltage Scaling

Voltage scaling is one of the most effective power reduction techniques. Lowering the supply voltage can reduce power consumption since CMOS circuit power quadratically scales with the supply voltage. Voltage scaling has been applied extensively in VLSI designs due to the simple implementation requirements [144]. Compared with dynamic voltage and frequency scaling (DVFS) [138] that requires DC/DC voltage converter, specialised frequency register and voltage controlled oscillator (VCO), voltage scaling technique does not employ these hardware components simplifying the design flow [145]. In addition, for DVFS to execute voltage and frequency scaling, the operating system needs to decide the minimum CPU clock speed that meets the workload requirements and this in itself is challenging particularly for designs with a large number of tasks with different deadlines. Multiple supply voltage (MSV) is another technique that creates voltage islands on the ASIC with each island powered by different supply voltage [145]. In essence, the supply voltage of each island is lowered to reduce power consumption while fulfilling the performance requirement. This is effectively applying voltage scaling at the island level resulting in MSV at the ASIC level. These islands in general are big architectural level designs with well-defined requirements. Therefore, it is a prerequisite for a good MSV implementation that the voltage scaling technique is applied efficiently at the architecture level.

As already highlighted in the literature survey in chapter 2, section 2.1.2, the earlier in the design flow the voltage scaling technique is applied, the better rewards in terms of power reduction. The aim of this chapter is to investigate combining voltage scaling and the PSI method of chapter 3 to create a new method that can be applied at the architecture level to further reduce dynamic power consumption.

In chapters 4 and 5, the PSI was used to optimise the naive pipeline stages with the supply voltage fixed at 1.3V. This chapter explores the possibility of utilising slack to reduce the supply voltage and therefore reduce dynamic power consumption. Since voltage has a quadratic relationship to power consumption, a voltage-aware PSI should yield better results in terms of power without affecting performance. The supply voltage reduction is considered during the analysis of the naive pipeline stages and once a solution is found the new supply voltage will be used during the normal operation of the design and the same for each FE; this will be elaborated on further in section 6.2.

Although the power reduction technique employed by the PSI method in chapter 3 aims to reduce the number of registers, the slack in each stage has so far not been considered. Reducing the supply voltage implies FEs delay will get longer, and to accommodate the new delays, each stage must have sufficient slack otherwise timing violations will occur. In general, better slack is achieved when there are few FEs in a stage but the down side of that means there will be a large number of stages and hence a large number of registers. This presents a trade-off between keeping more of the naive stages and the slack that these stages offer. The larger the slack, the lower the supply voltage and therefore the bigger the saving in terms of power consumption. However, power saving may not be possible if a large number of naive stages are kept and the slack available gives a voltage reduction which may not be enough to offset the power consumed by the large number of registers. The aim is to find a balance between the number of registers in the stages and the slack offered by the stages to give a voltage reduction that offsets the power consumed by the registers. In other words, the cost function for the PSI will no longer be just reducing the total number of registers but instead the slack will also be considered as an additional cost function, and that is what is investigated in this chapter.

The outline for this chapter is as follows. Section 6.1 gives a brief motivational example and defines the concept of slack and worst slack. Section 6.2 describes the updates necessary to make the PSI flow voltage aware. Section 6.3 presents experimental results for the challenging design of MPEG-1 motion compensation (chapter 5) showing the PSI with voltage scaling can reduce power consumption. Finally, section 6.4 concludes this chapter.

6.1 Motivational Example

In this section, a simple example is given to show that slack in pipeline stages can be used to lower the supply voltage and hence reduce power consumption. Figure 6.1 shows design A and two possible pipeline stage solutions (1 and 2) generated by PSI, both of which meet the system clock period of 10ns but with different slack. Solution 1 has one pipeline stage and the slack in this stage is 1ns whereas solution 2 has two pipeline stages with two slacks, 5ns and 6ns. Worst slack in this context is defined as the smallest time unit resulting from subtracting element(s) delay from the clock period. For example, solution 1 has worst slack of 1 ns compared to solution 2 with worst slack of 5 ns. Identifying the value of worst slack in a pipelined design is important because it determines how far the supply voltage can be lowered before timing violation occurs.

It is clear from this example that the supply voltage reduction for solution 1 will be limited whereas solution 2 can reduce voltage significantly since there is five times the slack of solution 1 to be used (worst slack in solution 2 is 5ns). The improved slack in solution 2 has come with a penalty of two additional registers; however, voltage reduction should still be able to offset the power consumed in these two registers and consume less overall dynamic power than solution 1.

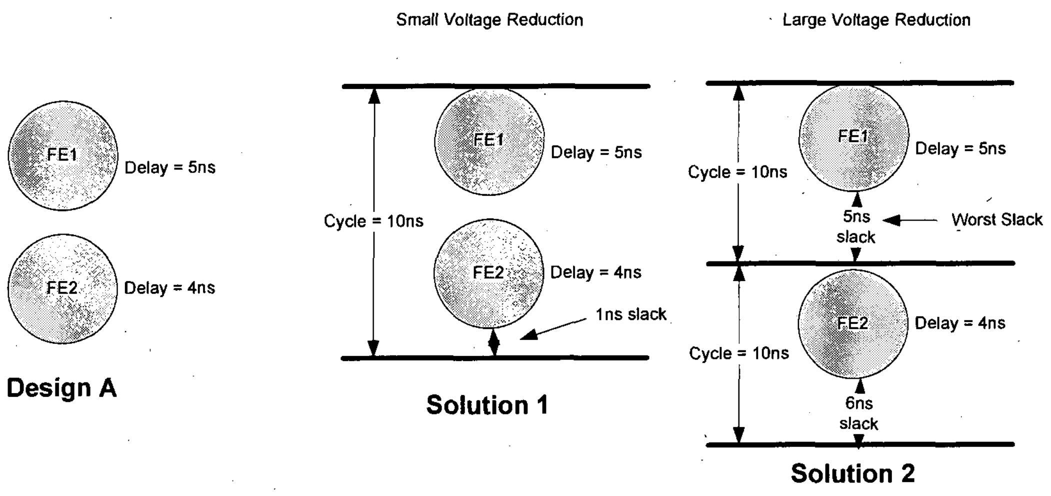


Figure 6.1 An Example Illustrating Worst Slack In Pipelined Designs

Integrating this into how the PSI optimises naive pipeline stages, solution 2 is viewed as keeping more of the naive stages. It is important to note the merit of both solutions in Figure 6.1 from a power consumption point of view is not determined in the PSI algorithm but instead in the analysis of the solutions generated by PSI. Section 6.2 will explain this point further. Although the latency has increased in solution 2 the main focus of PSI is to achieve required throughput with reduced power consumption, and therefore latency assumes a secondary role in the PSI method.

In this simple example, the power consumption of the additional registers can easily be absorbed by the voltage reduction but in other designs, this may not be the case. What is important though is to analyse the solutions that meet the clock period and from those reject the solutions that impact power due to a large number of registers, and without sufficient slack to offset the power consumed in the additional registers.

6.2 Updating the PSI Flow with Voltage Scaling (PSI-VS)

As already illustrated in section 6.1 given a system clock period the PSI will analyse the naive stages and generate different solutions that meet the clock period. Each solution potentially could have a different number of stages and therefore different worst slack. Adding voltage scaling as a parameter means each of these solutions will be examined to see if worst slack can be explored to reduce the supply voltage. Figure 6.2 shows the new functionality added to the result analysis phase of the PSI flow (Figure 3.12) in order to make voltage scaling possible. To make the comparison between the two flows easy to follow, in this chapter, Figure 3.12 is included as Figure 6.3. It is important that the new supply voltage does not violate timing requirement, and therefore a thorough analysis of elements delay at the new supply voltage is necessary to meet this requirement.

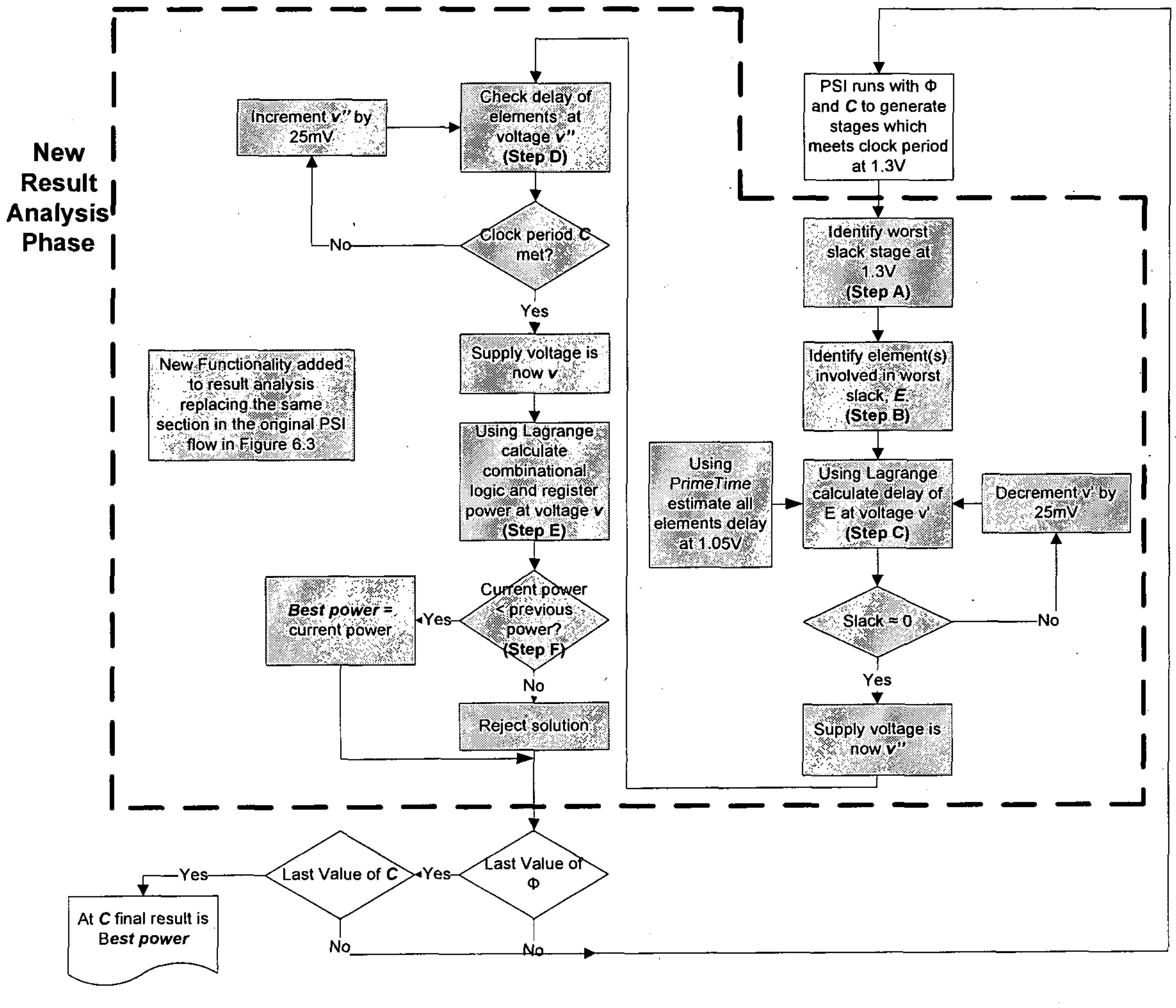
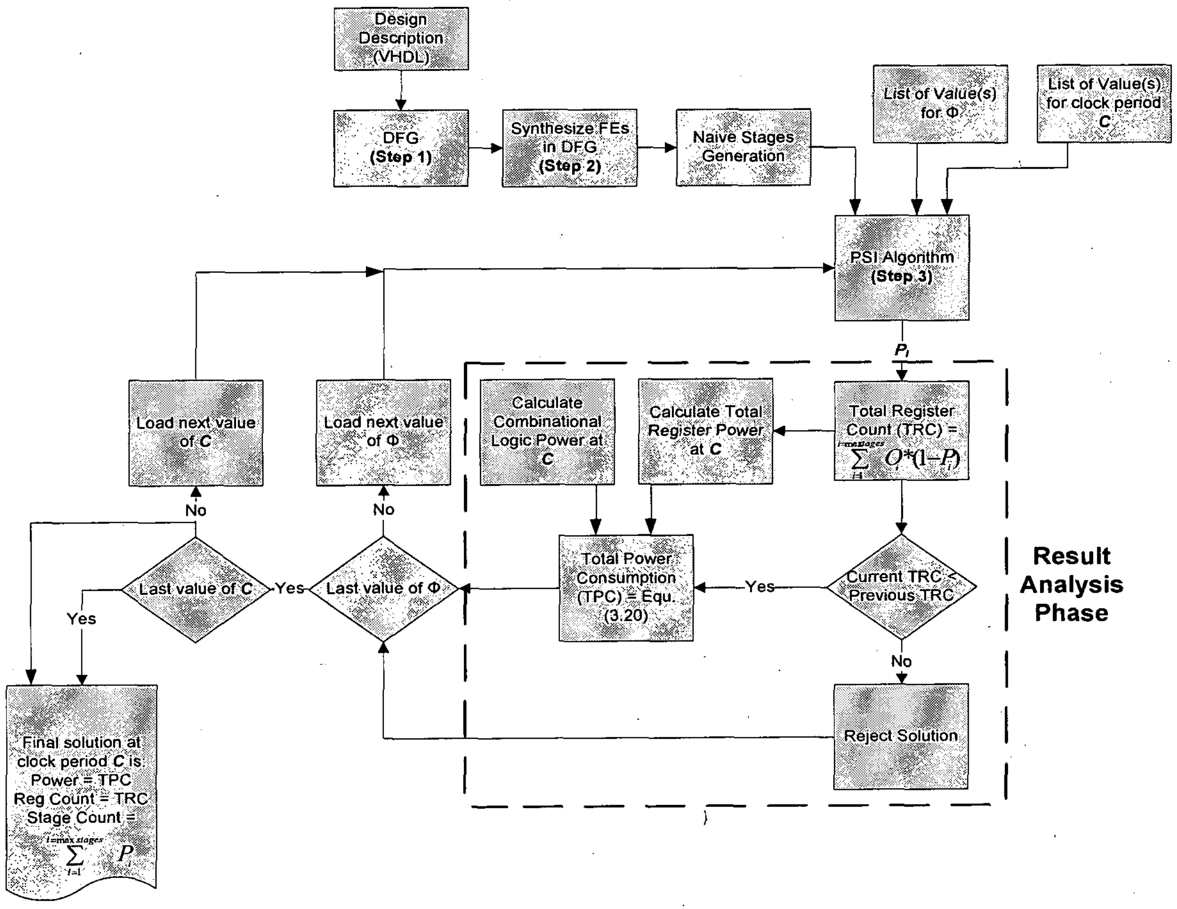

New Result Analysis Phase
PSI runs with Φ and C to generate stages which meets clock period at 1.3V
Identify worst slack stage at 1.3V (Step A)
Identify element(s) involved in worst slack, E (Step B)
Using PrimeTime estimate all elements delay at 1.05V
Using Lagrange calculate delay of E at voltage v' (Step C)
Decrement v' by 25mV
Slack = 0
Yes
No
Supply voltage is now v''
Check delay of elements at voltage v'' (Step D)
Increment v'' by 25mV
Clock period C met?
Supply voltage is now v
Using Lagrange calculate combinational logic and register power at voltage v (Step E)
Current power < previous power? (Step F)
Best power = current power
Reject solution
New Functionality added to result analysis replacing the same section in the original PSI flow in Figure 6.3
Last Value of Φ
Last Value of C
At C final result is Best power

Figure 6.2 The New Result Analysis Phase to Support Voltage Scaling

In Figure 6.2, the main PSI algorithm remains the same, and the only difference to the previous flow in Figure 6.3 is the analysis of PSI solutions. Instead of previously rejecting the set of solutions which have a large number of registers (Figure 6.3), these solutions will also be considered in the new analysis.

The new result analysis phase will have six steps (Steps A-F in Figure 6.2) executed on each solution. Step A identifies worst slack by examining each pipeline stage to determine its slack based on element(s) delay and system clock period and the worst slack is the smallest slack value in that solution. Knowing the worst slack and the stage it is in, Step B creates a list (*E*) which includes the FE(s) in the worst slack stage. Step C uses this list to determine the delay of the elements at a reduced supply voltage with the aim of making the slack zero or close to zero.



Design Description (VHDL)
DFG (Step 1)
Synthesize FEs in DFG (Step 2)
Naive Stages Generation
List of Value(s) for Φ
List of Value(s) for clock period C
PSI Algorithm (Step 3)
P_i
Load next value of C
Load next value of Φ
Calculate Combinational Logic Power at C
Calculate Total Register Power at C
Total Register Count (TRC) = $\sum_{i=1}^{i=max\,stages} O*(1-P_i)$
No
No
Last value of C
Yes
Last value of Φ
Total Power Consumption (TPC) = Equ. (3.20)
Yes
Current TRC < Previous TRC
Result Analysis Phase
Yes
No
Reject Solution
Final solution at clock period C is: Power = TPC Reg Count = TRC Stage Count = $\sum_{i=1}^{i=max\,stages} P_i$

Figure 6.3 Original PSI Flow

As elements delay at maximum voltage (1.3V based on the technology library) is already known (required by PSI to generate solutions), if elements delay is also known at another supply voltage then using Lagrange [103], it is possible to estimate the delay at different supply voltages. In this case, the state-of-the art 90nm technology provides library cells which are characterised for 1.05V. Using Synopsys PrimeTime tool [32], the FEs delay at 1.05V is estimated and used as an input to Step C. Equation 6.1 shows Lagrange delay where v is the voltage applied to the design, v_0 is 1.05V, v_1 is 1.3V, $d(v_0)$ is the delay of the FE at voltage v_0, $d(v_1)$ is the delay of the FE at voltage v_1, and $D(v)$ is the delay of the FE at voltage v.

$$\begin{aligned} D(v) &= \sum_{k=0}^{1} d(v_k) L_k(v) \\ L_0(v) &= \frac{(v - v_1)}{(v_0 - v_1)} \\ L_1(v) &= \frac{(v - v_0)}{(v_1 - v_0)} \end{aligned} \tag{6.1}$$

Using Lagrange, the delay of element(s) in ***E*** (list of element(s) contributing to worst slack) is estimated at the new supply voltage of 1.275V (1.3V-25mV assuming decrementing voltage step of 25mV. The step value is a parameter and can be set to any value). The slack at 1.275V is then checked (subtraction element(s) delay at 1.275V from clock period) and once it is zero or close to zero the new supply voltage is saved as ***v''***. In order to be sure that timing has not been violated in other stages (when using supply voltage ***v''***), Lagrange is used to estimate the delay of elements in the remaining stages and this is done in Step D.

If timing violation is seen at supply voltage ***v''***, the voltage is increased by 25mV and elements delay at that stage are evaluated using the new supply voltage. Further increases in supply voltage by steps of 25mV are possible to speed up elements delay and make the slack zero. Once a supply voltage ***v*** is found which removes the timing violation, all the subsequent stages will use this voltage to evaluate their elements delay. Stages prior to current stage do not require re-evaluation as they already meet timing with original supply voltage, and therefore higher supply voltage should make their delay even shorter.

Using Synopsys PrimePower tool, combinational logic and register power is estimated at 1.3V and 1.05V. Step E uses Lagrange power Equation 6.2, [103] - where v is the voltage applied to the design, v_0 is 0V, v_1 is 1.05V, v_2 is 1.3V, $p(v_0)$ is 0mW, $p(v_1)$ is the power of the FE at voltage v_1, $p(v_2)$ is the power of the FE at voltage v_2, and $P(v)$ is the power of the FE at voltage v - to estimate power consumption at supply voltage v. Finally, Step F checks if the current total power consumption is less than previous result, and if this is the case, the current result becomes the best result so far, otherwise it is rejected.

$$\begin{aligned}
P(v) &= \sum_{k=0}^{2} p(v_k)L_k(v) \\
L_0(v) &= \frac{(v-v_1)(v-v_2)}{(v_0-v_1)(v_0-v_2)} \\
L_1(v) &= \frac{(v-v_0)(v-v_2)}{(v_1-v_0)(v_1-v_2)} \\
L_2(v) &= \frac{(v-v_0)(v-v_1)}{(v_2-v_0)(v_2-v_1)}
\end{aligned} \tag{6.2}$$

If there are further values for Φ or C to be analysed, then the whole flow described above is run again. The final result from this flow is a solution for each clock period that meets timing but where possible at reduced supply voltage, and therefore reduced dynamic power consumption. One may question why the PSI algorithm in Figure 6.2 has been executed with FEs delay at 1.3V. The PSI algorithm could have used the elements delay at 1.05V since they are available. However, it is the intention of this thesis to give the designer the full range of performances that a pipelined design can achieve, and this could not be done starting with delays at 1.05V. Executing the PSI algorithm with delays at 1.3V gives it a chance to generate the best performance solution for a given design. If the designer does not require such performance, then the algorithm will find a solution for the reduced supply voltage. Starting with delays at 1.05V, one can only find solutions at this voltage (or lower supply voltage) but not at 1.3V and therefore the designer is unable to see what the best possible performance a design can achieve is.

Although so far the new approach has been described as an update to the "flow"; however, because voltage scaling introduces many new steps, the update could well be described as a new method. From this point, combining the PSI and voltage scaling will be known as the PSI-VS method. Table 6.1 compares the PSI (chapter 3) and PSI-VS outlining the main differences between the two methods.

Table 6.1 Comparison between PSI and PSI-VS Methods

PSI Method	PSI-VS Method
PSI algorithm is run to generate different solutions that meets clock period requirement.	Same as PSI method
For each solution, examine the number of registers. Reject the solutions with the highest register count.	For each solution, identify the worst slack and reduce the supply voltage accordingly. Reject the solutions with the highest power consumption.
Final solution is a pipeline stages with the least register count and power consumption that meets the clock period requirement.	Final solution is a pipeline stages with the least power consumption but not necessarily the least register count. The solution meets the clock period requirement.

It is important to note that given a clock period requirement, the low power PSI solution may be different to the low power solution for the same clock period using the PSI-VS method. Furthermore, for example, if the pipeline stages for the low power solution from the PSI method are handed over to the PSI-VS (no change is made to the number and location of the stages) to apply voltage scaling, the power consumption using PSI-VS may not be the lowest power consumption at the given clock period that the PSI-VS can achieve. This is because at the given clock period, the PSI method was optimising the number of registers and not slack. When the PSI-VS is applied to the solution (the PSI solution), slack becomes an important factor and it may be that the stages in that solution offers little slack and therefore the potential for lowering the supply voltage is reduced. In other words, for a given clock period, the low power solution using the PSI method may be different to the solution from the PSI-VS method in terms of the number of registers, pipeline stages and, most importantly, the actual power consumption. It is expected that the PSI-VS method should be able to generate at worst case the same solution as the PSI method but, where slack is available, lower power consumption should be possible. To illustrate the points raised above, Figure 6.4 shows three example solutions all of which meet a clock period of 10ns for a given design using the PSI method. According to the PSI method, solution A is the least power consuming solution as the number of registers in A (20 registers) is smaller than solution B and C. Therefore, solutions B and C are rejected when using the PSI method. A careful consideration of solutions B and C reveal that they have more slack than solution A. In fact, solution B has worst slack of 4ns and C has 7ns compared with 1ns for A. Baring in mind these three solutions could have been generated using the PSI or PSI-VS (because the PSI algorithm is common between the two methods), the PSI-VS cannot reject solutions B and C in similar manner to the PSI method.

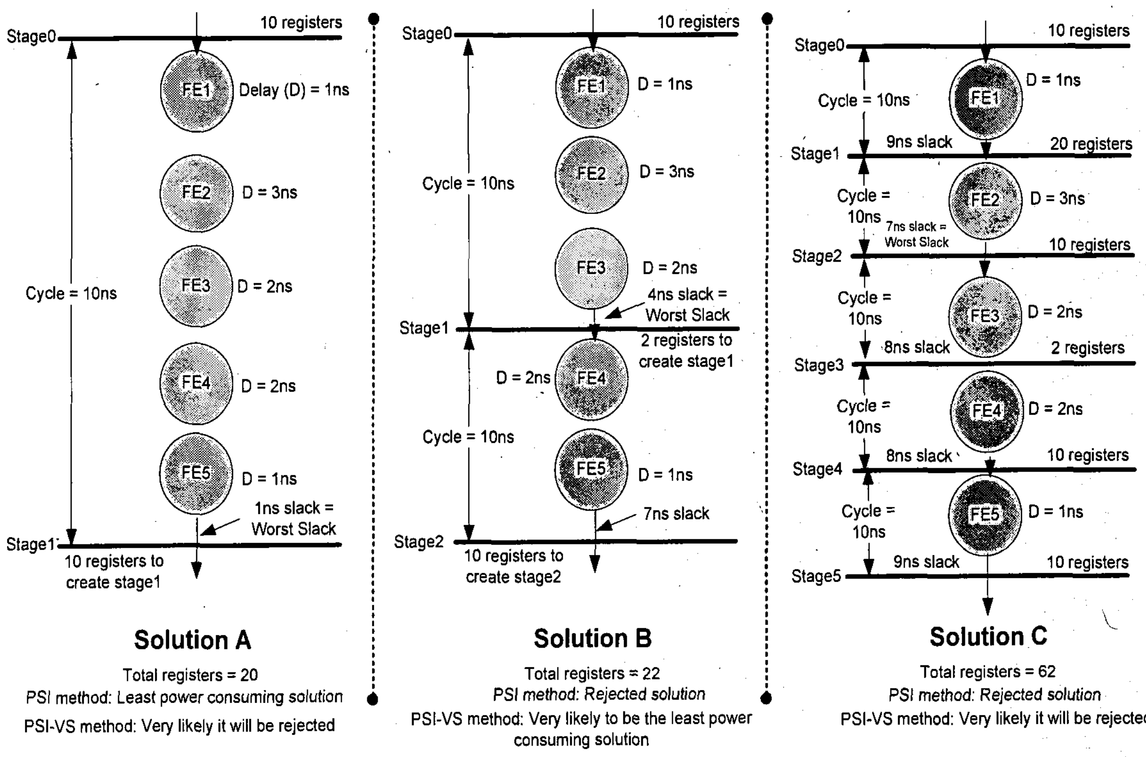


Figure 6.4 Three Pipeline Solutions with Different Merits for PSI and PSI-VS Methods

This is because the PSI-VS considers slack along with register count and therefore all three solutions must be considered to see if the worst slack in these solutions could be used to lower the supply voltage. Intuitively, solution C might be considered the least power consuming solution when applying voltage scaling because it has the biggest worst slack (7ns) out of the three solutions. One could envisage this slack is used to lower the supply voltage considerably and therefore power consumption is reduced significantly. However, on closer examination of solution C, the stages have added a significant number of registers totalling 62 and even if the supply voltage is lowered by a big factor, the additional power consumption in these registers would outweigh the benefits of the supply voltage reduction. On the face of it, solution B could be the least power consuming solution when applying voltage scaling even though two additional registers have been added when compared to solution A. Nevertheless, without a thorough analysis of the three solutions it is not possible to say for sure which is the better solution in terms of power consumption. This is where the PSI-VS method can help in identifying the least power consuming solution. For these reasons, the low power solution using the PSI-VS method may be different in terms of stage count, register count and power consumption when compared to the PSI method with the same clock period requirement. In addition, the PSI-VS is expected to take longer to run when compared to the PSI method as each solution has to be examined for potential scaling of the supply voltage. Finally, it is clear that solution C has the longest latency while at the same time it has the highest throughput. As already indicated in chapter 3, section 3.2, in this thesis throughput assumes a primary role and latency is secondary.

6.3 Experimental Results

This section aims to demonstrate the low power capabilities of PSI-VS on the MPEG-1 motion compensation module introduced in chapter 5. This design has been chosen over the TDPFPADD due to its many challenging complexities and large number of FEs. A number of clock periods ranging from 4800ps to 104ns were evaluated for this design and six experiments were run: Experiment 1, showing the dynamic power consumption with and without VS; and Experiment 2, illustrating register count trend. In Experiment 3, stage counts were analysed and the supply voltage for the best solutions at each clock period is shown in Experiment 4. Experiment 5 illustrates the slack at each clock period. Experiment

6 compared the solutions from the exhaustive search (ES) algorithm with VS to PSI-VS. Finally, Experiment 7 considers the trend in Φ values for the PSI-VS solutions.

All of the main steps (A to F) and sub-steps (e.g. PrimeTime estimation) highlighted in Figure 6.2 have been automated using TCL scripting language [89]. This language has the advantages of being easy to learn and implement, and is used by many EDA tools. The script was executed on a Linux 64bit system with two CPUs running at 2.6GHz and 15G RAM.

6.3.1 Motion Compensation Test Bench and PSI-VS

The motion compensation module has already been described in section 5.4 and in the context of the PSI-VS investigation, the same test bench is used along with the input stimulus. The input to Step C in Figure 6.2 is the delay estimation of each FE in the motion compensation at 1.05V using Synopsys PrimeTime tool [32] (delays are shown in Table 6.2). The flow used in PrimeTime as shown in Figure 6.5 highlights the verilog netlist (netlists are re-used from section 5.4) for each of the FEs in the motion compensation and the technology library with cell characterisation at 1.05V.

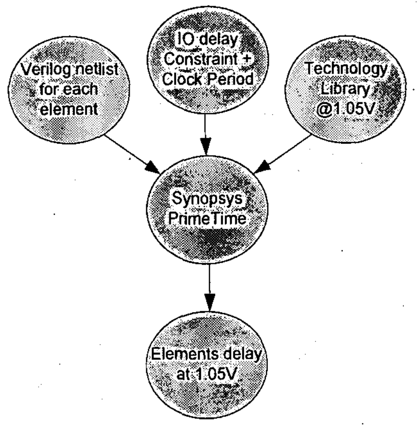

Verilog netlist for each element
IO delay Constraint + Clock Period
Technology Library @1.05V
Synopsys PrimeTime
Elements delay at 1.05V

Figure 6.5 Flow for Estimating FEs Delay @1.05V

This flow has been automated using TCL scripting language [89] to speed up the implementation, and is shown in Appendix E.2. The 90nm technology library allows maximum supply voltage of 1.3V and minimum voltage of 0.9V; however, only cell libraries at 1.3V and 1.05V are available and characterised. When executing the Lagrange delay equation in Step C of Figure 6.2, the minimum voltage it is allowed to evaluate is 0.9V in accordance with the minimum supply voltage offered by the technology library.

Table 6.2 Motion Compensation FEs Delay @ 1.3V and 1.05V

No.	Element Name†	Operation	Delay at 1.3V (ps)	Delay at 1.05V (ps)
1	Res_minus_inst_0	-	2600	3179
2	Res_minus_inst_2	-	3800	4753
3	Res_minus_inst_1	-	3800	4753
4	Res_equal_inst_0	=	2800	3232
5	Res_equal_inst_1	=	2800	3232
6	Res_equal_inst_2	=	2800	3232
7	Res_mul_inst_0	*	9600	10727
8	Res_mul_inst_1	*	8200	9312
9	Res_mul_inst_2	*	8200	9312
10	Res_comp_v_r_inst	Logic	1000	1165
11	Res_comp_h_r_inst	Logic	1000	1165
12	Res_greaterthan_inst_0	>	4100	4681
13	Res_add_inst_1	+	3800	4753
14	Res_minus_inst_3	-	3800	4753
15	Res_equal_inst_3	=	2800	3232
16	Res_minus_inst_4	-	3800	4753
17	Res_add_inst_0	+	3800	4753
18	Res_add_inst_3	+	3800	4753
19	Res_minus_inst_5	-	3800	4753
20	Res_equal_inst_4	=	2800	3232
21	Res_greaterthan_inst_1	>	4100	4681
22	Res_add_inst_2	+	3800	4753
23	Res_minus_inst_6	-	3800	4753
24	Res_right_inst	Logic	1200	1421
25	Res_down_inst	Logic	1200	1421
26	Res_mul_inst_3	*	8200	9312
27	Res_mul_inst_4	*	8200	9312

No.	Element Name†	Operation	Delay at 1.3V (ps)	Delay at 1.05V (ps)
28	Res_minus_inst_7	-	2600	3179
29	Res_add_inst_6	+	3800	4753
30	Res_add_inst_4	+	3800	4753
31	Res_lessequal_inst_5	<=	4100	4681
32	Res_add_inst_7	+	3800	4750
33	Res_greaterequal_inst_6	>=	4100	4681
34	Res_greaterequal_inst_7	>=	4100	4681
35	Res_lessequal_inst_2	<=	4100	4681
36	Res_add_inst_5	+	3800	4753
37	Res_down_ptr_inst	Logic	1000	1177
38	Res_right_ptr_inst	Logic	1000	1177
39	Res_shfl_inst_0	<<	5400	6113
40	Res_shfl_inst_1	<<	5400	6113
41	Res_notequ_inst_4	!=	4100	4675
42	Res_right_shift_inst	Logic	1000	1116
43	Res_down_shift_inst	Logic	1000	1116
† Elements in this table are listed from Figure 6.6 starting from top to bottom, right to left.				

Using Table 6.2 and based on the observation that in each pipeline stage of the motion compensation module (Figure 6.6 - this figure was first shown in chapter 5, Figure 5.9 and included here to aid the discussion), there could only be one critical path element dictating the delay of that stage. A critical path element represents the longest element delay in the stage. This simplifies the PSI-VS input data array to that shown in Table 6.3.

Table 6.3 Input Data Array to the PSI-VS

Stages	Delay (ps) @ 1.3V	Delay (ps) @ 1.05V	Outputs
1	2600	3179	119
2	1900	2377	335
3	1900	2377	171
4	4800	5364	474
5	4800	5364	153
6	2050	2341	864

Stages	Delay (ps) @ 1.3V	Delay (ps) @ 1.05V	Outputs
7	2050	2341	242
8	4100	4656	260
9	4100	4656	383
10	4100	4656	153
11	2050	2341	567
12	2050	2341	89
13	1000	1177	51
14	2750	3057	250
15	2750	3057	85

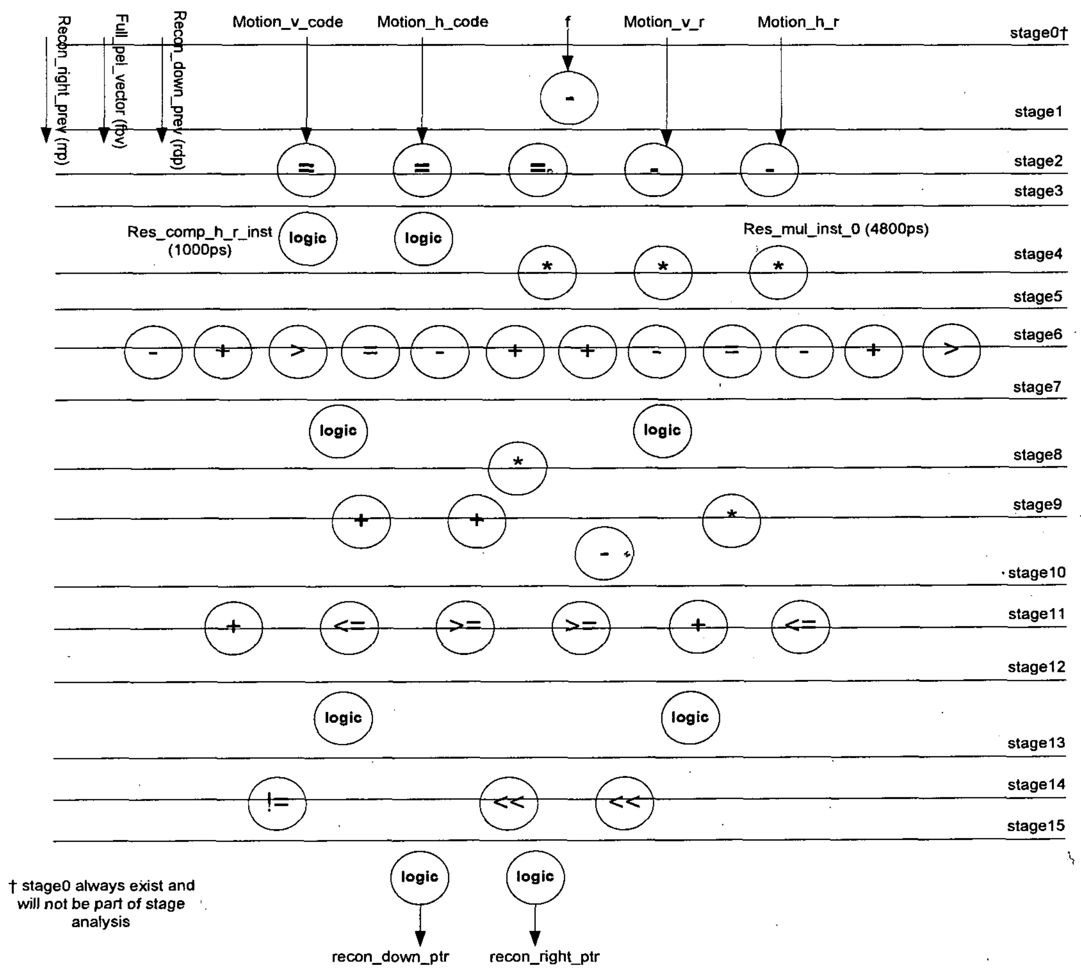


Figure 6.6 The Motion Compensation Test Bench

For example in stage4 of Figure 6.6, the critical path element is represented by the multiplier to the far right (Res_mul_inst_0) that has a delay of 9600ps at 1.3V and 10727ps at 1.05V as shown in Table 6.2. The remaining FEs in stage 4 do not influence the delay of the stage as they have smaller delays compared to Res_mul_inst_0. Note, in stage4, Res_mul_inst_0, has been pipelined for the reasons outlined in chapter 5, section 5.4 and the actual delay for this multiplier is divided by half to give a delay of 4800ps at 1.3V and 5364ps at 1.05V. These are the delays used in the input data array in Table 6.3 for stage 4. The same process is applied for all the remaining stages. It is important to note that the non-critical path elements do not influence the calculation of the worst slack in the stage. Worst slack is tied in with the critical path elements as these elements represent the longest possible path in the stage. For example, if the clock period requirement was set at 4800ps, then from Table 6.3, stage 4 would have worst slack delay of 0ns. On the other hand, if the clock period is set to 5000ps, the worst slack for stage 4 would be 200ps.

It is worth highlighting that the input data array for the PSI-VS represented by Table 6.3 differs from Table 5.2 in chapter 5 representing the input data array to the PSI method, in that it has an additional column indicating FEs delay at 1.05V. These delays are used in the Lagrange delay calculation (Equation 6.1) which is part of step C in Figure 6.2.

6.3.1.1 Experiment 1: Power Consumption Result

In this experiment, dynamic power consumption for the motion compensation module is estimated at different clock periods to evaluate the capability of PSI-SV method. The output from this experiment is the pipeline stages that meet the clock period requirement at reduced dynamic power consumption, achieved by lowering the supply voltage below the maximum of 1.3V. The first task is to find the supply voltage which makes the slack zero and then estimate the power consumption using this supply voltage. The following paragraphs explain this further.

As already mentioned in section 6.2, FEs delay at different voltages (1.3V down to 0.9V) is estimated using Lagrange delay, Equation 6.1, and the known elements delay in Table 6.3

(delays at 1.3V and 1.05V which represent the staring point for Equation 6.1). The final supply voltage is found when the worst slack is made zero (or close to zero) and there are no timing violations in any stage. This final supply voltage is then used to estimate the power consumption of the motion compensation module.

In order to use Lagrange power equation (Equation 6.2) with different supply voltages (between 1.3V and 0.9V), the power consumption for the combinational logic and a register must be determined at the two available technology libraries (1.3V and 1.05V) to provide the seed for Equation 6.2. Furthermore, in this experiment, the total power consumption is estimated at different clock periods, and therefore the combinational logic power and the register power must be evaluated at the same clock periods. Estimating the power in PrimeTime tool for each of the designer specified clock periods is a runtime intensive and impractical. A more practical approach is to use power equations describing the trends of combinational logic and register power using a selection of clock periods (see Appendix E.1 on how these equations were generated). These equations are used to estimate power consumption at any given clock period (a designer may choose to run the PSI with different clock periods compared to those used to estimating combinational and register power) and with Lagrange, Equation 6.2, the power is estimated at different supply voltages. Figure 6.7 illustrates the way in which combinational and register power equations are used in conjunction with Lagrange power equation.

Two equations were generated for the combinational logic power and two equations for the register power representing power consumption at 1.3V and 1.05V. The combinational logic power equations represent the power consumed by all the combinational logic in the motion compensation and not just that of an individual FE. This eliminates the need to generate a separate equation for each of the 43 FEs in the motion compensation and therefore leads to a substantial saving in run time and complexity. The total power consumption for motion compensation design is the sum of combinational logic power and register power multiplied by total number of registers as shown in Figure 6.7.

Figure 6.8 shows total power consumption when executing the PSI-VS with clock periods ranging from 4800ps to 10400ps. The figure also shows the power consumption of PSI without voltage scaling as a comparison (these results are taken from chapter 5, section 5.5.3). As expected, this figure shows that the PSI-VS generate results for each clock period and for most clock periods they are significantly better than PSI without VS.

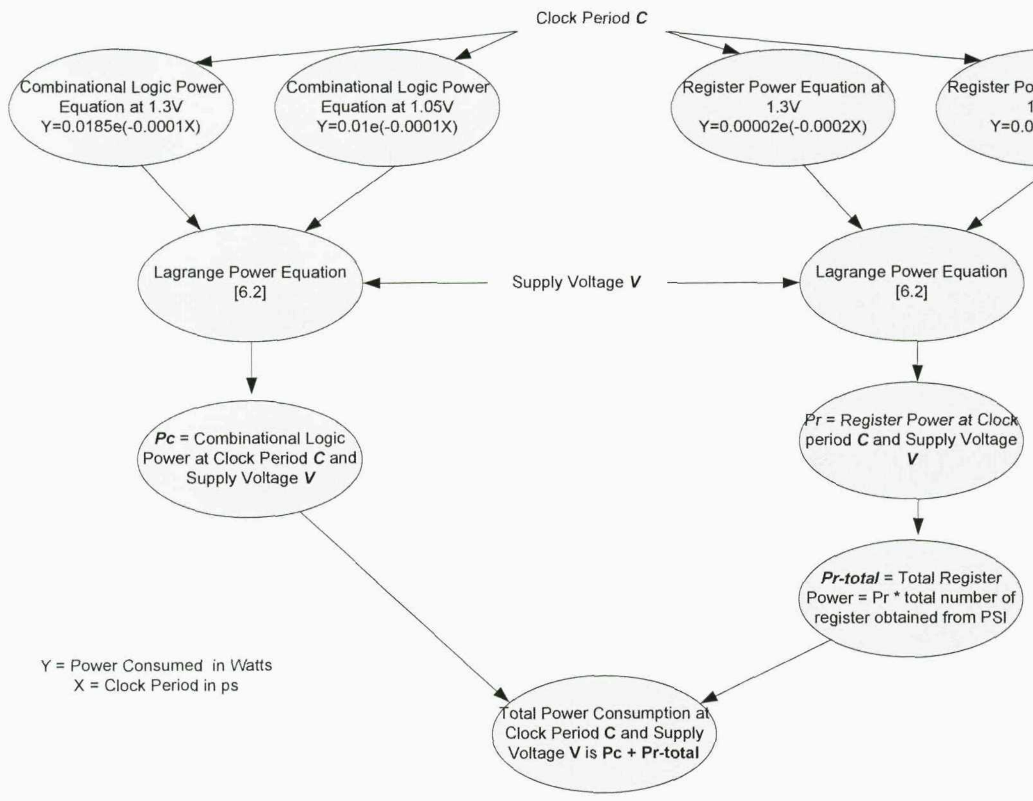

Clock Period C
Combinational Logic Power Equation at 1.3V Y=0.0185e(-0.0001X)
Combinational Logic Power Equation at 1.05V Y=0.01e(-0.0001X)
Register Power Equation at 1.3V Y=0.00002e(-0.0002X)
Register Power Equation at 1.05V Y=0.0183X^(-1)
Lagrange Power Equation [6.2]
Supply Voltage V
Lagrange Power Equation [6.2]
Pc = Combinational Logic Power at Clock Period C and Supply Voltage V
Pr = Register Power at Clock period C and Supply Voltage V
Pr-total = Total Register Power = Pr * total number of register obtained from PSI
Y = Power Consumed in Watts
X = Clock Period in ps
Total Power Consumption at Clock Period C and Supply Voltage V is Pc + Pr-total

Figure 6.7 Inputs/Outputs to/from Lagrange Power Equation

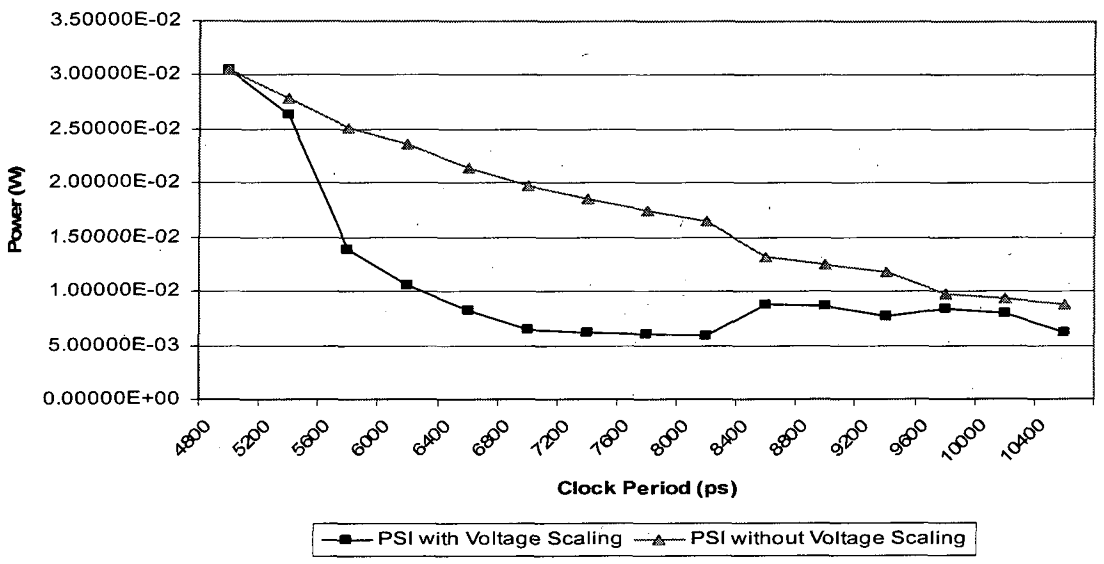


Figure 6.8 Power consumption Comparison between PSI with/without Voltage Scaling

For example, the clock periods between 4800ps to 5200ps show that the PSI-VS solutions consume almost the same power as PSI without VS. This is because the clock period is aggressive with little slack in the stages. In fact, at 4800ps, the slack is 0ps since the clock period is the same as the delay in stage4 (or stage5 as shown in Table 6.3). For the clock periods between 5600ps to 8000ps, significant power saving between 45-68% is observed. It is important to note that such power saving is happening at a high frequency, enabling designers to choose the frequency that suits their requirements with greatly improved power consumption. Clock periods between 8400ps to 10400ps also show power saving in the PSI-VS result; however, this is a moderate saving when compared with the previous clock periods. One reason for this is to do with the PSI results at these periods retaining more of the naive pipeline stages (indicating the greedy nature of PSI algorithm) and therefore more registers. In other words, as the clock period increases, the PSI will find solutions that meet the clock period but with small slack, and is unable to revert to some of the better solutions in the previous clock periods where slack is better. For example, the PSI-VS result at 8400ps is worse than that at 8000ps as a result of the PSI finding a solution at 8400ps that meets timing but has less slack than 8000ps, and therefore the supply voltage could not be reduced to as low a level as that of clock period 8000ps (see Experiment 4 for supply voltage trend).

In summary, this experiment has confirmed (as expected) that using supply voltage as a means to reduce power consumption is a feasible option for PSI. The results indicate significant power saving can be achieved and, more importantly, at high clock frequencies.

6.3.1.2 Experiment 2: Register Count Results

This experiment examines the results generated from PSI-VS (using the same range of input clock periods as those in Experiment 1) for the total register count. Figure 6.9 shows the results from PSI-VS and PSI without VS (results from chapter 5, section 5.5.4) as the clock period is increased from 4800ps to 10400ps. It is interesting to note that the falling trend in the number of registers for the PSI method is not mirrored by the PSI-VS method.

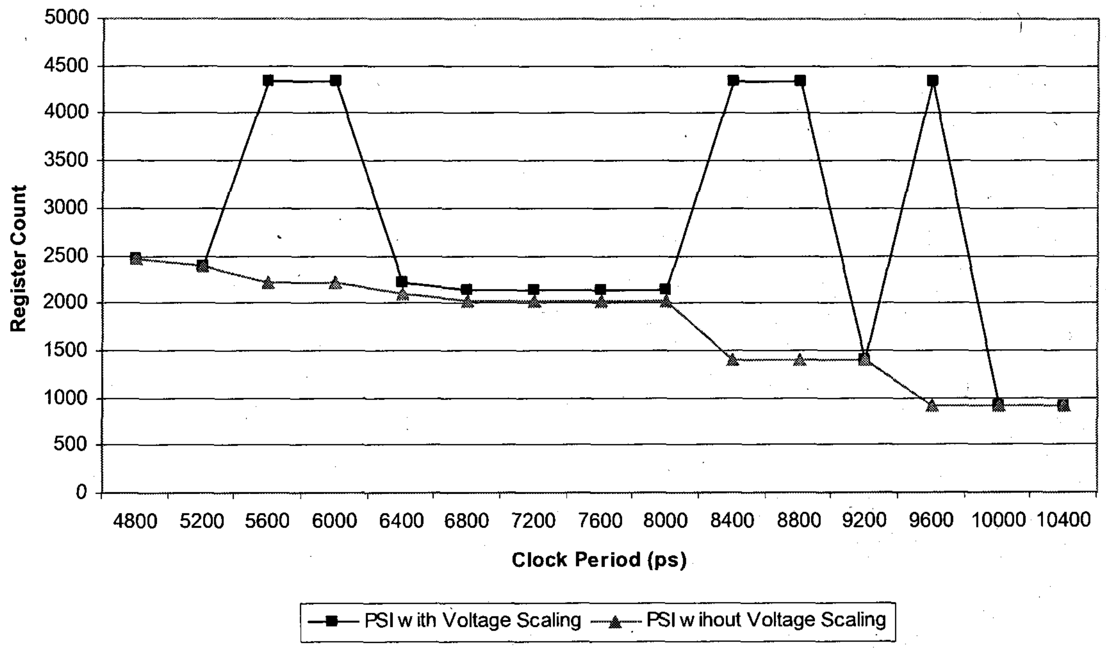


Figure 6.9 Register Count Comparison between PSI with/without Voltage Scaling

In fact, it is observed that at certain clock period (5600ps, 6000ps, 8400ps 8800ps and 9600ps) the total register count is maximum and yet power consumption for PSI-VS is lower than that for PSI without VS. As already mentioned in Experiment 1, slack plays a major part in terms of power reduction since large slack allows the supply voltage to be reduced to such a level that the extra number of registers can be easily offset by the supply voltage and still make further power savings.

In summary, the PSI-VS method changes the characteristics of the register count as a function of the clock period, and whereas one has come to expect that as clock period increases register count decreases, combining voltage scaling changes all that. The final low power result is dependant on the combination of both register count and slack available.

6.3.1.3 Experiment 3: Stage Count Results

This experiment examines the total number of stages for each of the evaluated clock periods (clock periods used in this experiment are the same as those in Experiment 1). Figure 6.10 shows the trend for PSI-VS results and, as can be seen, these resemble the same type of characteristics seen already in Experiment 2 (register count).

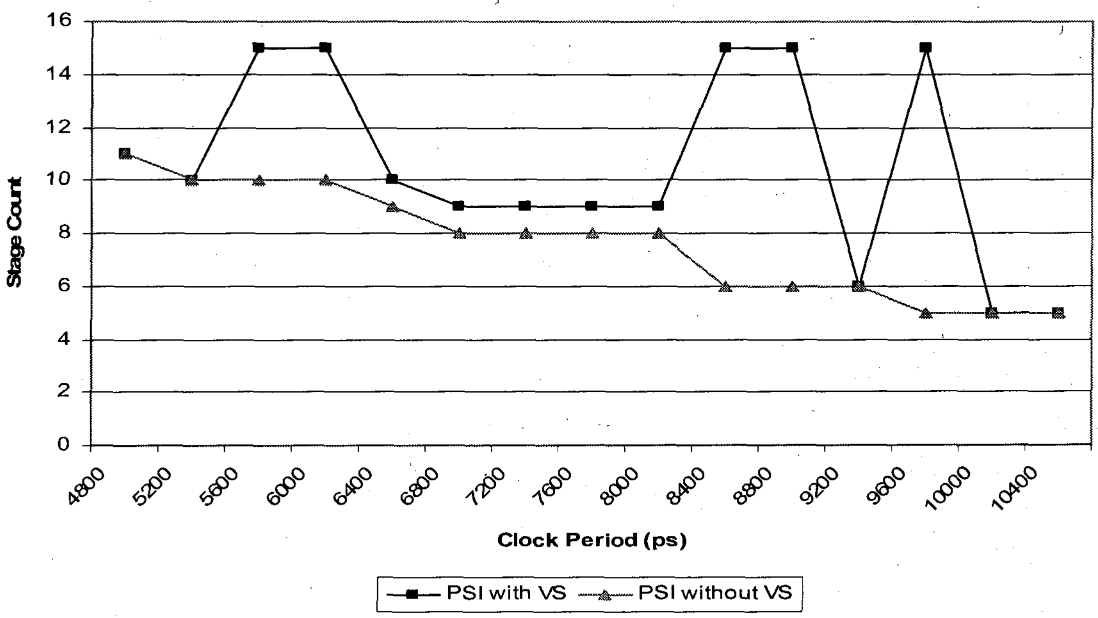


Figure 6.10 Stage Count Comparison between PSI with/without Voltage Scaling

This is to be expected as the increase in the number of registers can only mean an increase in the number of stages. Again, the same observation as in Experiment 2 can be made: voltage scaling as a parameter to the PSI changes the reduction in stage count characteristics as the clock period is increased. This may be an issue for designs that expect low latency, in which case the solution that gives the least latency should be chosen; however, in the context of this thesis, performance is the main goal and latency is of secondary importance.

6.3.1.4 Experiment 4: Supply Voltage Tend

In this experiment for each of the clock periods (clock periods evaluated are the same as in Experiment 1) the supply voltage used to generate the low power solutions is investigated. Figure 6.11 show the trend of supply voltage as the clock period is increased and, as can be seen, a large proportion of clock periods have a solution using the minimum voltage of 0.9V. There are two observations to note, one at the high frequency end of the clock and the second at the low frequency. At the high frequency end (clock periods 4800ps and 5200ps) the clock period is very aggressive and there are few opportunities to remove naive pipeline stages; more importantly, the slack available is not sufficient to reduce the supply voltage and hence it remains close to the maximum (1.3V).

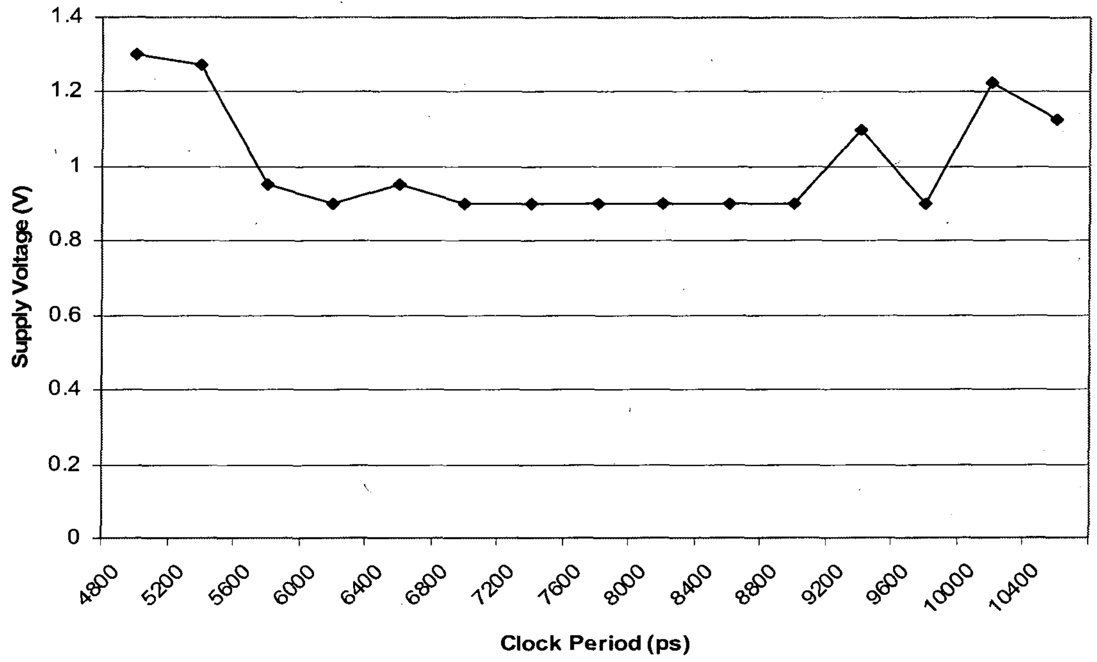


Figure 6.11 PSI-VS Supply Voltage Trend

At the low frequency end (clock periods 9200ps, 10000ps and 10400ps), the supply voltage is generally below 1.2V (with the exception to 10000ps which is just above 1.2V) and power saving is achieved at those periods. What is interesting however is the fact that the voltage at those periods (9200ps, 10000ps and 10400ps) is not close to the minimum voltage (0.9V) as one would expect from such large clock periods (low frequency). This is to do with what has been alluded to in the previous experiments - that there is a link between slack and the total number of registers. For example at 9200ps, the total number of registers for the low power solution was 1396 (Figure 6.9) and the worst slack was 1000ps (Figure 6.13). The 1000ps slack allowed the lowering of the supply voltage from 1.3V to 1.1V (Figure 6.11), and generated the lowest power consuming solution at that period. All the other solutions at 9200ps clock period had the wrong combination of worst slack and register count leading to higher power consumption.

To highlight this point further, Figure 6.12(a) shows the 6-pipeline stages low power solution at 9200ps clock period. In contrast, Figure 6.12(b) shows one of the rejected solutions which had better worst slack (4400ps) than (a) but consumed more power and therefore was an inferior solution to (a). Note that what is shown in Figure 6.12 is the critical path elements with their delays at 1.3V (since worst slack is calculated at the maximum voltage which is in this case is 1.3V). As already mentioned in the previous paragraph, the low power solution in Figure 6.12(a) used a supply voltage of 1.1V and consumed 7.67mW compared with the solution in Figure 6.12(b) operating with the minimum supply voltage of 0.9V, and yet consuming more power at 8.78mW.

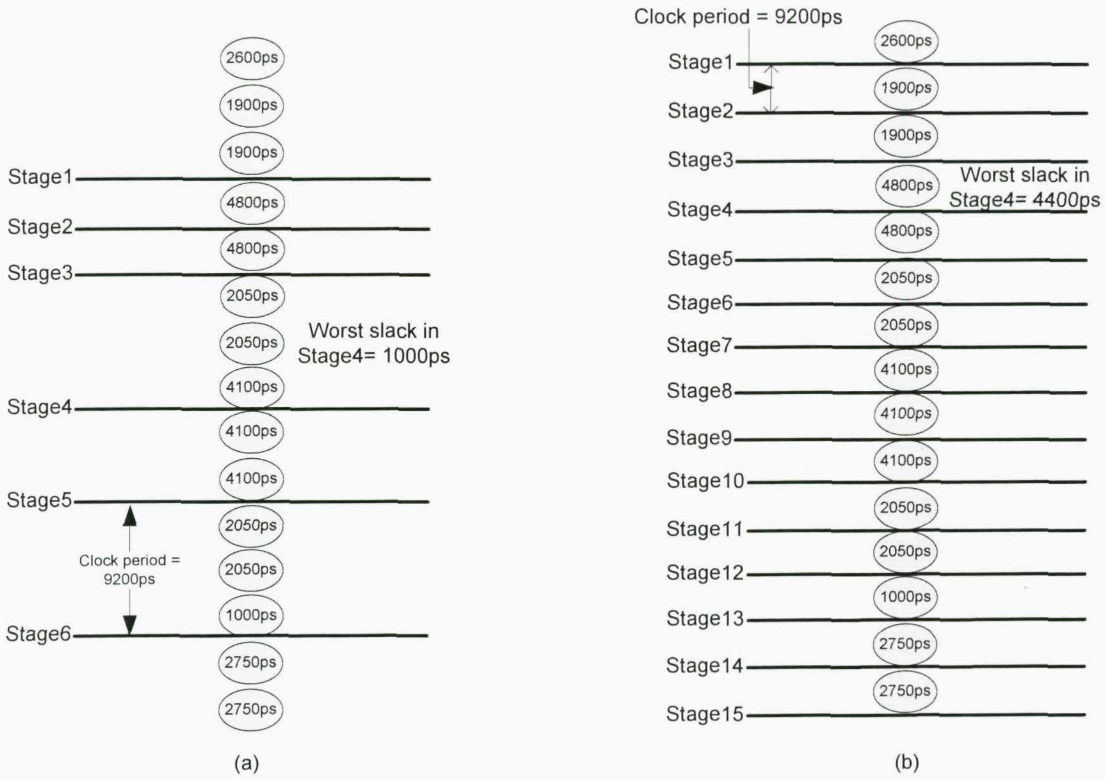


Figure 6.12 Motion Compensation Pipeline Solutions, (a) Low Power Solution Operating at 1.1V, (b) A Rejected Solution with Higher Power Consumption Operating at 0.9V

Therefore, the solution in Figure 6.12(a) was operating at a higher supply voltage while being 12.6% more power efficient than the solution in Figure 6.12(b). It is not surprising that solution (b) is more power hungry than (a) and this is down to the large number of registers implementing the full 15 pipeline stages in the motion compensation design.

Table 6.4 shows the actual critical path elements delay for the solutions in Figure 6.12 at 1.1V and 0.9V. These delays have been obtained using Lagrange delay, Equation 6.1, as part of Step C in Figure 6.2. Consider the worst slack in stage4, Figure 6.12 solution (a), the corresponding FEs involved in this worst slack are the elements 6, 7 and 8 in Table 6.4. Note, these elements in Table 6.4 have delay of 2283ps, 2283ps and 4545ps at 1.1V respectively, giving a total delay for stage4 of 9111ps, which is still within the limit of the clock period of 9200ps used in the solution (a).

Table 6.4 Critical Path Elements Delay at 1.1V and 0.9V for the Solutions in Figure 6.12

Critical path Elements in Figure 6.12	Delay (ps) @ 1.3V	Delay (ps) @ 1.1V	Delay (ps) @ 0.9V
1	2600	3063	3526
2	1900	2282	2663
3	1900	2282	2663
4	4800	5251	5702
5	4800	5251	5702
6	2050	2283	2516
7	2050	2283	2516
8	4100	4545	4989
9	4100	4545	4989
10	4100	4545	4989
11	2050	2283	2516
12	2050	2283	2516
13	1000	1142	1283
14	2750	2996	3241
15	2750	2996	3241

In general, the PSI-VS method was able to find a solution that lowered the supply voltage to its minimum (0.9V) for 53% of the tested clock periods, and 67% if one counts those that were close to the minimum voltage. This is a significant power saving demonstrating the capability of the PSI-VS method. Furthermore, if the PSI-VS method is unable to reduce the supply voltage the corresponding result will not be worse than the PSI without VS, meaning there is a fall back to the original PSI results.

6.3.1.5 Worst Slack Trend

This experiment investigates worst slack trend as the clock period is increased from 4800ps to 104ns. The worst slack reported in this experiment is from the least power consuming solutions generated by the PSI-VS method for each of the tested clock periods (4800ps-

10400ps). Note; the worst slack is calculated using elements delay at 1.3V (as explained in section 6.2) and it is shown in Figure 6.13.

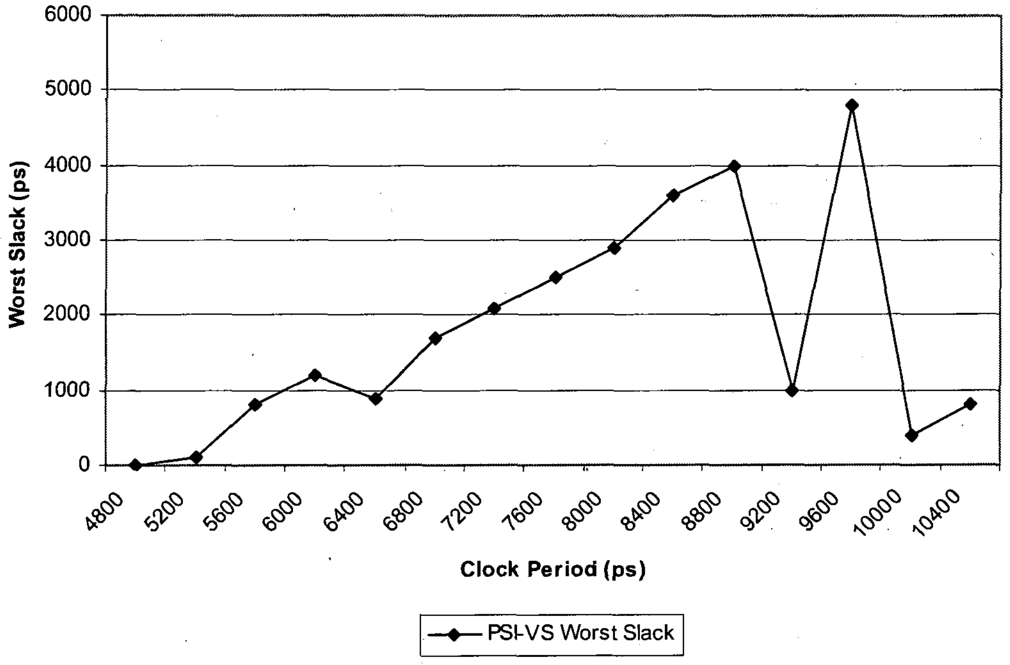


Figure 6.13 PSI-VS Worst Slack at the Low Power Solutions

Figure 6.13 above highlights a few important points. First, at the clock period of 4800ps the worst slack is 0ps. This is because the critical path elements in stage 4 or stage 5 of Table 6.3 show these elements having a delay of 4800ps, which is equivalent to the clock period requirement and hence slack is 0ps. Generally, the worst slack after 4800ps increases as the clock period is increased until 9200ps, where the worst slack declines to 1000ps followed by an increase to almost 5000ps at the clock period of 9600ps, and then tumbles to below 1000ps for the remaining clock periods. The explanation for this sudden decline in the worst slack at 9200ps has already been alluded to in Experiment 4, section 6.3.1.4. At this clock period, the number of stages in the low power solution was six (Figure 6.10), providing a balance between the number of registers implementation the six stages and the 1000ps worst slack. This worst slack translated to lowering the supply voltage to 1.1V (Figure 6.11) which was sufficient to generate the lowest power consuming pipeline solution. This scenario is also taking place at the clock periods of 10000ps and 10400ps and therefore the same explanation can be given.

In summary, the worst slack that contributes to the least power solution will depend on the clock period, the number of stages and the total register count implementing these stages. The larger the number of registers the more the supply voltage has to be lowered, to offset the power consumed in the registers. In order to do that worst slack value needs to be bigger. It may be that even when the worst slack value is so big that the supply voltage can be lowered to its minimum, the pipeline solution would still consume higher power than other solutions due to the excessive number of registers as already shown in Figure 6.12(b).

6.3.1.6 Exhaustive Search with Voltage Scaling

The aim of this experiment is to investigate if the exhaustive search algorithm with voltage scaling could offer better results than PSI-VS in terms of power reduction, performance and run time.

The Exhaustive Search (ES) has already been presented in chapter 4, section 4.3.5.1 and in the context of this experiment, ES has been updated to include support for voltage scaling (VS) (from now on ES with VS will be abbreviated to ES-VS). In similar manner to PSI-VS, the core ES algorithm was kept the same and the only changes done to support VS were made during the result analysis. Each result generated by ES was analysed to determine if voltage scaling could be used to reduce power consumption without affecting performance. The supply voltage used in this experiment ranges from 1.3V to 0.9V similar to PSI-VS (already discussed in section 6.3.1.1). To make the comparison fair, the same range of clock periods (4800ps-10400ps) used in Experiment 1, section 6.3.1.1 was applied to ES-VS. For each clock period, ES-VS generated a number of results which met timing, and those results were individually analysed to determine if supply voltage could be scaled without impacting performance (i.e. clock period under analysis). The result that gives least power consumption at that clock period is saved and the next clock period results are analysed, and from this, a plot of power versus clock periods was generated as shown in Figure 6.14.

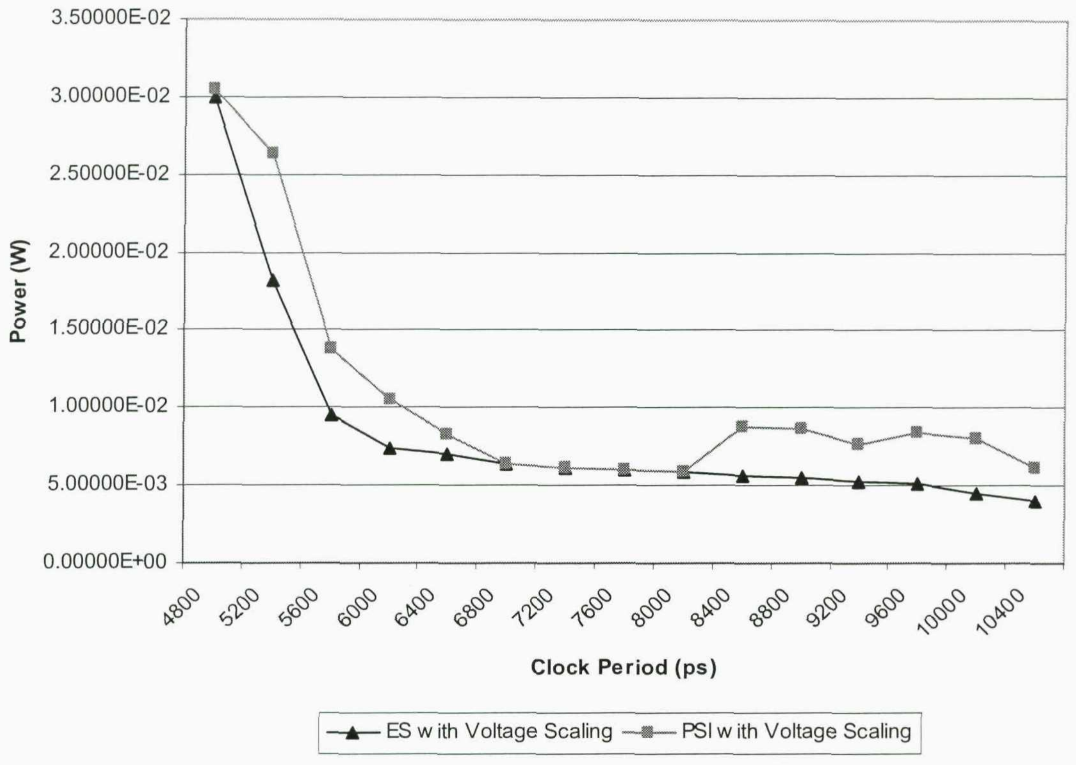


Figure 6.14 Comparison of ES-VS Power Consumption with PSI-VS

Figure 6.14 also shows the PSI-VS power result for those clock periods obtained in Experiment 1. It is clear from this figure that the PSI-VS results do not match those of ES-VS for every clock period. For example, the ES-VS results at clock periods between 5200ps-6400ps and 8400p-10400ps consume less power than PSI-VS. However, for clock periods between 6800ps-8000ps, both PSI-VS and ES-VS generate identical results. The better results for ES-VS mentioned above are not surprising since the ES algorithm is exhaustively analysing all the possible combinations of the naive pipeline stages (see section 4.3.5.1), and therefore is bound to find better results and the best case improvement is seen at the clock period of 5200ps where ES-VS generated a result with 31% less power consumption than PSI-VS. For the clock periods between 8400ps to 10400ps, the PSI-VS result is worse than that for ES-VS. One reason for this is to do with the PSI results at these periods retaining more of the naive pipeline stages (indicating the greedy nature of the PSI algorithm) and therefore more registers. In other words, as the clock period increases, the PSI will find solutions that meet the clock period, but with small slack and is unable to revert back to some of the better solutions in the previous clock periods (e.g. the solution at 8000ps), where slack is better. For example, the PSI-VS result at 8400ps is worse than that at 8000ps, and this is to do with the PSI finding a solution at 8400ps that meets timing but has less slack than 8000ps, and therefore the supply voltage could not be reduced as low as that of the clock period 8000ps.

The second important point is run time implication of ES-VS compared to that of PSI-VS. It is not surprising that ES-VS takes longer to generate the results and for this experiment it took four minutes and 55 seconds compared with PSI-VS which took only 5.83 seconds (using the same 64bit Linux machine with two CPUs running at 2.6GHz and 15G RAM). This is a significant run time improvement; however if run time is not an issue (and generally this is the case for small designs), ES-VS may be the algorithm of choice since it could generate better results for some clock periods. If, on the other hand, the design is complex with many stages, this may mean run time becomes a big factor since (as already seen in Figure 5.14) ES has exponential run time as stages is increased. Under these circumstances – a large number of stages - and if the designer can tolerate the extra power consumption, PSI-VS will be a valid choice since it has linear run time as stages are increased (Figure 5.14). In terms of performance, ES-VS is able to generate solutions for each clock period similar to PSI-VS, and therefore performance is always met for the evaluated clock periods.

In summary, this experiment has shown that ES-VS could generate better results in terms of power reduction than PSI-VS for some of the evaluated clock periods. On the other hand run time could prohibit ES-VS usage for designs that have a large number of stages (or if designers wanted to evaluate a large number of clock periods), and in those cases PSI-VS would be better suited at the expense of moderate increase in power consumption.

6.3.1.7 The Trend of Φ Values for the PSI-VS Solutions

This section presents the values of Φ that resulted in the low power solutions generated by the PSI-VS method. Figure 6.15 shows the trend in Φ values for the different clock periods.

This figure highlights that a number of solutions have used the total 15 stages to generate a low power solution. This is indicated by the value of Φ = 0 which, in Figure 6.15, has been used for five solutions at clock periods of 5600ps, 6000ps, 8400ps, 8800ps and 9600ps. These five solutions indicate the effect of worst slack and voltage scaling on the final solution, and explain why, when comparing Figure 6.15 to Figure 5.15, the trend of Φ values for both figures is completely different. In conclusion, Φ value that leads to the generation of the low power solution will depend not only on the PSI algorithm flags but also on the worst slack and voltage scaling.

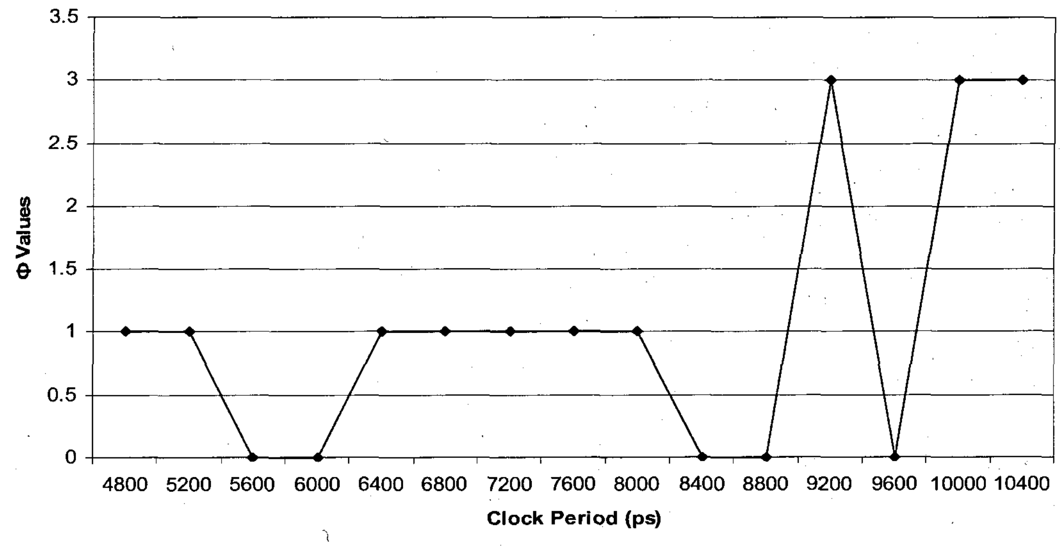


Figure 6.15 Trend of Φ Values for the Different Clock Periods

6.4 Concluding Remarks

This chapter investigated combining voltage scaling (VS) and pipeline stages analysis. The PSI method introduced in chapter 3 was modified to include support for voltage scaling resulting in a new method called PSI-VS. This chapter has shown that further dynamic power consumption reduction in pipeline stages can be achieved by exploiting the slack available in the stage to lower the supply voltage. Experimental results on the synthesisable VHDL implementation of the MPEG-1 motion compensation design and using a 90nm technology library, have shown (Experiment 1) that the PSI-VS solutions consume up to 68% less dynamic power than the PSI without VS. This is a significant power saving demonstrating the capabilities of the PSI-VS method over the PSI without VS.

Experiments 2 and 3 highlighted that the traditional thinking of decreasing register/stage count as the clock period is increased is no longer valid when voltage scaling is combined with the PSI method. This is because the number of stages influences the worst slack in the stage. In general, these experiments show that the more pipeline stages a design has, the bigger the worst slack value and therefore the more the supply voltage can be lowered. However, the larger the number of stages, the more registers are needed to implement these stages, and it may be that lowering the supply voltage to its minimum may not be enough to offset the power consumed in these registers. This leads to a significant conclusion that there is a trade-off between the number of registers in the pipeline stages and the worst slack. This trade-off is exploited fully by the PSI-VS method.

Investigating the supply voltage trend in Experiment 4 for the MPEG-1 motion compensation design shows that the PSI-VS method is able to use the minimum supply voltage of 0.9V for 53% of the tested clock periods, and an overall 67% of the clock periods used a supply voltage close to the minimum. This demonstrates that the PSI-VS method is able to exploit fully the range of supply voltages and use the minimum supply voltage when this gives the least power consuming solution. This experiment also emphasises that at a given clock period, the least power solution does not necessarily have to use the minimum supply voltage. Lowering the supply voltage depends on the worst slack and this in turn is linked to the number of stages and register count as indicated in Experiments 2 and 3.

Experiment 5 investigated the worst slack in the pipeline stages as the clock period is increased. It is observed that the worst slack that contributes to the lowest power pipeline solution will depend on the clock period, the number of stages and the total register count implementing these stages. The larger the number of registers, the more the supply voltage has to be lowered, to offset the power consumed in the registers, and in order to do that the worst slack value needs to be bigger. It is also shown that even when the worst slack value is the largest possible value enabling the supply voltage to be lowered to its minimum, the pipeline solution would still consume higher power than other solutions due to the excessive number of registers. This highlights the dependence of worst slack on the clock period, number of stages and register count, confirming the findings in Experiments 2 and 3.

Finally, in Experiment 6, the exhaustive search (ES) algorithm was updated with voltage scaling (ES-VS), and the experimental results using the motion compensation design have shown that ES-VS can generate better results in terms of power saving than the PSI-VS for some of the evaluated clock periods. On the other hand, the excessive ES-VS run time which, for the motion compensation design took four minutes and 55 seconds compared with only 5.8 seconds for the PSI-VS, could prohibit the ES-VS usage for designs with large number of stages or if the designer evaluated large number of clock periods. In these cases, the PSI-VS method would be better suited because of the significantly small run time at the expense of moderate increase in power consumption.

In summary, combining voltage scaling and the PSI method has been shown to further reduce the power consumption of the pipeline stages while meeting the required clock period.

Chapter 7

Conclusion and Future Work

Power consumption is a major issue for VLSI design and is set to get worse due to technology scaling and increasing device integration unless preventative measures are taken. One of the main drivers for reducing power consumption is the rapid growth in consumer portable electronics. These devices are increasingly running computationally intensive multimedia applications with advanced audio and video coding techniques, which puts a strain on the tight energy budget of the powering batteries. This is of particular concern to the mobile phone industry where keeping customers communicating for longer generate greater revenues for operators and provides handset manufacturers with an edge over their competitors. In addition, there is pressure to develop efficient algorithms and tools to meet the shortening time-to-market and low product cost constraints that are essential for the success of the next-generation products. The work presented in this thesis has focused on minimising the power consumption by analysing pipeline stage insertion in digital designs at the architecture level. To achieve low power pipeline stages, a new algorithm and methods have been developed to generate solutions efficiently, with different merits from which designers can choose the solution that meets their requirements. In particular, three key issues have been addressed. First, power consumption in the pipeline stages at the architecture level has been investigated. Second, a systematic approach to the pipeline stage analysis was developed to generate low power solutions without impacting performance. Finally, voltage scaling was combined with the pipeline stage insertion method to further reduce the power consumption.

The following section 7.1 will summarise the main contributions made by the presented work. Section 7.2 will outline some related areas for future work.

7.1 Summary and Research Contributions

Pipelining digital designs helps to increase throughput and therefore performance, but it also introduces challenges in terms of power consumption. Any optimisation technique to reduce power consumption must meet targeted performance and provide a way to explore power-performance trade-offs.

Chapter 1 highlighted the premise that the earlier in the design cycle the power issue is tackled the more benefit is gained. Architecture level was identified as an appropriate level of the design flow to apply pipeline stage insertion methods due to the presence of specific implementation of FEs at this level. At this level, the optimisation algorithm is aware of the elements delay and outputs which are parameters needed for pipeline stage insertion method.

Chapter 2 extensively reviewed the latest relevant research and identified further need to develop low power pipelining techniques. The previously reported methods were either applied at a lower level of the design flow, such as gate level, with higher cost in terms of implementing the stages efficiently, or did not take into account FEs delay and outputs. Furthermore, the test benches used to validate the previous methods were small to medium size, which does not reflect the complex and large designs common in today's VLSI circuits. In addition, oversimplification of the power estimation for those test benches resulted in overall power consumption that in some cases was unduly optimistic. These problems provided the motivation to find a new technique resulting in the systematic pipeline stage insertion (PSI) method introduced in chapter 3. The PSI method operated at the architecture level guided by a new PSI algorithm that took into account FEs delay and outputs while meeting design performance. The number of outputs in a FE was identified as a critical cost function because it translated directly to registers if the pipeline stage was inserted at the output of that element. The result from the method was a pipeline stages with the least number of registers and hence the least dynamic power consumption that meets the required clock period. An automated flow was proposed to bring together the PSI method and the result analysis, thus enabling designers to experiment with different clock periods helping to explore the power-performance space in an efficient and predictable way. Finally, a brief discussion highlighted how the PSI algorithm could be integrated into high-level synthesis flow to take advantage of the fast time-to-market offered by the latter.

Chapters 4 and 5 applied the PSI method to two real-life complex designs, triple data path floating-point adder (TDPFPADD) and MPEG-1 motion compensation respectively. The designs were coded in structural RTL, and synthesised using common industry tools with 90nm technology library. Accurate power analysis based on widely used industry tools provided realistic power consumption overcoming the shortcomings of the previously reported methods. The results show that the PSI method is able to optimise the pipeline stages to reduce the number of registers while meeting the performance target. For the TDPFPADD case study, the PSI method was able to reduce the dynamic power consumption by 18.1% compared to the naively inserted stages and 24.2% compared to a previously reported approach. For the MPEG-1 motion compensation case study, the power reduction was even higher and a reduction of up to 30% was observed compared to the naively inserted stages. Further experiments were undertaken to compare the PSI method with the exhaustive search (ES) algorithm using the two case studies. The result showed that the PSI generates solutions with power consumption as good as the ES but with significant run time advantage. It was observed that the PSI method had a linear run time complexity compared with exponential time complexity for the ES. This enables the PSI to operate on large complex designs without run time penalty. In chapter 4, it was shown that for 90nm, CMOS standard technology library different physical design layout had limited impact on interconnect length and therefore interconnect capacitance. The experiments also showed that adding more pipeline stages does not reduce interconnect switching power and, more importantly, that internal power is made worse and the latter component of power has a significant impact on overall power consumption. Clock tree power analysis showed that increasing the pipeline stages would increase the number of clock tree buffers and this would lead to the clock tree consuming more interconnect switching power and internal power (consumed within the clock buffers). These results are significant and highlight the care that needs to be taken when adding a large number of pipeline stages. Leakage power is set to get worse as technology continues to scale beyond 100nm and every effort is needed to keep this component of power consumption under control. Reducing the number of cells in the design including reduction in total registers should help in reducing leakage power and extending standby time

Finally, chapter 6 investigated combining voltage scaling (VS) with the PSI method resulting in a new method called PSI-VS. This method further reduced the dynamic power consumption in pipeline stages by utilising the slack available in the stages to lower the supply voltage. Extensive experimental results showed that there is a trade-off between slack and the number of registers implementing the stages. Using MPEG-1 motion compensation as a case study it was observed that the increase in the number of pipeline stages to provide adequate slack may harm the overall power consumption because it increases the register count. Moreover, even when lowering the supply voltage to its minimum, the voltage may still be inadequate to offset the power consumed in the registers. With this in mind, the PSI-VS method searches all the possible solutions to find the one that has the right balance between slack and the number of registers. The work also highlighted that the traditional thinking of decreasing pipeline stages/registers as the clock period is increased is no longer valid when voltage scaling is used. For the MPEG-1 motion compensation case study, the PSI-VS generated solutions with up to 68% less dynamic power compared to the PSI without VS. This is a significant power saving, demonstrating the capabilities of the PSI-VS method over PSI without VS.

In conclusion, the broad aim of this thesis was the development of systematic methods to reduce the dynamic power consumption in pipeline stages at the architecture level. Overall, the work in this thesis has added further evidence to the premise that performing optimisation at the architecture level is very beneficial and, as such, provided the research community with methods that demonstrate this benefit. These methods offer the advantage of being efficient at exploring the power-performance trade-offs in pipeline stages for digital designs. The work has also added further weight to the use of voltage scaling at the architecture level as a mean to reducing power consumption of the pipeline stages. Finally, from an EDA tools point of view the thesis has established the foundation for integrating the PSI methods to a high-level synthesis flow.

7.2 Future Research Directions

There are two main areas of research that could expand the work in this thesis:

- Integration of PSI into high-level synthesis flow
- Leakage Power Reduction

These two areas will be discussed in the following sections.

7.2.1 Integrating PSI Algorithm into High-level Synthesis Flow

High-level synthesis (HLS) is the process of automatically mapping a behavioural description at the architecture level of VLSI design flow to a structural implementation at the register-transfer level in terms of FEs, registers and interconnects. HLS is becoming more important in today's aggressive time-to-market schedule because it is an automated and fast flow. In the past decade, the research community has generated a substantial body of work on HLS low power optimisation techniques, including voltage scaling and switched capacitance reduction [8, 12-18, 70-72, 102, 104-113]. As a starting point for this task, one could use an already available and tested high-level synthesis suite such as MOODS [31] to be the initial platform for integration. This should speed up the integration and make available to the PSI algorithm functions and procedures that have already been written and tested. There are a number of challenges that need to be solved, and at the forefront is how to get the HLS flow to generate the naive pipeline stages. This will involve identifying appropriate scheduling and data path synthesis algorithms capable of generating designs optimised for dynamic power reduction through the PSI, but also with reduced area overhead [90, 132-135].

7.2.2 Leakage Power Reduction

Leakage power in deep submicron designs is becoming an increasing component of overall power [40-43]. Minimising this component of power is essential especially in portable electronic devices as it has direct impact on standby power and thus battery life. The trend is set to get worse with technology scaling unless effective techniques are introduced to bring leakage under control. Therefore, it would be particularly interesting to investigate the incorporation of leakage power into the PSI algorithm as a cost function. For example, it is well known that leakage current of a gate is a strong function of its input values. This is because the inputs affect the number of off transistors in the NMOS and PMOS networks of logic gates. Take for example a 2-input NAND gate - it is shown that when both inputs are at logic value of zero, the least leakage current is achieved [90]. Putting this in the PSI algorithm context, when examining the naive stages the algorithm should also examine the influence on the logic driven by the pipeline stage registers. If a pipeline stage register could be used to drive a value into the combinational logic, such that the overall logic in the stage produces lower leakage current, then the stage should inserted at that point. It may be that the registers would have to be reset, and the reset value is used to drive the logic when the design will be in idle mode for a reasonably long length of time. Another alternative method would be to use the DFT scan chain to shift in values sequentially to the registers. This method may be rewarding for a substantially longer idle time as the shifting of values would undoubtedly have an impact on switching power and the idle time must try to offset this power.

References

[1] Massoud Pedram, “Power Minimization in IC Design: Principles and Applications”, ACM Transaction on Design Automation of Electronic Systems, Vol. 1, No. 1, January 1996, Pages 3-56

[2] Wolfgang Nebel, Jean Mermet, “Low Power Design in Deep Submicron Electronics”, Kluwer Academic Publishers, 1997

[3] Richard G. Lyons, “Understanding Digital Signal Processing”, Addison-Wesley, 1997

[4] Behrooz Parhami, “Computer Arithmetic Algorithms and Hardware Designs”, Oxford University Press, 2000

[5] “IEEE Standard for Binary Floating-Point Arithmetic”, ANSI/IEEE Std, 754-1985, Aug 1985

[6] J. Y. F. Tong and D. Nagle, “Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic”, IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 8, No.3, June 2000, pp. 273-286

[7] Jan M. Rabaey and Massoud Pedram, “Low Power Design Methodologies”, Kluwer Academic Publishers, 2002

[8] L.Y. Chiou, K. Muhammad and K. Roy, “DSP Data path Synthesis for Low-Power Applications”, In Proc. In IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 2, May 2001, pp. 1165-1168

[9] M. Ercegovac, D. Kirovski, G. Mustafa and M. Potkonjak, “Behavioural Synthesis Optimisation Using Multiple Precision Arithmetic”, In Proc. In IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 5, 12-15 May 1998, pp. 3113-3116

[10] M. C. Molina, J. M. Mendias and R. Hermida, “High-Level Synthesis of Multiple-Precision Circuits Independent of Data-Objects Length”, In Proc. In Design Automation Conference, June 10-14 2002, pp. 612-615

[11] G. A. Constantinides, P. Y. K. Cheung and W. Luk, “Optimal Data path Allocation for Multiple-Word length Systems”, IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 13, Issue 1, Jan 2005, pp. 39-57

[12] M. Ercegovac, D. Kirovski and M. Potkonjak, “Low-Power Behavioural Synthesis Optimisation Using Multiple Precision Arithmetic”, In Proc. In 36th Design Automation Conference, 21-25 June 1999, pp. 568-573

[13] M. C. Molina, J. M. Mendias, R. Hermida, “High-Level Allocation to Minimise Internal Hardware Wastage”, In Proc. In Conference and Exhibition on Design, Automation and Test in Europe, 2003, pp. 264-269

[14] M. C. Molina, R. R. Sautua, J. M. Mendias and R. Hermida, “Bit-Level Allocation for Low Power in Behavioural High-Level Synthesis”, In Proc. In Power and Timing Modelling, Optimization and Simulation, 13th International Workshop, PATMOS 2003, Torino, Italy, Sept 10-12 2003, pp. 617-627

[15] M. S. Bright and T. Arslan, "Synthesis of Low-Power DSP Systems Using a Genetic Algorithm", In IEEE Transaction on Evolutionary Computation, Vol. 5 Issue 1, February 2001, pp. 27-40

[16] R. S. Martin and J. P. Knight, "Power-Profiler: Optimising ASICs Power Consumption at the Behavioural Level", In Proceedings of the 32nd Design Automation Conference, June 1995, pp. 42-47

[17] W.T. Shiue and C. Chakrabarti, "ILP-Based Scheme for Low Power Scheduling and Resource Binding", In IEEE International Symposium on Circuits and Systems, Geneva, 28-31 May 2000, pp. 279-282

[18] C. Park, T. Kim and C. L. Liu, "An Efficient Data Path Synthesis Algorithm for Behavioural-Level Power Optimisation", In Proc. In IEEE Inter. Symp. On Circuits and Systems, Vol. 1, 30 May-2 June 1999, pp. 294-297

[19] G. A. Constantinides, P. Y. K. Cheung and W. Luk, "Heuristic Datapath Allocation for Multiple Wordlength Systems", In Proc. Of Design, Automation and Test in Europe, Munich Germany, 2001, pp. 791-797

[20] F. Fang, T. Chen and R. A. Rutenbar, "Floating-Point Bit-Width Optimisation for Low-Power Signal Processing Applications", In Proc. In IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 3208-3211, May 2002

[21] A. Clements, "The Principles of Computer Hardware", Oxford University Press, 1991

[22] J. M. Rabaey, "Digital Integrated Circuits", Prentice Hall Electronics and VLSI Series, Prentice-Hall Inc., 1996

[23] K. C. Chang, “Digital Systems Design with VHDL and Synthesis”, The Institute of Electrical and Electronics Engineers, Inc., 1999

[24] S.C. Wong, G.Y. Lee and D.J. Ma, “Modelling of Interconnect Capacitance, Delay and Crosstalk in VLSI”, IEEE Transaction on Semiconductor Manufacturing, Vol. 13, Issue. 1, February 2000, pp. 108-111

[25] L. Thede, “Analogue and Digital Filter Design Using C”, Prentice Hall

[26] C. B. Rorabaugh, “DSP Primer”, McGraw-Hill, 1999

[27] L. B. Jackson, “Digital Filters and Signal Processing”, Second Addition, Kluwer Academic Publishers, 1993

[28] D. Hanselman and B. Littlefield, “Mastering Matlab 6: A Comprehensive Tutorial and Reference”, Prentice Hall, Inc., 2001

[29] S. Ogg, L. Gutierrez, K. Loupos and A. Moustafa, “Arithmetic Circuit Analysis and Design”, University of Southampton, Electronic and Computer Science Department, March 24 2003, MSc Thesis

[30] R. Burch, F. Najm, P. Yang and T.N. Trick, “A Monte Carlo Approach for Power Estimation”, IEEE Transaction on VLSI Systems, Vol. 1, No. 11, pp. 63-71 March 1993

[31] A. C. Williams, A. D. Brown, and M. Zwolinski, “Simultaneous Optimisation of Dynamic Power, Area and Delay in Behavioural Synthesis”, IEE Proceedings on Computers and Digital Techniques, Vol. 147, No. 6, November 2000, p. 383-390

[32] www.synopsys.com

[33] www.magma-da.com

[34] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski and P. G. Emma "Integrated Analysis of Power and Performance for Pipelined Microprocessors", IEEE Transaction on Computers, Vol. 53, Issue 8, Aug. 2004, pp. 1004-1016

[35] V. Srinivasan, V. Zyuban, D. Brooks, M. Gschwind, P. Bose, P. N. Strenski and P. G. Emma "Optimising Pipelines for Power and Performance", In Proc. of 35th Annual IEEE/ACM International Symposium on Microarchitecture, Nov. 2002, PP. 333-344

[36] S. Heo and K. Asanovic, "Power-Optimal Pipelining in Deep Submicron Technology" In Proc. Of Inter. Symp. On Low Power Electronics and Design, August 9-11, 2004, pp. 218-223

[37] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "A Low Power Approach to Floating Point Adder Design" In Proc. In International Conference on Computer Design: VLSI in Computers and Processors, 12-15 Oct. 1997, pp. 178-185

[38] R. V. K. Pillai, "On Low Power Floating Point Data Path Architecture" PhD. Thesis, Concordia University, Montreal, Quebec, Canada, April 1999

[39] J. L. Hennessy and D. A. Patterson, "Computer Architecture A Quantitive Approach" Morgan Kaufmann Publishers Inc., 1990.

[40] J. H. Anderson and F. N. Najm, "Active Leakage Power Optimisation for FPGAs", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 3, March 2006, pp. 423-437.

[41] M. L. Mui, K. Banerjee and A. Mehrotra, “Supply and Power Optimization in Leakage-Dominant Technologies”, IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, No. 9, September 2005, pp. 1362-1371

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

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

[44] J. Di and J. S. Yuan, “Power-aware pipelined multiplier design based on 2-dimensional pipeline gating”, Proceedings of the 13th ACM Great Lakes symposium on VLSI 2003, Washington, D. C., USA April 28 - 29, 2003, pp. 64-67

[45] S. Parameswaran and H. Guo, “Power Reduction in Pipelines”, Design Automation Conference 1998, Proceedings of the ASP-DAC ’98 Asia and South Pacific, m10-13 Feb. 1998, pp. 545-550

[46] M. Olivieri and M. Raspa, “Power Efficiency of Application-Dependent Self-Configuring Pipeline Depth in DSP Microprocessors”, In Proc. In Symposium on Parallel and Distributed Processing, 22-26 April 2003, pp. 185.1

[47] M. Olivieri, “Theoretical System-Level Limits of Power Dissipation Reduction Under a Performance Constraint in VLSI Microprocessor Design”, IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 10, No. 5, Oct. 2002, pp. 595-600

[48] R. V. K. Pillai, D. Al-Khalili, A. J. Al-Khalili and S. Y.A. Shah, “A Low Power Approach to Floating Point Adder Design for DSP Applications”, Journal of VLSI Signal Processing, Vol. 27, Issue 3, March 2001, pp. 195-213

[49] P. M. Seidel and G. Even, "Delay-Optimized Implementation of IEEE Floating-Point Addition", IEEE Transaction on computers, Vol. 53, No. 2, Feb. 2004, pp. 97-113

[50] W. Hokenek, R. K. Montoye and P. W. Cook, "Second-Generation RISC Floating Point with Multiply-Add Fused", IEEE Journal of Solid-State Circuits, Vol. 25, No. 5, Oct. 1990, pp. 1207-1213

[51] T. Lang and J. D. Bruguera, "Floating-Point Fused Multiply-Add with Reduced Latency", In Proc. In IEEE International Conference on Computer Design: VLSI in computers and Processors, 16-18 Sept. 2002, pp. 145-150

[52] M. Uya, K. Kaneko and J. Yasui, "A CMOS Floating Point Multiplier", IEEE Journal of Solid-State Circuits, Vol. 19, No. 5 Oct. 1984, pp. 697-702

[53] G. Even and W. J. Paul, "On the Design of IEEE Compliant Floating Point Units", IEEE Transactions on Computers, Vol. 49, No. 5, May 2000, pp.398-413

[54] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", ACM Computing Surveys, Vol. 23, No. 1, March 1991, pp. 5-48

[55] R. K. Yu and G. B. Zyner, "167 MHz Radix-4 Floating Point Multiplier", In Proc. In Symposium on Computer Arithmetic, 19-21 July 1995, pp. 149-154

[56] F. Fang, T. Chen, R. Rutenbar, "Lightweight Floating Point Arithmetic: Case Study of Inverse Discrete Cosine Transform", EURASIP Journal on Signal Processing, Special Issue on Applied Implementation of DSP and Communication Systems, Sept 2002, pp. 879-892

[57] R. V. K. Pillai, S. Y. A. Shah, A. J. Al-Khalili and D. Al-Khalili, "Low Power Floating Point MAFs – A Comparative Study", In Proc. In International Symposium on signal Processing and its Applications, Malaysia 13-16 August 2001, pp. 284-287

[58] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance" In Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA), pages 13-22, January 1999

[59] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Power Implications of Precision Limited Arithmetic in Floating Point FIR Filters", In Proc. In International Symposium on Circuits and Systems, Vol. 1, July 1999, pp. 165-168

[60] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili , "Power Implications of Additions in Floating Point DSP – an Architectural Perspective", In Proc. In AFRICON Conference, Vol. 1, 28 Sept – 1 Oct 1999, pp. 581-586

[61] J. Costello and D. Al-Khalili, "Behavioural Synthesis of Low Power Floating Point CORDIC Processors", In Proc. In IEEE International Conference on Electronics, Circuits and Systems, Vol. 1, 17-20 Dec. 2000 pp. 506-509

[62] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "A Low Power floating Point Accumulator", In Proc. In International Conference on VLSI Design, 4-7 Jan. 1998, pp. 330-333

[63] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "On the Power Implications of Floating Point Addition in IIR Filters", In Proc. In International Conference on Microelectronics, 14-16 Dec 1998, pp. 200-203

[64] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Low Power Architecture for Floating Point MAC Fusion", IEE Proceedings - Computers and Digital Techniques, July 2000, Vol. 147, Issue 4, pp. 288-296

[65] K. E. Wires, M. J. Schulte and J. E. Stine, "Combined IEEE Compliant and Truncated Floating Point Multipliers for Reduced Power Dissipation", In Proc. In International Conference on Computer Design, 23-26 Sept 2001, pp. 497-500

[66] S. Y. A. Shah, "On Synthesis and Optimisation of Floating Point Unit", MSc Thesis, Concordia University, Montreal, Quebec, Canada, Oct. 2000.

[67] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko and T. Sumi, "Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition", IEEE Journal of Solid State Circuits, Vol. 31, No. 8, 1996, pp. 1157-1164

[68] T. Wu and Y. Lin, "Storage Optimization by Replacing some Flip-Flops with Latches", In Proc. In Design Automation Conference, 16-20 Sept. 1996, pp. 296-301

[69] T. M. Burks, K. A. Skallah and T. N. Mudge, "Critical Paths in Circuits with Level-Sensitive Latches", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, Issue 12, Dec. 1995, pp. 1526-1545

[70] D. D. Gajski and L. Ramachandran, "Introduction to High-Level Synthesis", IEEE Design and Test of Computers, Vol. 11, Issue: 4, Winter 1994, pp. 44-54

[71] R. A. Walker and S. Chaudhuri, "Introduction to scheduling problem", Design and Test of Computers, IEEE, Vol. 12, Issue 2, Summer 1995, pp. 60-69

[72] M. Ochoa-Montiel, B. M. Al-Hashimi and P. Kollig, "Impact of Multicycled Scheduling on Power-Area Trade-offs in Behavioural Synthesis", IEEE International Symposium on Circuits and Systems, Vol. 4, ISCAS 23-26 May 2005, pp. 4163-4166

[73] D. Harris and I. Sutherland, "Logical Effort of Carry Propagate Adders", In Proc. In Conference on Signals, Systems and Computers, Vol. 1, 9-12 Nov. 2003, pp. 873-878

[74] O. J. Bedrij, "Carry-Select Adder", IRE Transactions on Electronic Computers, Vol. EC-11, June 1962, pp. 340-346

[75] M. S. Schmookler and K. J. Nowka, "Leading Zero Anticipation and Detection – A Comparison of Methods", In Proc. In IEEE Symp. On Computer Arithmetic, 11-13 June 2001, pp. 7-12

[76] V. G. Oklobdzija, B. R. Zeydel, H. Dao, S. Mathew and R. Krishnamurthy, "Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders", In IEEE Symp. On Computer Arithmetic, 15-18 June 2003, pp. 272-279

[77] H. Dao, V. G. Oklobdzija, "Application of Logical Effort Techniques for Speed Optimisation and Analysis of Representative Adders", In IEEE Conference on Signals, Systems and Computers, Vol. 2, 4-7 Nov. 2001, pp. 1666-1669

[78] M. R. Santoro, G. Bewick and M. A. Horowitz, "Rounding Algorithms for IEEE Multipliers", In IEEE Proc. In Symp. On Computer Arithmetic, 6-8 Sept. 1989, pp. 176-183

[79] J. Sklansky, "Conditional Sum Addition Logic," IRE Trans. Electronic Computing, Vol. EC-9, No. 6, June 1960, pp. 226-231

[80] N. T. Quach and M. J. Flynn, "An Improved Algorithm for High-Speed Floating-Point Addition", Technical Report CSL-TR-90-442, August 1990, http://arith.stanford.edu/techrep.html

[81] N. Quach, N. Takagi and M. Flynn, "On Fast IEEE Rounding", Technical Report CSL-TR-91-459, January 1991, http://arith.stanford.edu/techrep.html

[82] D. Radhakrishnan, "Low-Voltage Low-Power CMOS Full Adder", In IEEE Proc. On Circuits, Devices and Systems, Vol. 148, Issue 1, Feb. 2001, pp. 19-24

[83] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, “Evaluation of 1’s Complement Arithmetic for the Implementation Low Power CMOS Floating Point Adders”, In Proc. In IEEE Conf. On Electrical and Computer Engineering, Vol. 1, 25-28 May 1997, pp. 153-156

[84] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, “Energy Delay Measures of Barrel Switch Architectures for Pre-Alignment of Floating Point Operands for Addition”, In IEEE Inter. Symp. On Low Power Electronics and Design, 18-20 Aug 1997, pp. 235-238

[85] G. Zhang, Z. Qi and W. Hu, “A Novel Design Of Leading Zero Anticipation Circuit With Parallel Error Detection”, In IEEE Inter. Symp. on Circuits and Systems, Vol. 1, 23-26 May 2005, pp 676-679

[86] X. Wu and M. Pedram, “Low Power Sequential Circuit Design Using Priority Encoding and Clock Gating”, In Proc. of Inter. Symp. on Low Power Electronics and Design, 2000, pp. 143-148

[87] A. Raghunathan, S. Dey, N. K. Jha, “Glitch Analysis and Reduction in Register Transfer Level Power Optimisation”, In Proc. of Design Automation Conference, 3-7 June 1996, pp. 331-336

[88] T. Lang, E. Musoll and J. Cortadella, “Individual Flip-Flops with Gated Clocks for Low Power Datapaths”, In IEEE Transactions on Circuits and Systems, Vol. 44, Issue 6, June 1997, pp. 507-516

[89] B. B. Welch, “Practical Programming in Tcl and TK”, Third Edition, 1997, Prentice Hall PTR

[90] B. M. Al-Hashimi, “System-on-Chip: Next Generation Electronics”, The Institution of Electrical Engineers, IEE Circuits, Devices and Systems Series 18, 2006

[91] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Circuitry by Retiming" 3rd Caltech Conference on VLSI, 1981, pp. 87-116

[92] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R. Brodersen, "Optimizing Power Using Transformations", IEEE Trans. on CAD of ICs & Syst., Vol. 14, No. 1, pp 12–31, 1995

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

[94] A. Bovik, "Handbook of Image and Video Processing", Second Edition, 2005, Academic Press

[95] I. E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia", 2003, Wiley

[96] I. E. G. Richardson, "Video Codec Design", 2002, Wiley

[97] Moving Picture Experts Group (MPEG), http://www.chiariglione.org/mpeg

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

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

[100] K. M. Patel, B. C. Smith and L. A. Rowe, "The Berkeley software MPEG-1 video decoder" In ACM Transactions on Multimedia, Computing, Communication and Applications (TOMCCAP), Vol. 1, Issue 1, February 2005, pp. 110-125

[101] T. Akenine-Moller and E. Haines, "Real-Time Rendering", 2nd Edition, 2002, A.K. Peters

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

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

[104] A. C. Williams, "A Behavioural VHDL Synthesis System Using Data Path Optimisation", PhD Thesis, University of Southampton 1997

[105] J. Bhasker, "A VHDL Primer", Prentice Hall 1999

[106] G. Economakos, P. Oikonomakos, I. Poulakis, I. Panagopoulos, G. Papakonstantinou "Behavioural Synthesis with SystemC", Design Automation and Test in Europe (DATE), 2001, pp. 21-25

[107] G. De Micheli, "Synthesis and Optimisation of Digital Circuits", McGraw – Hill 1994

[108] P. Babighian, L. Benini and E. Macii, "A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation", In IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, Vol. 24 No. 1, Jan. 2005, pp. 29-42

[109] P. Oikonomakos, "High-level Synthesis for On-line Testability", PhD Thesis, University of Southampton, 2004

[110] R. Henning and C. Chakrabarti "An Approach to Switching Activity Consideration During High-Level, Low-Power Design Space Exploration", In IEEE Trans. On Circuit and Systems II: Analogue and Digital Signal Processing", Vol. 49, No. 5, May 2002, pp. 339-351

[111] C. Lyuh, T. Kim and C. L. Liu, "An integrated Data Path Optimisation for Low Power Based on Network Flow Method", IEEE/ACM International Conference on Computer Aided Design, 4-8 Nov. 2001, pp. 553-559

[112] S. Hong and T. Kim, "Bus Optimisation for Low-Power Data Path Synthesis based on Network Flow Method", ICCAD, 2000

[113] K. K. Parhi, "Algorithm transformation technique for concurrent processors", Proceedings of the IEEE, Vol. 77, Issue 12, Dec 1989, pp. 1879-1895

[114] P. Ackland and P. D'Arcy, "A new generation of DSP architectures" in Proc. of IEEE CICC, May 1999

[115] D. Kim, D. Shin and K. Choi, "Pipelining With Common Operands for Power-Efficient Linear Systems", IEEE Trans. On VLSI Systems, Vol. 13, No. 9, Sept. 2005, pp. 1023-1034

[116] D. Kim, and K. Choi, "Power-conscious high-level synthesis using loop folding", in Proc. 34th ACM/IEEE Design Automation Conf., June 1997, pp. 441-445

[117] H. Q. Dao, B. R. Zeydel and V. G. Oklobdzija, "Energy Optimisation of Pipelined Digital Systems Using Circuit Sizing and Supply Scaling", IEEE Trans. On VLSI Systems, Vol. 14, No.2 Feb. 2006, pp. 122-134

[118] D. Harris, R. F. Sproull and I. E. Sutherland, "Logical Effort: Designing Fast CMOS Circuit", San Mateo, CA: Morgan Kaufmann 1999

[119] H. Q. Dao, B. R. Zeydel and V. G. Oklobdzija, "Energy minimisation method for optimal energy-delay extraction", presented at the European Solid-State Circuits Conf. Estoril, Portugal Sep. 2003

[120] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski and P. G. Emma, “Optimizing Pipelines for Power and Performance”, Proc. 35th Annual IEEE/ACM International Symposium on Microarchitecture, 18-22 Nov. 2002, pp. 333-344

[121] B. Koo, S. Kim, S. Lee, M. Choi, K. Park N. Eum, J. Kim and H. Cho, “A Pipelined Low Power Architectural MPEG-4 Video Codec Chip with Deblocking Filter for Mobile Wireless Multimedia”, Proc. 5th International Conference on ASIC, Vol. 2, 21-24 Oct 2003, pp. 934-937

[122] H. Shimada, H. Ando, T. Shimada, “Pipeline Stage Unification: A Low-Energy Consumption Technique for Future Mobile Processors”, Proc. of the 2003 International Symposium on Low Power Electronics and Design, 25-27 Aug 2003, pp. 326-329

[123] A. Panato, S. Silva, F. Wagner, M. Johann, R. Reis and S. Bampi, “Design of Very Deep Pipelined Multipliers for FPGAs”, Proc. of Design Automation and Test in Europe Conference and Exhibition, Vol. 3, 16-20 Feb. 2004, pp. 52-57

[124] A. Garcia, W. Burleson, J. L. Danger, « Low Power Digital Design in FPGAs : A Study of Pipeline Architectures implemented in a FPGA using a low supply voltage to reduce power consumption”, The IEEE Inter. Symp. On Circuits and Systems, Vol.5, 28-31 May 2000, pp. 561-564

[125] A. Hartstein and T. R. Puzak, “Optimum Power/Performance Pipeline Depth”, Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003, pp. 117-125

[126] J. Di and J. S. Yuan, “Power-aware Pipelined Multiplier Design Based On 2-Dimentional Pipeline Gating”, Proc. pf the 13th ACM Great Lakes Symp. On VLSI, 2003, pp. 64-67

[127] M. Olivieri and M. Raspa, "Power Efficiency of Application-Dependant Self-Configuring Pipeline Depth in DSP Microprocessors", Proc. of Inter. Parallel and Distributed Processing Symposium, 22-26 April 2003, pp 1-7

[128] S. Heo and K. Asanovic, "Power-Optimal Pipelining in Deep Submicron Technology", Proc. of the 2004 Inter. Symp. On Low power electronics and design, 9-11 Aug. 2004, pp. 218-223

[129] A. Efthymiou, J. D. Garside and I. Papaefstathiou, "A Low-Power Processor Architecture Optimized for Wireless Devices", Proc. of 16th IEEE Inter. Conference on Application-Specific Systems, Architecture Processors, 23-25 July 2005, pp. 185-190

[130] D. I. Lazorenko, A. A. Chemeris, "Low-power issues for soc", Proc. of the 2006 IEEE 10th Inter. Symp. On Consumer Electronics, 28-01 June/July 2006, pp. 1-3

[131] H. Ali and B. M. Al-Hashimi, "Architecture Level Power-Performance Trade-offs for Pipelined Designs", IEEE International Symposium on Circuits and Systems, 27-30 May 2007 New Orleans, pp. 1791-1794

[132] D. Gajski, N. Dutt, A. Wu and S. Lin, "High level synthesis", Kluwer Academic Publisher, New York, 1993

[133] R. A. Walker, S. Chaudhuri, "Introduction to the scheduling problem", Design & Test of Computers, IEEE Vol. 12, Issue 2, summer 1995, pp. 60-69

[134] H. C. Yang and L. R. Dung, "On Multiple-voltage High-level Synthesis Using Algorithmic Transformations", Proceedings of the Asia and South Pacific Design Automation Conference, Vol. 2, 18-21 Jan. 2005 pp. 872-876

[135] S. Koziel, W. Szczesniak, “Application of Adaptive Evolutionary Algorithm for Low Power Design of CMOS Digital Circuits”, 9th International Conference on Electronics, Circuits and Systems, Vol. 2, 15-18 Sept 2002, pp. 685-688

[136] K. Roy, H. Mahmoodi, S. Mukhopadhyay, H. Ananthan, A. Bansal and T. Cakici, “Double-Gate SOI Devices for Low-Power and High-Performance Applications”, Proc. of the 2005 IEEE/ACM International Conference on Computer Aided Design, San Jose, USA, 2005, pp. 217-224

[137] J. C. Chi, H. H. Lee, S. H. Tsai and M. C. Chi, “Gate Level Multiple Supply Voltage Assignment Algorithm for Power Optimization Under Timing Constraint”, IEEE Trans. On Very Large Scale Integration (VLSI) Systems, Vol. 15 Issue 6, June 2007, pp. 637-648

[138] Marcus T. Schmitz, Bashir M. Al-Hashimi and Petru Eles, “System-level Design Techniques for Energy-efficient Embedded Systems”, Kluwer Academic Publishers, Dec 2003

[139] C. Yoon, J. Kook, R. Woo, S. Lee, K. Lee and H. Yoo, “Low Power Motion Compensation Block IP with embedded DRAM Macro for Portable Multimedia Applications”, 2001 Symp. On VLSI Circuits, 14-16 June, 2001, pp.99-102

[140] C. Chien, H. Chen, L. Huang and L. Guo, “A Low-Power Motion Compensation IP Core Design for MPEG-1/2/4 video decoding”, IEEE Int. Symp. On Circuits and Systems 2005, 23-26 May 2005, Vol. 5, pp. 4542-4545

[141] A. M. Shams, A. M. Elgamel and A. M. Bayoumi, “Hybrid Mesh-Based/Block-Based Motion Compensation Architecture”, Proc. of the Second Int. Workshop on Digital and Computational Video, 8-9 Feb. 2001, pp. 194-201

[142] J. Choi, M. Yanagisawa, T. Ohtsuki and N. Togawa, "VLSI Architecture for a Flexible Motion Estimation with Parameters", Proc. of the 2002 Conf. on Asia South Pacific Design Automation/VLSI design, 2002, pp. 452

[143] G. V. Varatkar and N. R. Shanbhag, "Energy-Efficient Motion Estimation Using Error-Tolerance", Proc. of the 2006 Int. Symp. On Low Power Electronics and Design, Germany, 2006, pp. 113-118

[144] C. Chunhong, M. Sarrafzadeh, "Simultaneous Voltage Scaling and Gate Sizing for Low-Power Design", IEEE Trans. On Circuits and Systems II: Analogue and Digital Signal Processing, Vol. 49, Issue 6, June 2002, pp. 400-408

[145] M. Keating, D. Flynn, R. Aitken, A. Gibbons and K. Shi, "Low Power Methodology Manual for Systems-on-Chip Design", Springer, 2007

[146] S. G. Narendra and A. Chandrakasan, "Leakage in Nanometer CMOS Technologies", Springer, 2006

[147] S. H. Lo, D. A. Buchanan, Y. Taur and W. Wang, "Quantum-Mechanical Modelling of Electron Tunnelling Current from the Inversion Layer of Ultra-Thin-Oxide nMOSFET's", IEEE Electron Device Letters, Vol. 18, No. 5, May 1997, pp. 209-211

[148] A. P. Chandrakasan and R. W. Brodersen, "Low Power Digital CMOS Design", Kluwer Academic Publishers, 1995

[149] S. Wilton, W. Luk and S. Ang, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," Int'l Conf. on Field-Prog. Logic and its App., Antwerp, Aug. 2004

[150] J. Lamoureux and S. Wilton, "On the Interaction between Power-aware FPGA CAD Algorithms", Proc. ICCAD, 2003.

[151] Y. Yi and R. Woods, “Hierarchical Synthesis of Complex DSP Functions Using IRIS”, IEEE Trans on Computer Aided Design, Vol. 25, No. 5, May 2006, pp806-820

[152] C-Y Wang and K. K.Parhi, “The MARS High-Level DSP Synthesis System”, VLSI Design Methodologies for Digital Signal Processing Architectures, Kluwer Academic Publishers, 1994

[153] Keane, G., Spanier, J. R., and Woods, R., "Low-Power design of Signal Processing systems using Characterization of silicon IP cores", 33rd Asilomar Conference on Signals, Systems and Computers, Asilomar, USA, Oct. 1999, invited paper, IEEE Computer Society, pp. 767-771

Appendix A

TDPFPADD VHDL Code and Scripts

This appendix provides the synthesisable VHDL code for the TDPFPADD architecture used in chapter 4, section 4.1 to verify the PSI method. The code is presented in the following six sections A.1 to A.6 and can found in the attached CD.
Section A.7 presents the Magma synthesis script described in chapter 4, section 4.2.2.1 and section A.8 illustrate the PSI flow script used in chapter 4, section 4.3.

A.1 Exponent Logic

This block was described in section 4.1.1.1 and the code used in this thesis is given in the attached CD at the directory shown below. The VHDL implementation is hierarchical with top-level representing the Exponent Logic and below it the necessary components are instantiated. In addition the test bench to test the Exponent Logic is also provided at the following directory.

VHDL file : <cd drive> /TDPFPADD/src/ExponentLogic/ExponentLogic.vhdl
Testbench: <cd drive>/TDPFPADD/testbenches/TDPFPADD/TDPFPADD_testbench.vhdl

A.2 Control Logic

This block was described in section 4.1.1.2 and the VHDL code along with the test bench is provided in the attached CD at the following directory.

VHDL file : <cd drive> /TDPFPADD/src/ControlLogic/ControlLogic.vhdl
Testbench: <cd drive>/TDPFPADD/testbenches/TDPFPADD/TDPFPADD_testbench.vhdl

A.3 Bypass Data Path

This block was described in section 4.1.1.3 and the VHDL code is provided in the attached CD at the following directory. There is no test bench provided with this code because of the simplicity of the block.

VHDL file : <cd drive> /TDPFPADD/src/BYPASS/BYPASS.vhdl

A.4 Leading Zero Anticipatory (LZA) Data Path

This data path was described in section 4.1.1.4. The VHDL coding style adopted here is such that this path along with all the other paths is instantiated in the VHDL file shown below resulting in the TDPFPADD design. A test bench is also provided at the following directory.

VHDL file : <cd drive> /TDPFPADD/src/TDPFPADD/TDPFPADD.vhdl
Testbench: <cd drive>/TDPFPADD/testbenches/TDPFPADD/TDPFPADD_testbench.vhdl

A.5 Leading Zero Bounded (LZB) Data Path

This data path was described in section 4.1.1.5. The VHDL code for this data path is shown below. The coding style is hierarchical which means there are VHDL components instantiated and they are shown in the vhdl file below. However, common components that are already been instantiated in the previous paths are not included here. A test bench is also provided at the following directory.

VHDL file : <cd drive> /TDPFPADD/src/LZB/LZB.vhdl
Testbench: <cd drive>/TDPFPADD/testbenches/TDPFPADD/TDPFPADD_testbench.vhdl

A.6 Result Integration/Flag Logic

The operation for this block was described in section 4.1.1.6. The synthesisable VHDL code is shown in the file below. The verification of this block is done as part of the TDPFPADD verification, therefore no standalone test bench is needed.

VHDL file : <cd drive> /TDPFPADD/src/ResultIntegration/ ResultIntegration.vhdl
Testbench: <cd drive>/TDPFPADD/testbenches/TDPFPADD/TDPFPADD_testbench.vhdl

A.7 Magma Synthesis Script

The synthesis process described in section 4.2.2.1 was automated to provide the user with flexibility to run multiple synthesis runs and to take away the burden of manual operation. The following script is based on Magma BlastFusion tool commands version v4.2.54.34 (further details can be found in [33]) and is provided in the attached CD at the following directory. An example floorplan creation is also provided at the following directory.

Synthesis script : <cd drive> /TDPFPADD/synth/magma_synthesis.tcl
Floorplan script : <cd drive> /TDPFPADD/synth/example_floorplan.tcl

A.8 PSI Flow Script

The implementation of the PSI flow described in section 4.3 to analyse the power-performance of TDPFPADD architecture is provided in the attached CD at the following directory.

PSI flow script : <cd drive> /TDPFPADD/psi_flow/psi_flow.tcl

Appendix B

TDPFPADD Precision Reduction Analysis

In chapter 4 section 4.3.1.1, it was highlighted that register power consumption becomes less dominant when clock period is relatively large. In the case of TDPFPADD this represent clock period >3000ps, and adding or removing small number of registers did not have significant impact on overall power consumption. This is because at these clock periods, combinational logic power consumption is the dominant part of total power consumption. Similar observation was noticed during the analysis of precision reduction in TDPFPADD. Precision reduction can be used as a way of reducing power consumption by inhibiting the switching activity associated with the bits involved in precision reduction. Figure B.1 show grey boxes which represent reduction in precision. These grey bits and any associated logic when not involved in addition as part of precision reduction, the switching activity can be reduced by using clock gating. The TDPFPADD architecture was modified to operate with multiple precision reduction modes. The precision mode was set by dedicated input signals. A choice of 0, 2, 4, 8, and 16 bit reduction could be set by the dedicated input signals with 0 bit reduction resulting in operating the TDPFPADD without any reduction in precision. The architecture was implemented in VHDL hardware language and synthesised at 200MHz using 90nm technology library. Magma BlastRTL [33] was used as the synthesis engine with power estimation performed with Synopsys PrimePower [32]. It was noted that before certain precision reduction power actually became worse. Figure B.2 show power trend with precision reduction. For example, at precision reduction of 4, the new architecture consumed more power than the architecture without precision reduction capability. The reason for this can be explained in the light of our earlier observation: at relatively low frequencies combinational logic power consumption becomes the dominant part of overall power consumption. Changing the adder architecture to allow multiple precision modes meant introducing some combinational logic to maintain the correct operation of the adder and to implement clock gating. This extra logic dominated the power picture and the reduction in power due to reduction in switching activity was outweighed buy the power consumed in the additional combinational logic. It is only when precision is reduced by 16 bits, the power consumption of the multi-precision adder becomes less than the original adder as shown in Figure A.2. For this small reduction in power with relatively large reduction in precision, designers may only consider the multi-precision architecture when applications could tolerate the reduced precision and when the application run time is sufficiently long.

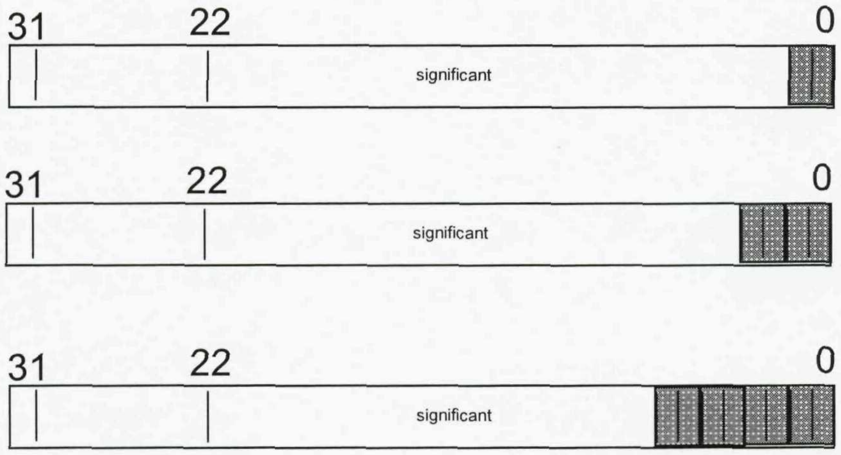


Figure B.1 Effect of precision on Floating-Point Number, Grey Area Represent Reduction in Precision

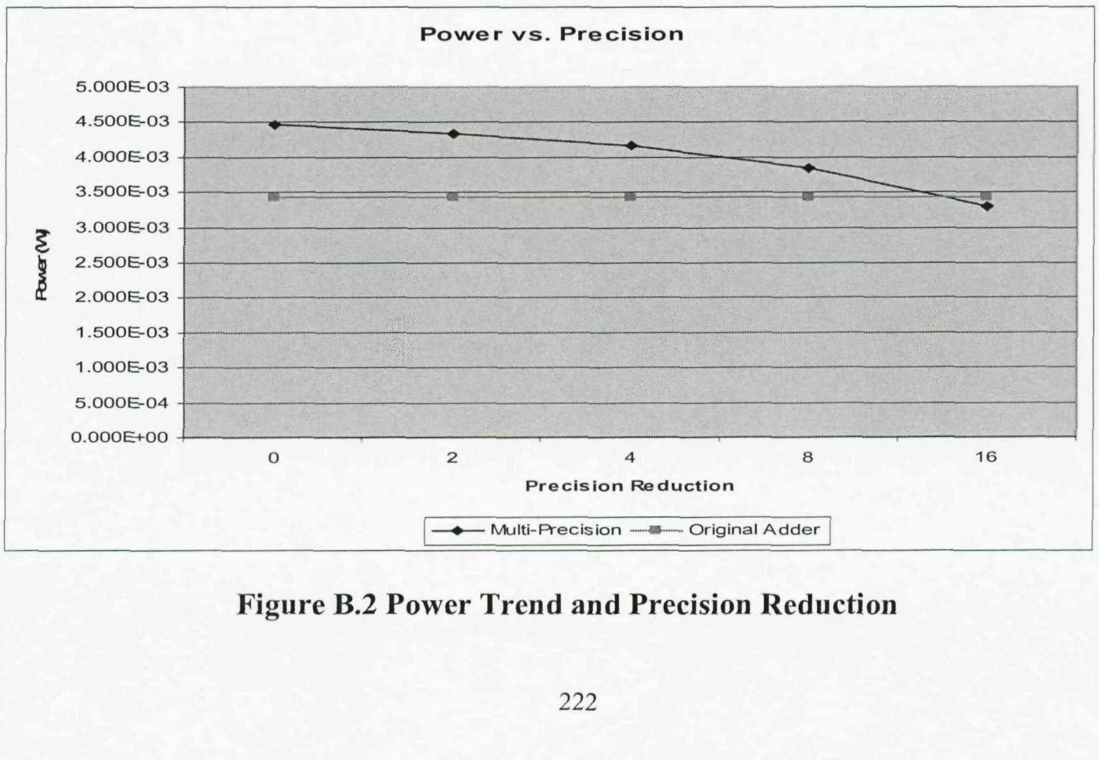


Figure B.2 Power Trend and Precision Reduction

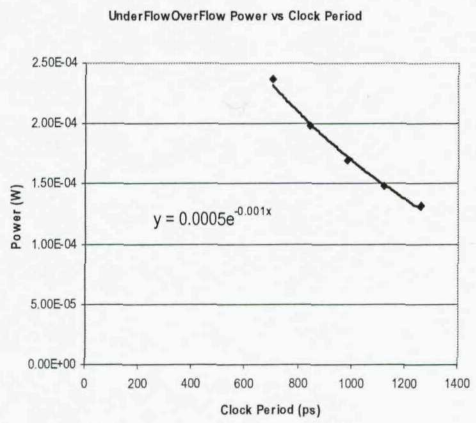
Appendix C

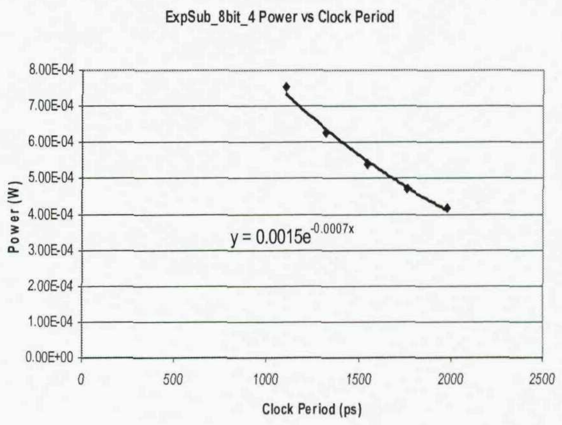
Experiments Data for the TDPFPADD Case Study

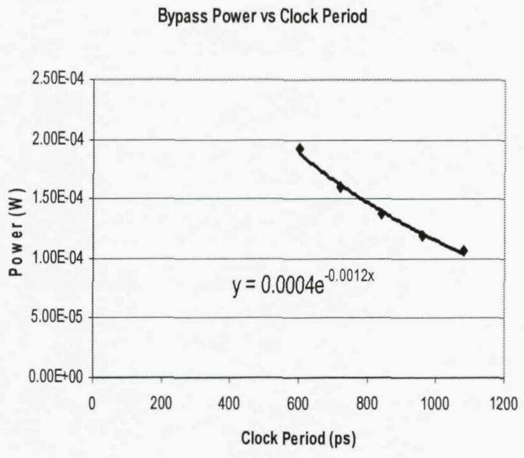
The following two sections presents the PSI method experiments data used in chapter 4. Section C.1 provides TDPFPADD FEs power equations. Section C.2 shows the power results from executing the PSI algorithm with different values for Φ.

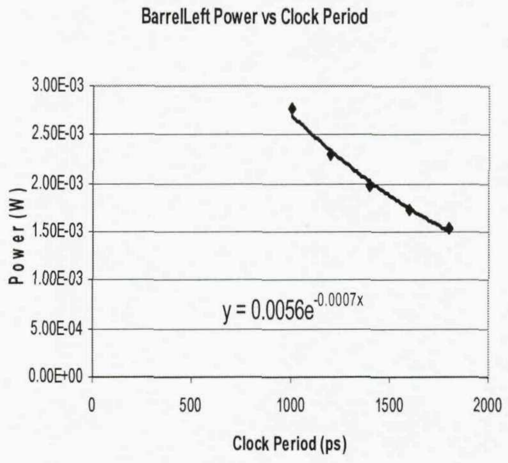
C.1 TDPFPADD Power Equations

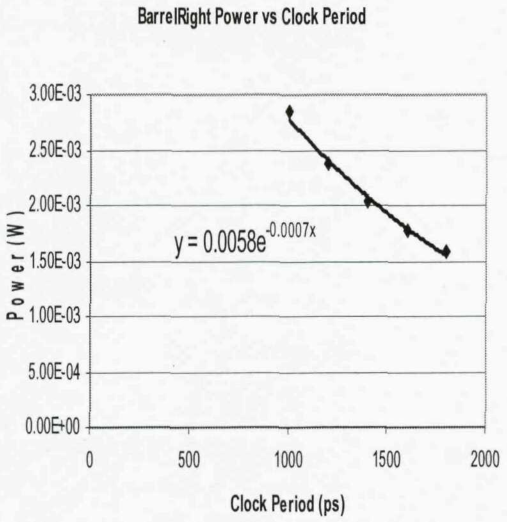
In chapter 4, section 4.2.2.2 it was stated that equations describing the power consumption as a function of the clock period was established for each FEs. This is done from the plots shown in this appendix. The power consumption was estimated at five different clock periods using PrimePower tool for each element. From these data a best fit equation was established as shown in the plots below. These equations were used as part of the TDPFPADD power estimation.

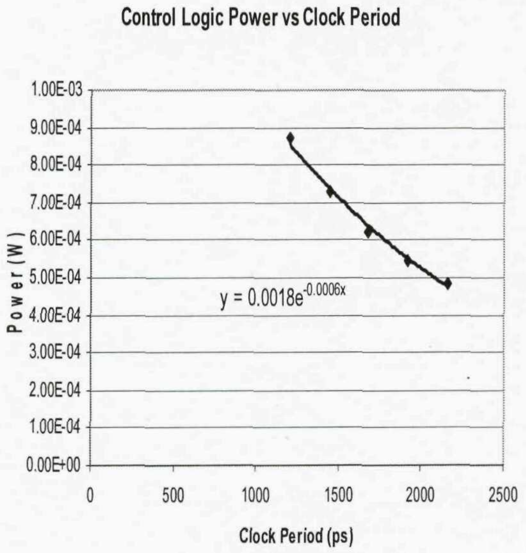


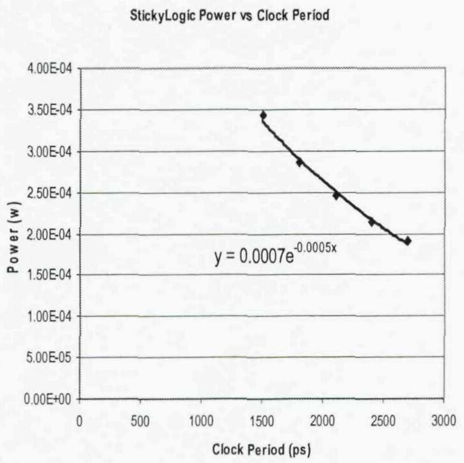


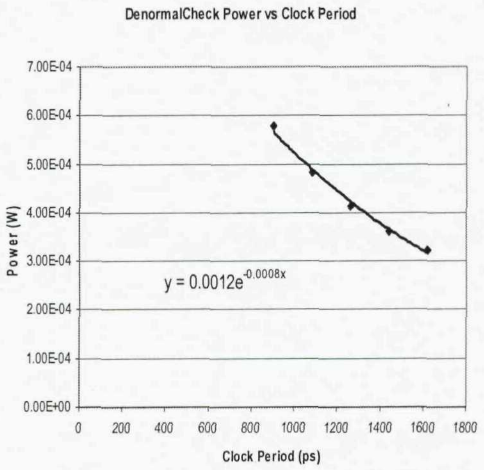

Bypass Power vs Clock Period
y = 0.0004e^-0.0012x
Power (W)
Clock Period (ps)

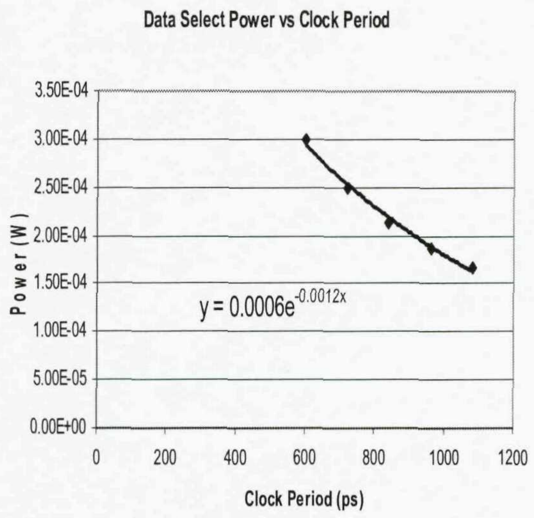

BarrelLeft Power vs Clock Period
y = 0.0056e^-0.0007x
Power (W)
Clock Period (ps)

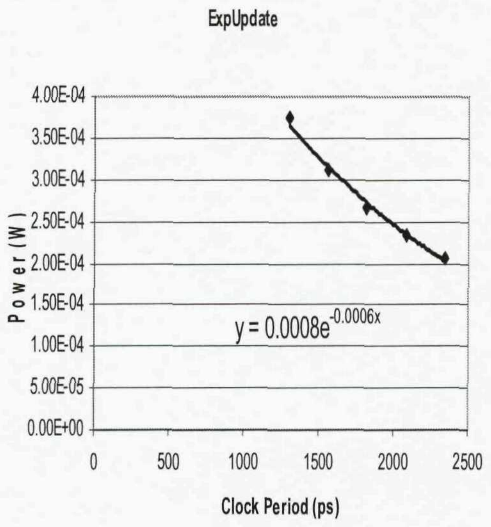

BarrelRight Power vs Clock Period
y = 0.0058e^-0.0007x
Power (W)
Clock Period (ps)

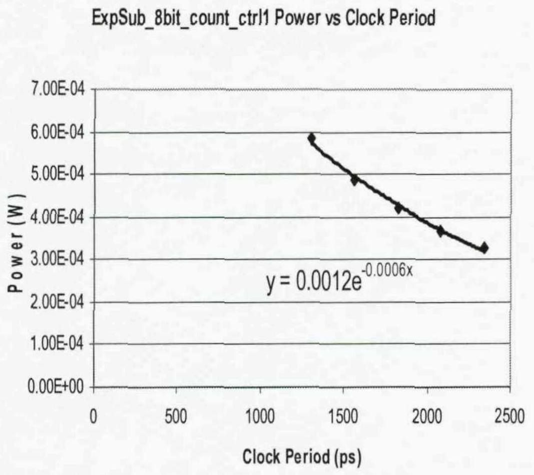

Control Logic Power vs Clock Period
y = 0.0018e^-0.0006x
Power (W)
Clock Period (ps)

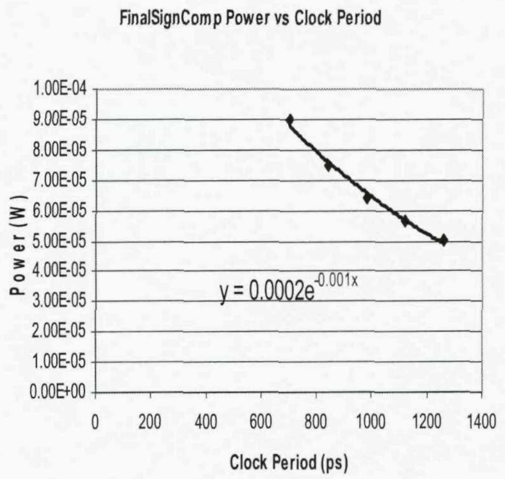

StickyLogic Power vs Clock Period
y = 0.0007e^-0.0005x
Power (w)
Clock Period (ps)

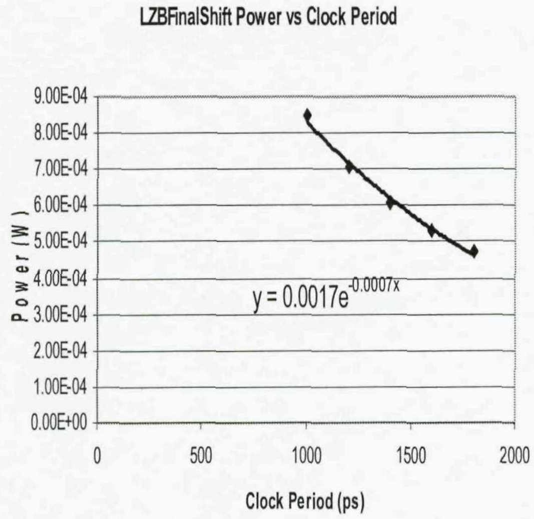

DenormalCheck Power vs Clock Period
y = 0.0012e^-0.0008x
Power (W)
Clock Period (ps)

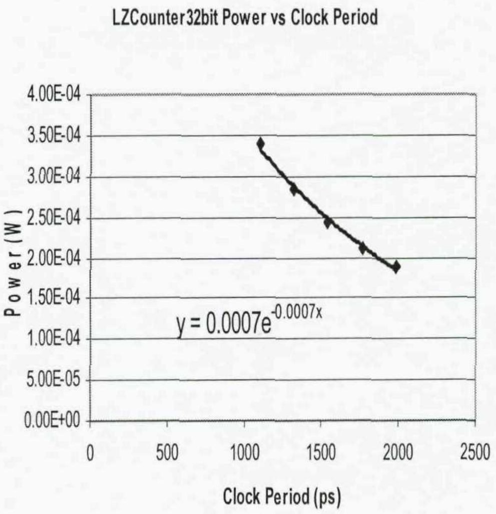

Data Select Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0006e^-0.0012x
3.50E-04
3.00E-04
2.50E-04
2.00E-04
1.50E-04
1.00E-04
5.00E-05
0.00E+00
0
200
400
600
800
1000
1200

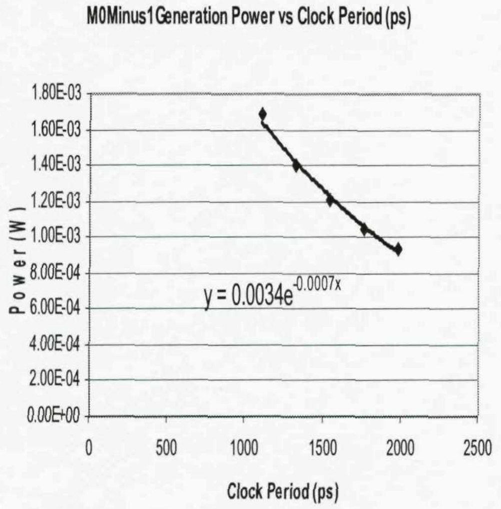

ExpUpdate
Power (W)
Clock Period (ps)
y = 0.0008e^-0.0006x
4.00E-04
3.50E-04
3.00E-04
2.50E-04
2.00E-04
1.50E-04
1.00E-04
5.00E-05
0.00E+00
0
500
1000
1500
2000
2500

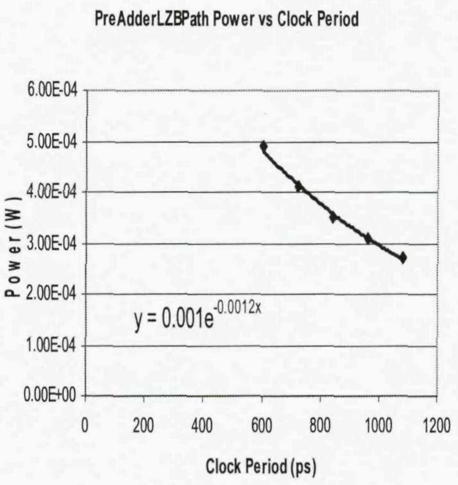

ExpSub_8bit_count_ctrl1 Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0012e^-0.0006x
7.00E-04
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
0
500
1000
1500
2000
2500

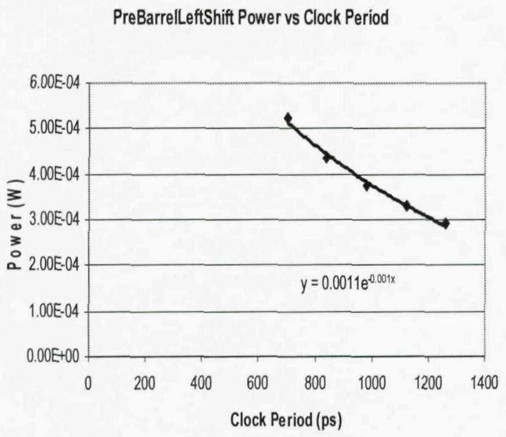

FinalSignComp Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0002e^-0.001x
1.00E-04
9.00E-05
8.00E-05
7.00E-05
6.00E-05
5.00E-05
4.00E-05
3.00E-05
2.00E-05
1.00E-05
0.00E+00
0
200
400
600
800
1000
1200
1400

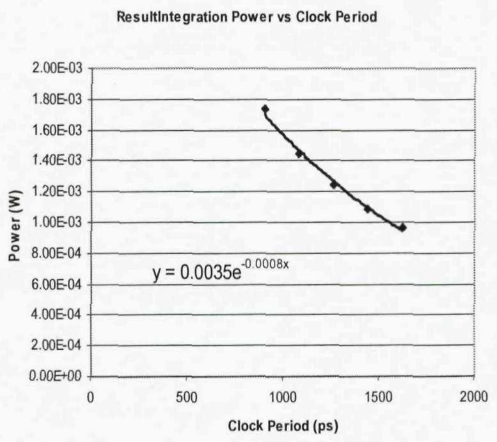

LZBFinalShift Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0017e^-0.0007x
9.00E-04
8.00E-04
7.00E-04
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
0
500
1000
1500
2000

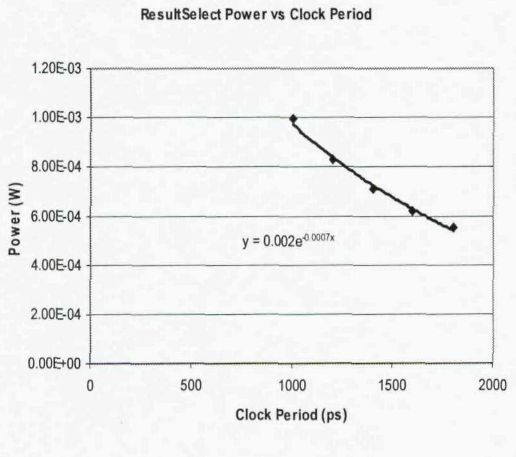

LZCounter32bit Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0007e^-0.0007x
4.00E-04
3.50E-04
3.00E-04
2.50E-04
2.00E-04
1.50E-04
1.00E-04
5.00E-05
0.00E+00
0
500
1000
1500
2000
2500

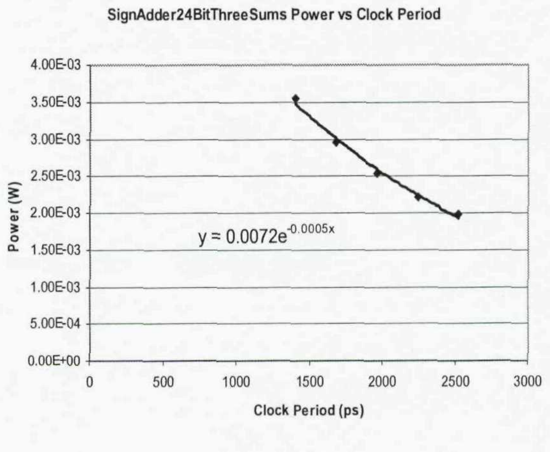

M0Minus1Generation Power vs Clock Period (ps)
Power (W)
Clock Period (ps)
y = 0.0034e^-0.0007x
1.80E-03
1.60E-03
1.40E-03
1.20E-03
1.00E-03
8.00E-04
6.00E-04
4.00E-04
2.00E-04
0.00E+00
0
500
1000
1500
2000
2500


PreAdderLZBPath Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.001e^-0.0012x
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
0
200
400
600
800
1000
1200


PreBarrelLeftShift Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0011e^-0.001x
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
0
200
400
600
800
1000
1200
1400


ResultIntegration Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0035e^-0.0008x
2.00E-03
1.80E-03
1.60E-03
1.40E-03
1.20E-03
1.00E-03
8.00E-04
6.00E-04
4.00E-04
2.00E-04
0.00E+00
0
500
1000
1500
2000


ResultSelect Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.002e^-0.0007x
1.20E-03
1.00E-03
8.00E-04
6.00E-04
4.00E-04
2.00E-04
0.00E+00
0
500
1000
1500
2000


SignAdder24BitThreeSums Power vs Clock Period
Power (W)
Clock Period (ps)
y = 0.0072e^-0.0005x
4.00E-03
3.50E-03
3.00E-03
2.50E-03
2.00E-03
1.50E-03
1.00E-03
5.00E-04
0.00E+00
0
500
1000
1500
2000
2500
3000

C.2 Power Results for Φ Between 0 to 3

In chapter 4, section 4.3.3.2 the PSI algorithm was executed with different values for Φ and clock period and the power consumption of TDPFPADD was estimated for each combination. Table C.1 and C.2 shows the power consumption figures when Φ was set to a value ranging from 0 to 3 with interval rate of 0.1 while the clock period was set to a value from 1500ps to 5000ps.

Table C.2. 1 Power Consumption Figures for Different Φ and Clock Period Values

	Clock Period (ps)					
Φ	1500	1700	1900	2100	2300	2500
0.0	5.08E-02	4.44E-02	3.88E-02	3.39E-02	2.96E-02	2.59E-02
0.1	5.08E-02	4.44E-02	3.88E-02	3.39E-02	2.96E-02	2.59E-02
0.2	5.08E-02	4.44E-02	3.88E-02	3.39E-02	2.96E-02	2.59E-02
0.3	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.4	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.5	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.6	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.7	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.8	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.79E-02	2.44E-02
0.9	5.08E-02	4.44E-02	3.65E-02	3.19E-02	2.61E-02	2.27E-02
1.0	5.08E-02	4.44E-02	3.44E-02	3.01E-02	2.61E-02	2.13E-02
1.1	5.08E-02	4.44E-02	3.44E-02	3.01E-02	2.50E-02	2.03E-02
1.2	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.3	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.4	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.5	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.6	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.7	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.8	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
1.9	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.0	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.1	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.2	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.3	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.4	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.5	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.6	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.7	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.8	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
2.9	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02
3.0	5.08E-02	4.21E-02	3.44E-02	3.01E-02	2.52E-02	2.03E-02

Table C.2. 2 Power Consumption Figures for Different Φ and Clock Period Values

	Clock Period (ps)						
Φ	2700	3000	3300	3600	4000	4400	5000
0.0	2.27E-02	1.86E-02	1.52E-02	1.25E-02	9.58E-03	7.37E-03	4.98E-03
0.1	2.27E-02	1.86E-02	1.52E-02	1.25E-02	9.58E-03	7.37E-03	4.98E-03
0.2	2.27E-02	1.86E-02	1.52E-02	1.25E-02	9.58E-03	7.37E-03	4.98E-03
0.3	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.4	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.5	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.6	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.7	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.8	2.14E-02	1.75E-02	1.44E-02	1.18E-02	9.06E-03	6.98E-03	4.73E-03
0.9	1.98E-02	1.49E-02	1.22E-02	1.00E-02	7.68E-03	5.93E-03	3.76E-03
1.0	1.86E-02	1.39E-02	1.14E-02	9.40E-03	7.20E-03	5.57E-03	3.76E-03
1.1	1.78E-02	1.32E-02	1.09E-02	8.96E-03	6.86E-03	5.32E-03	3.60E-03
1.2	1.78E-02	1.34E-02	1.10E-02	8.30E-03	6.43E-03	4.91E-03	3.11E-03
1.3	1.78E-02	1.34E-02	1.10E-02	8.30E-03	6.43E-03	4.91E-03	3.11E-03
1.4	1.78E-02	1.34E-02	1.10E-02	7.84E-03	6.08E-03	4.65E-03	2.94E-03
1.5	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
1.6	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
1.7	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
1.8	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
1.9	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.0	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.1	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.2	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.3	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.4	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.5	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.6	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.7	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.8	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
2.9	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03
3.0	1.67E-02	1.23E-02	1.01E-02	7.84E-03	6.08E-03	4.65E-03	2.97E-03

Appendix D

Berkley Motion Compensation Module

The following sections highlight the implementation of Berkley Motion Compensation in C-code and VHDL code described in chapter 5, section 5.4. The codes can be found in the attached CD at the directories given below.

D.1 Berkley Motion Compensation

This section provide the C-code implementation of the motion compensation algorithm from Berkley [99] used in chapter 5 to validate the PSI algorithm. The code can be found at the following directory.

C-code file : <cd drive>/MPEG1_MotionComp/c_src/MPEG1_MotionComp.c

D.2 VHDL Code of Berkley Motion Compensation

This section shows the translation of the motion compensation algorithm (by the author) into synthesisable VHDL code used in chapter 5 to validate the PSI algorithm. The top-level entity is called MotionVector and can be found at the following directory in the attached CD. A test bench is provided in the following directory.

VHDL file: <cd drive>/ MPEG1_MotionComp/vhdl_src/ MotionComp.vhdl
Testbench: <cd drive>/MPEG1_MotionComp/testbench/gate_sim/ motion_tb.vhdl

Appendix E

Experiments Data for the Motion Compensation Case Study

In chapter 5, section 5.5.5, the energy efficiency curve was observed to peak at low frequency. This appendix aims to provide an explanation for this observation in section E.1 and highlights the influence of register and combinational power on the energy efficiency curve. Section E.2 shows the PrimeTime script highlighted in Figure 6.3 chapter 6, section 6.3.1 to estimate FEs delay at 1.05V.

E.1 Register and Combinational Power Consumption

In section 5.5.5 it was mentioned that the reason for the PSI energy efficiency curve peaking at low frequency 33Mhz (30 ps) was down to the fact that combination logic at that period is the dominant power consumer when compared to register power. This is confirmed by Figure E.1.1 This figure illustrate that at 30ns clock period combinational power is almost 1mW compared with 0.01mW for registers. As the clock period is increased it is observed that the total power almost merge with combinational power curve indicating that combinational power is the significant contributor. However we also notice that the gap between combinational and register power curves narrows down significantly after 30ns clock period which is important because this manifest itself to the rapid decline in the energy efficiency curve shown in Figure 5.15.

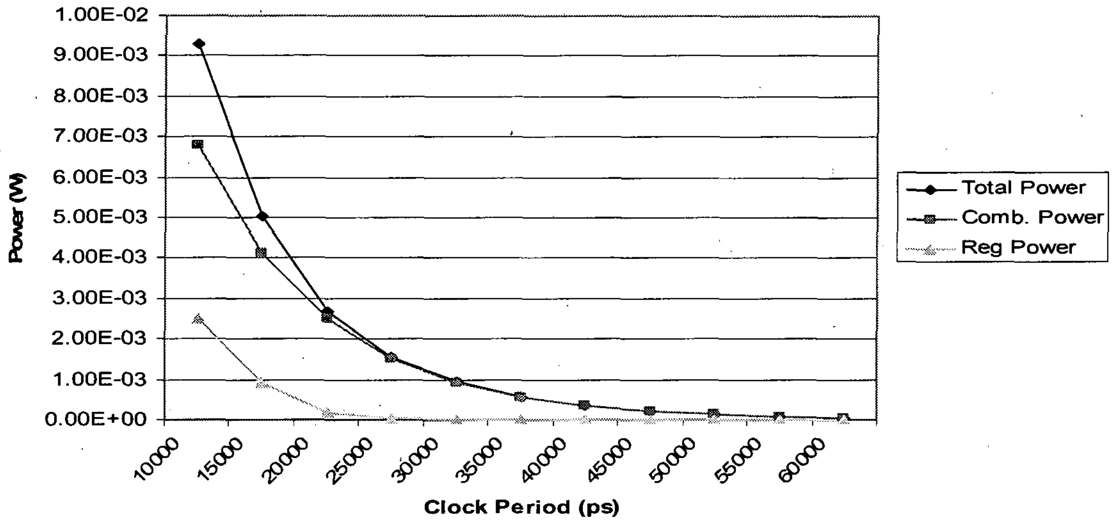


Figure E.1. 1 Breakdown of Combinational and Register Power Consumption

Figure E.1.2 and Figure E.1.3 shows the data used to generate the power equations for combinational logic and a register respectively. These equations are the bases of the result shown in Figure E.1.1.

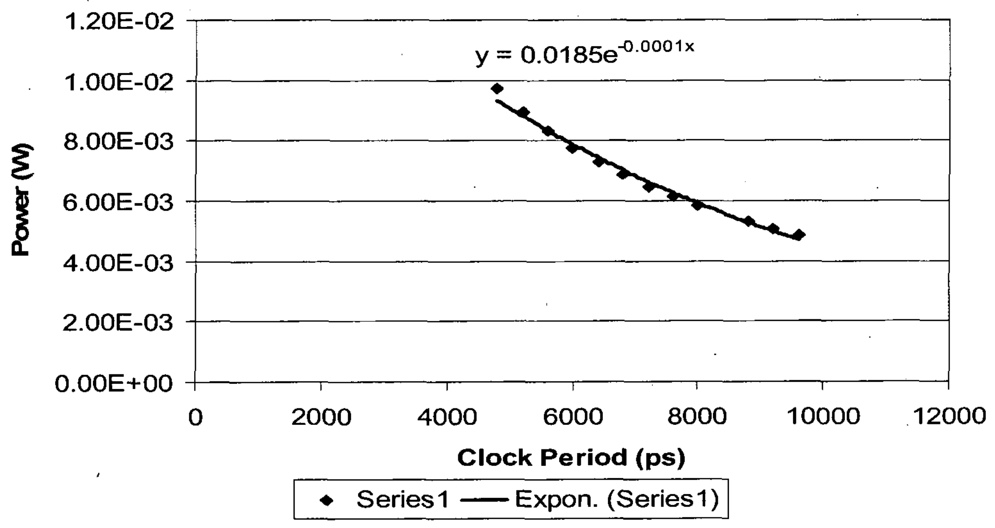


Figure E.1. 2 Combinational Logic Power Consumption versus Clock Period

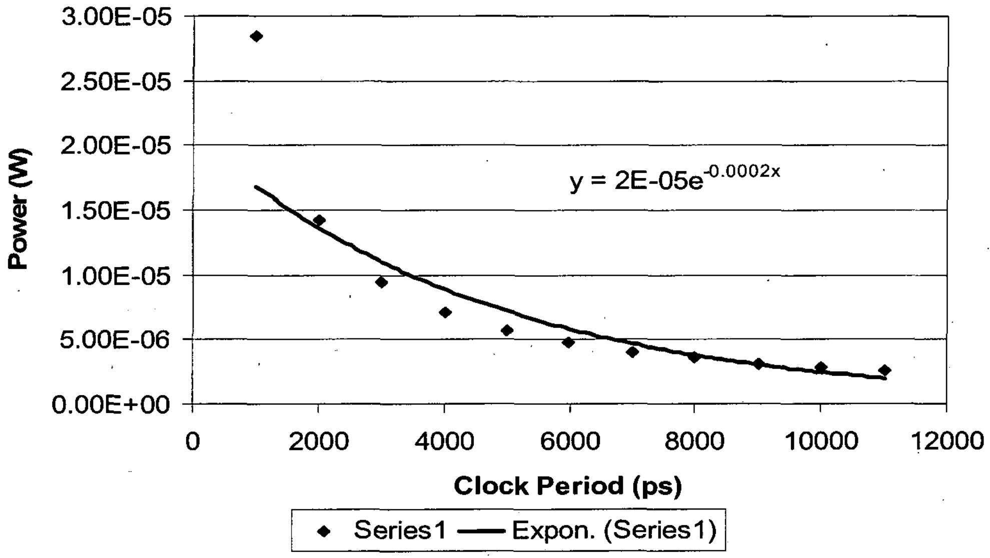


Figure E.1. 3 Register Power Consumption versus Clock Period

E.2 PrimeTime Delay Estimation Script

The following shows the PrimeTime script used in chapter 6, section 6.3.1 and illustrated in Figure 6.3 used to estimate FEs delay at 1.05V. Note the script firstly initialises the work paths by defining the relevant variables which points to the area where data is stored. It then loads in the technology library with the variable link_library and after that the verilog netlist of the FE is loaded with read_file command. After the clocks are defined and some default settings are applied the delay of the elements are found with the aid of this command [get_attr [get_timing_paths] slack]. For further details about PrimeTime commands see [32].

```
#----------------------------------------
# Initialise work paths
#----------------------------------------
set volcano_dir /projects/asicwork/haiali/mpeg_player/magma/fixnetlist_volcanos
set netlist_dir /projects/asicwork/haiali/mpeg_player/magma/netlists_nopipe
set log_files  /projects/asicwork/haiali/mpeg_player/magma/ModuleDelay
set power_results ./power_results

foreach voltage "1.05" {
set design_list ""
```

```
#------------------------------------------------------------------
# load libs
#------------------------------------------------------------------
foreach volc [glob -nocomplain $volcano_dir/*.volcano] {
regexp ".*/.*module_(.*)\.volcano" $volc whole_match netlist
regexp ".*/(.*)_module" $volc whole_match inst

set file_id1 [open ${log_files}/${inst}.log r]
while { [gets $file_id1 line] >= 0} {
  if {[regexp ".*timing at (.*)p$" $line whole_match per]} {
  }
}
close $file_id1

remove_design -all

set pyramid /nokia/fa_nmp/groups/rapido/v1.0/backend/db0_1_lib0_18/library

set search_path [concat $pyramid/CORE/synopsys $pyramid/CTS/synopsys
$pyramid/CORE_NT/synopsys $pyramid/CORE_LP/synopsys
$pyramid/CTS_LP/synopsys]

set link_library [concat "*" GS50_N_25_${voltage}_CORE.db
GS50_N_25_${voltage}_CORE_NT.db GS50_N_25_${voltage}_CORE_LP.db]

#------------------------------------------------------------------
# load design and activity files
#------------------------------------------------------------------
read_file -format verilog ${netlist_dir}/${netlist}.v
current_design $netlist
link
#------------------------------------------------------------------
# apply clocks as for synth
#------------------------------------------------------------------
set_operating_conditions N_25_${voltage}
# Clock Declaration

create_clock -name clock -period [expr $per / 1000.0] -waveform "0 [expr [expr $per
/ 1000.0] / 2]" [get_ports {clock}]
set_input_delay -clock clock  0.2 [remove_from_collection [all_inputs] "clock"]
set_output_delay -clock clock 0.2 [all_outputs]

#------------------------------------------------------------------
# apply default parameters
#------------------------------------------------------------------
set_input_transition 0.4 [all_inputs]
set_max_transition 0.4 [current_design]
#-------------------------------------------------------------------
# backannotation
#-------------------------------------------------------------------
read_parasitics -format SPEF  ${netlist_dir}/${netlist}.spef
report_annotated_parasitics

puts "design: [get_object_name [current_design]] per: $per slack: [get_attr
[get_timing_paths] slack]"

lappend design_list "inst: $inst design: [get_object_name [current_design]] per: $per
slack: [get_attr [get_timing_paths] slack] path delay: [expr $per - [expr 1000.0 *
[get_attr [get_timing_paths] slack]]]"
}
set file_id1 [open module_delays${voltage}v.txt w]
foreach p $design_list {
   puts $file_id1 $p
}
 close $file_id1
}
```