DEVELOPMENT OF A LOW-POWER LMS ADAPTIVE FILTER FOR EEG SIGNALS

THESIS

PRESENTED AS A PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE WITH MAJOR IN ELECTRONIC ENGINEERING
(ELECTRONIC SYSTEMS)

BY

LUIS ALBERTO VALENCIA MEDINA

MONTERREY, N.L., MEXICO. DECEMBER, 2007
The members of the thesis committee hereby approve the thesis of Luis Alberto Valencia Medina as a partial fulfillment of the requirements for the degree of Master of Science with major in

Electronic Engineering
(Electronic Systems)

Thesis Committee

--------------------
Alfonso Ávila, Ph.D.
Thesis Advisor

--------------------
Sergio Martínez, Ph.D.
Thesis Reader

--------------------
Graciano Dieck, Ph.D.
Thesis Reader

--------------------
Graciano Dieck Assad, Ph.D.
Director of the Graduate Programs in Mechatronics and Information Technologies

December, 2007
DEVELOPMENT OF A LOW-POWER LMS ADAPTIVE FILTER FOR EEG SIGNALS

BY

LUIS ALBERTO VALENCIA MEDINA

THESIS

PRESENTED TO THE GRADUATE PROGRAM IN MECHATRONICS AND INFORMATION TECHNOLOGIES

THIS THESIS IS A PARTIAL REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE WITH MAJOR IN

ELECTRONIC ENGINEERING (ELECTRONIC SYSTEMS)

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

MONTERREY CAMPUS

DECEMBER, 2007
A mis padres y abuelos...
Acknowledgements

I would like to start by expressing my sincere gratitudes to my advisor Professor Alfonso Ávila for his attentions, guidance, and friendship over these years. His help and patience are greatly appreciated. I would like to acknowledge Professor Sergio Omar Martínez for his valuable time and for his teachings. I truly admire his constant drive toward his work. I am grateful to Professor Graciano Dieck for his comments and uplifting attitude toward my thesis work.

I am truly grateful for having participated in an exchange program. I thank everyone involved, particularly Professors David Garza and Hesham El-Rewini. Thanks to my professors at SMU specially Scott Douglas and Dennis Frailey. Thanks to ITESM, CONACYT, and USAID.

Thanks to all my friends and everyone who has supported me; I really appreciate the times we had together. Thanks to Cinthya and César for letting me be part of the family; to Ale and Armando for their friendship in times when I needed it the most; and to Bety and Erik for their laughs. Also, super special thanks to “La Cueva” and the “LabMEMS Crew”: Carlos, Marco, Farid, David, Saracho, Ever, Israel, Salvador, Toño, Chava, José Luis, Fernando, Bere, Daniel? et al. So much fun...

I would like to thank my beautiful girlfriend Mónica for her loving support and understanding during the most hectic times. She is my source of inspiration and love; she is my everything. Better times are here milady...

Finally, I would like to express my gratitude to my family, this work is the result of your never-ending support and love. Thank you for helping me achieve an important objective in the pursuit of the goal.

Luis Alberto Valencia Medina

Instituto Tecnológico y de Estudios Superiores de Monterrey

December, 2007
DEVELOPMENT OF A LOW-POWER LMS ADAPTIVE FILTER FOR EEG SIGNALS

Luis Alberto Valencia Medina, M.S.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2007

Thesis Advisor: Alfonso Ávila, Ph.D.

Abstract

The aim of this thesis is the evaluation of power consumption rates for different LMS adaptive digital filter architectures. The filter is used to attenuate the power line interference in the recording of EEG signals.

A Simulink-based design flow is utilized to provide rapid hardware evaluation of the architectures proposed, from algorithm design to physical layout. To demonstrate the advantages of this design flow, a pipelined filter was implemented in 0.35 µm technology from AMS.

In order to simplify the power consumption, delay, and area estimation, and given the large number of transistors contained in the adaptive filters, a characterization methodology is introduced. Due to the high regularity of the filter, the basic functional units were characterized using this methodology.

Finally, this thesis work analyzes the performance and power consumption of five dedicated DSP architectures implementing the LMS algorithm. Low-power operation was accomplished by using voltage scaling while the throughput is maintained by means of pipelining and relaxed look-ahead techniques. Simulation results show that if the filters were to be run at their maximum allowable clock frequency, power savings at around 75% and up can be obtained by using a supply voltage lower than 1.95 V. For practical purposes, the area overhead can be neglected.
Contents

Acknowledgements i

Abstract iii

1 Introduction 1
   1.1 Statement of the Problem 2
   1.2 Objectives 3
   1.3 Previous Work 4
   1.4 Thesis Outline 5

2 Principles of Low-Power CMOS Digital Design 7
   2.1 Introduction 7
   2.2 Definitions of Power and Energy 8
      2.2.1 Power-Delay Product 9
      2.2.2 Energy-Delay Product 10
   2.3 Power Reduction in CMOS Digital Circuits 10
      2.3.1 Voltage Scaling 11
         2.3.1.1 Parallel Processing for Low Power 12
         2.3.1.2 Pipelining for Low Power 13
      2.3.2 Reduction of Switching Activity 15
      2.3.3 Reduction of Switched Capacitance 17
   2.4 Pipelining Adaptive Digital Filters 18
      2.4.1 Relaxed Look-Ahead Techniques 19
         2.4.1.1 Product relaxation 19
         2.4.1.2 Sum relaxation 21
         2.4.1.3 Delay relaxation 21
2.5 A Survey of Power Optimization Techniques .................................. 23
2.5.1 Memory Systems .................................................. 23
2.5.2 Buses ................................................................. 25
2.5.3 Processor ............................................................. 26
2.5.4 Dynamic Voltage Scaling ............................................. 28
2.5.5 Software ............................................................ 28
2.5.6 Power Management Techniques in the Industry ....................... 29
2.5.7 Power-Performance Analysis Tools .................................. 31

3 Filter Architecture .......................................................... 32
3.1 Introduction ............................................................ 32
3.2 Least-Mean-Square Algorithm ........................................... 32
  3.2.1 Overview of the LMS Algorithm .................................... 33
  3.2.2 LMS Adaptation Algorithm ....................................... 34
3.3 Adaptive Noise Canceling ............................................... 36
3.4 Input Signals Models .................................................. 38
  3.4.1 EEG Signal Model ............................................... 38
3.5 Application of Relaxed Look-Ahead Techniques .......................... 40
  3.5.1 Delay Relaxation ................................................ 40
  3.5.2 Sum Relaxation .................................................. 41
3.6 The Architecture ........................................................ 41

4 Design Flow and System Implementation .................................... 43
4.1 Introduction ............................................................ 43
4.2 Design Styles .......................................................... 44
4.3 Design Flow ............................................................ 44
  4.3.1 Algorithm Design ................................................ 46
    4.3.1.1 Matlab/Simulink ............................................ 47
  4.3.2 RTL Model ........................................................ 48
    4.3.2.1 System Generator .......................................... 48
  4.3.3 Gate-Level Model ............................................... 49
    4.3.3.1 LeonardoSpectrum ......................................... 49
  4.3.4 Physical Layout .................................................. 51
    4.3.4.1 IC Station ................................................. 51
D LeonardoSpectrum Script 100

Bibliography 103

Vita 112
List of Figures

2.1 Energy-Delay Product .................................................. 11
2.2 Single-stage implementation of a logic function .................. 13
2.3 L-Parallel structure realizing the same logic function as in Figure 2.2 . 14
2.4 Critical paths for original and 3-parallel systems ................. 15
2.5 M-level pipelined structure realizing the function shown in Figure 2.2 . 16
2.6 Critical paths for original and 3-level pipelined systems ......... 16
2.7 Signal glitching in multi-level static CMOS digital circuits ........ 17
2.8 Gating signals $X_i$ to enable clock to certain blocks on demand ... 17
2.9 A 4-level look-ahead pipelined 1st-order recursion .............. 20
2.10 A 4-level product-relaxed look-ahead pipelined 1st-order recursion ... 21
2.11 A 4-level sum-relaxed look-ahead pipelined 1st-order recursion ... 22

3.1 LMS algorithm block diagram ....................................... 35
3.2 F-block ................................................................. 36
3.3 Weight-update block ................................................... 37
3.4 Adaptive noise canceling system .................................... 38
3.5 Block diagram of the stationary MPA model ..................... 39
3.6 The pipelined architecture ............................................ 42

4.1 Classification of custom and semi-custom design styles .......... 45
4.2 Waterfall design flow for ASIC development .................... 46
4.3 Simulink-based design flow ....................................... 47
4.4 Design environment in Simulink using System Generator ........ 49
4.5 Logic synthesis process ............................................. 50
4.6 Automated layout design flow .................................... 52
4.7 Typical simulation process ......................................... 53
4.8 Gate-level simulation ............................................... 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>Eldo input and output files</td>
<td>54</td>
</tr>
<tr>
<td>4.10</td>
<td>Simulink model of the pipelined LMS algorithm</td>
<td>57</td>
</tr>
<tr>
<td>4.11</td>
<td>User-defined parameters</td>
<td>58</td>
</tr>
<tr>
<td>4.12</td>
<td>Simulation configuration</td>
<td>59</td>
</tr>
<tr>
<td>4.13</td>
<td>Hardware co-simulation</td>
<td>60</td>
</tr>
<tr>
<td>4.14</td>
<td>Test bench results</td>
<td>61</td>
</tr>
<tr>
<td>4.15</td>
<td>Preliminary physical layout for the pipelined adaptive filter</td>
<td>65</td>
</tr>
<tr>
<td>4.16</td>
<td>Block-level characterization</td>
<td>66</td>
</tr>
<tr>
<td>4.17</td>
<td>Circuit configuration used for power estimation</td>
<td>67</td>
</tr>
<tr>
<td>4.18</td>
<td>Characterization in Simulink</td>
<td>68</td>
</tr>
<tr>
<td>4.19</td>
<td>Vector translation process</td>
<td>69</td>
</tr>
<tr>
<td>4.20</td>
<td>Measuring area with the ruler</td>
<td>70</td>
</tr>
<tr>
<td>5.1</td>
<td>Power consumption of a multiplier block for different word sizes</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Propagation delay of a multiplier block for different word sizes</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Power-delay product of a multiplier block for different word sizes</td>
<td>74</td>
</tr>
<tr>
<td>5.4</td>
<td>Power consumption of an adder block for different word sizes</td>
<td>75</td>
</tr>
<tr>
<td>5.5</td>
<td>Propagation delay of an adder block for different word sizes</td>
<td>76</td>
</tr>
<tr>
<td>5.6</td>
<td>Power-delay product of an adder block for different word sizes</td>
<td>77</td>
</tr>
<tr>
<td>5.7</td>
<td>Power consumption of a register block for different word sizes</td>
<td>78</td>
</tr>
<tr>
<td>5.8</td>
<td>Propagation delay of a register block for different word sizes</td>
<td>78</td>
</tr>
<tr>
<td>5.9</td>
<td>Power-delay product of a register block for different word sizes</td>
<td>79</td>
</tr>
<tr>
<td>5.10</td>
<td>Serial LMS architecture</td>
<td>80</td>
</tr>
<tr>
<td>5.11</td>
<td>Pipelined LMS architecture with D1=1, D2=1, and LA=1</td>
<td>81</td>
</tr>
<tr>
<td>5.12</td>
<td>Pipelined LMS architecture with D1=11, D2=1, and LA=1</td>
<td>81</td>
</tr>
<tr>
<td>5.13</td>
<td>Pipelined LMS architecture with D1=42, D2=3, and LA=1</td>
<td>82</td>
</tr>
<tr>
<td>5.14</td>
<td>Pipelined LMS architecture with D1=45, D2=3, and LA=2</td>
<td>83</td>
</tr>
<tr>
<td>5.15</td>
<td>Comparison of the filter performance for different architectures</td>
<td>84</td>
</tr>
<tr>
<td>5.16</td>
<td>Comparison of MSE for different architectures</td>
<td>85</td>
</tr>
<tr>
<td>5.17</td>
<td>Comparison of the filter latency</td>
<td>85</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Comparison of Serial LMS and Pipelined LMS algorithms ........... 42

5.1 Area and number of transistors of a multiplier block ............... 75
5.2 Area and number of transistors of an adder block .................. 76
5.3 Area and number of transistors of a register block ................. 79
5.4 Power and energy comparison for five adaptive filter implementations . 87
Chapter 1

Introduction

Epilepsy is one of the most common neurologic disorders. Approximately 10% of the United States population will experience a single seizure during their lifetime, and approximately one in 100 Americans has epilepsy [1].

Typically, suspected seizures are evaluated using a routine electroencephalogram (EEG), which typically is a 20-min sampling of the patient’s brain waves. Because a routine EEG is brief, it is unlikely that actual events are recorded. Ambulatory EEG (AEEG) is a valuable tool for characterizing seizures and seizure-like events in the home setting.

Long-term outpatient recording of the EEG answers a need where the usual short-term routine EEG is negative or questionable, simply because the paroxysmal activity did not occur at the time that the patient was being evaluated. Frequently the level of clinical suspicion does not warrant expensive inpatient evaluation. Extension of EEG recording outside the confines of the EEG laboratory on a routine basis allows physicians to take full advantage of the potential usefulness of continuous, long-term EEG monitoring. The increased amount of data which can be processed from the patient’s normal home environment serves to shorten hospitalization, decreases risk to the patient while providing more accurate diagnostic information, increases the number of patients who can be accommodated at epilepsy centers, and decreases per-patient costs. Long-term outpatient recording is particularly convenient for pediatric patients because it minimizes the time away from home [2].
1.1 Statement of the Problem

The BioMEMS Research Group at Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey is developing a wireless integrated system for the acquisition and processing of electroencephalographic (EEG) signals. Integration of the electrode and processing circuitry in a single chip represents a great advantage with respect to many of the existing systems for ambulatory EEG monitoring. The monolithic integration of a large number of functions on a single chip usually provides [3]:

- Lower area/volume and therefore, compactness.
- Lower power consumption.
- Less testing requirements at system level.
- Higher reliability, mainly due to improved on-chip interconnects.
- Higher speed, due to significantly reduced interconnection length.
- Significant cost savings.

Interference deriving from the power transmission lines may corrupt useful information, making it more difficult for the physician to interpret the signal of interest, and even equivocate his diagnostic. Electrical shielding may reduce the level of interference; however, a notch filter is necessary to attenuate the remaining interfering signal [4].

The ambulatory monitoring of EEG signals is affected by a great amount of induced noise on the acquired signals. This monitoring requires to store a great amount of information and to transmit it in an efficient way. The equipment for the ambulatory monitoring is portable, therefore, it requires a low-power consumption [5].

The current state of the art EEG signal recorders that are present in the biomedical market, are limited in the power consumption area. These portable equipments are offered in presentations that can operate up to 24 hours continuously running only from the batteries. The batteries are usually of the lithium-ion rechargeable type. However, due to the portable nature of the device, the size of the batteries must be miniaturized. This is troublesome because the user would have to recharge the battery everyday. Thus, obtaining the required performance within a limited power budget is one of the most challenging goals in custom EEG signal recording device designs. In
this specific application area, performance is of secondary importance since low-power consumption takes first priority, while the clock rate is merely in the kHz range [6].

In this thesis work, the development of a low-power LMS adaptive digital notch filter is proposed for the reduction of the interference introduced by power transmission lines in the recording of EEG signals. The thesis will evaluate power consumption rates for different LMS adaptive digital filter architectures using static CMOS logic family and pipelining for low power.

1.2 Objectives

The general objective of this thesis is to evaluate power consumption rates for different LMS adaptive digital notch filter architectures utilizing pipelining. The notch filter will be used to attenuate the power line interference in the recording of EEG signals. Therefore, the system will be an adaptive noise cancelling system. The filter will be implemented in an ASIC complying with the power constraints imposed by the project currently in development by the BioMEMS Research Group. The specific objectives are:

- To implement a low-power LMS adaptive digital notch filter.
- To build a prototype of the LMS adaptive filter in an FPGA utilizing hardware in the loop with Simulink simulations for different filter architectures, and compare their performance both in software and hardware co-simulation for the RTL model.
- To translate the VHDL code to standard cells and simulate the proposed architectures, measuring their power consumption at the transistor level.
- To analyze and implement the low-power design techniques for static CMOS logic family in the chosen architecture.
- To simulate several solutions given the methods, compare their performance, area, and power consumption, and find the optimal architecture to implement the filter according to its feasibility.
The key thesis contributions which address the goal of this research are the implementation of an LMS adaptive digital filter from algorithm design to physical layout; the demonstration of power savings by applying architectural transformations techniques; and the introduction of a design flow that allows the simplification of the whole design process by accelerating the software-hardware verification.

1.3 Previous Work

The use of very large-scale integration (VLSI) techniques in biomedical instrumentation has opened the doors towards the miniaturization and portability of monitoring systems. Among other benefits this portability gives more freedom of movements to the patient (of particular importance in long duration medical exams) [7].

Roy et al [6], implemented an ultra-low-power, delayed least mean square (DLMS) adaptive filter operating in the subthreshold region for hearing aid applications. Sub-threshold operation was accomplished by using a parallel architecture with pseudo nMOS logic style. The parallel architecture enabled them to operate the system at a lower clock rate and reduced supply voltage while maintaining the same throughput. Simulation results showed that the DLMS adaptive filter can operate at 22 kHz using a 400-mv supply voltage to achieve 91% improvement in power compared to nonparallel, CMOS implementation.

Shanbhag and Goel [8] presented a low-power and high-speed algorithms and architectures for complex adaptive filters. These architectures have been derived via the application of algebraic and algorithm transformations. A fine-grained pipelined architecture for the strength-reduced algorithm is then developed via the relaxed look-ahead transformation. Convergence analysis of the proposed architecture was supported via simulation results. The pipelined architecture allowed high-speed operation with negligible hardware overhead. It also enabled an additional power saving of 39 to 69% when combined with power-supply reduction.

Douglas, Zhu, and Smith [9] described a hardware-efficient pipelined architecture for the LMS adaptive FIR filter that produced the same output and error signals as would have been produced by the standard LMS adaptive filter architecture without adaptation delays. Unlike existing architectures for delayless LMS adaptation, the new architecture’s throughout is independent of the filter length.

Matsubara et al [10] proposed an adaptive LMS algorithm that can be pipelined
and has an efficient architecture for hardware implementation. The proposed algorithm shows the possibility of having LMS algorithms perform pipelined processing without degrading the convergence characteristics.

Dukel et al [11] described the implementation of a pipelined low-power 6 taps adaptive filter based on the LMS algorithm. The architecture shows a novel tradeoff between algorithmic performance and power dissipation. The power was characterized with a natural top-down design methodology with iterative improvement.

Matsubara et al [12] proposed an adaptive algorithm, which can be pipelined, as an extension of the delayed LMS adaptive algorithm. The proposed algorithm provides a capability to achieve high throughput with less degradation of the convergence characteristic than the DLMS algorithm. An efficient implementation of the architecture with less hardware is also considered.

Harada et al [13] presented a pipelined architecture for the normalized least mean square (NLMS) adaptive digital filter (ADF). The proposed architecture achieves a constant and a short critical path without producing output latency. In addition, it retains the advantage of the NLMS, i.e., that the step size that assures the convergence is determined automatically.

1.4 Thesis Outline

This thesis work is organized in five chapter. The distribution of these chapters is as follows. Chapter 1 presents an overview of the problem, along with the statement of the problem, general and specific objectives, and a literature review of important work performed in the area of interest.

Chapter 2 describes a theoretical framework or background needed to follow the material presented in the rest of the thesis. This material is comprised of a general view of the definitions of power and energy; the product-delay and energy-delay metrics; different ways to reduce the power consumption of CMOS digital circuits; an explanation of the relaxed look-ahead transformation techniques; and finally, a survey of diverse power optimization techniques at the architectural and micro-architectural levels.

Chapter 3 starts with an overview of the least-mean-square algorithm, and the LMS adaptation algorithm. Next, it deals with the basics of the adaptive noise cancelling configuration. The following section explains the EEG model used as input to the system. The chapter continues with the complete architecture as pipelining, i.e.,
relaxed look-ahead transformation techniques, and retiming are applied. A comparison between the original system and the pipelined system is presented at the end of the chapter.

Chapter 4 deals with the Simulink-based design flow and a description of the phases, along with the tools utilized during each step. The chapter continues with the implementation of an adaptive digital filter from the algorithm design to the physical layout specifying as much as possible the process followed. The chapter ends with the methodology employed for the characterization of the basic functional units to estimate their power, delay, and energy.

Chapter 5 introduces the results from the characterization methodology from the previous chapter. This section offers a power and delay estimation for each of the basic functional blocks. The following section defines five adaptive filter architectures that were evaluated in terms of their performance and power consumption. The closing section presents an evaluation and discussion of results.

Finally, Chapter 6 contains the conclusions of the thesis and discusses future work.
Chapter 2

Principles of Low-Power CMOS Digital Design

2.1 Introduction

Since the early days of computing, the major concerns of the architects always were performance, area, and cost, while power consumption was considered of secondary importance. Nowadays, thanks to the ubiquitous computing needs of our society (notebooks, mobile phones, personal digital assistants, and audio-video players), low power computation is becoming increasingly important. This need for low power consumption has caused a major paradigm shift in which power dissipation is being considered as important as performance and area.

The motivations behind power consumption reduction are varied. In portable applications, such as mobile phones, the goal is to keep the lifetime of the battery long and the packaging costs low. In the case of notebooks, another parameter of concern is heat dissipation. For high-performance systems (workstations and servers) the goal of power reduction is of significant scope and encompasses the minimization of system cost, such as cooling, packaging, energy, and long-term reliability. An example of the importance of this goal is given by Chandrakant Patel, a researcher from HP, who calculated that future big data centers, 1000 racks or larger, might need 10 MW to run the computers and a further 5 MW just to keep them cool enough to operate [14]. These diverse—and sometimes conflicting—requirements impact power optimization in different orders, and demand that the architect sacrifice certain other parameters in
question, like area or performance; thus, making tradeoffs to depend on the overall goal of the application.

Most digital signal processing systems are implemented as integrated circuits using CMOS technology due to its superior robustness, which simplifies the design process and facilitates design automation, and its almost complete absence of power consumption in steady-state operation mode.

This chapter reviews the principles of low-power CMOS digital design. The chapter starts with the definitions of power and energy. The next section overviews the product-delay and energy-delay metrics. The following section explains different approaches to reduce the power consumption in CMOS digital circuits. For more information on this subject, the reader should consult [15], [3], [16]. Section 2.4 discusses the relaxed-look ahead transformation techniques. The last section presents a survey of power optimization techniques with areas of application.

### 2.2 Definitions of Power and Energy

In order to understand the power optimization techniques and their limitations, we must comprehend the difference between power and energy. It is important to understand that the techniques to reduce power do not necessarily minimize energy. Energy is the capacity to do work, whereas power is the rate at which work is done. In other words, the power consumed by a device is, by definition, the energy consumed per unit time. The energy ($E$) required for a given operation is the integral of the power ($P$) consumed over the operation time ($T_{op}$),

$$E = \int_0^{T_{op}} P(t) dt. \quad (2.1)$$

At the elementary transistor gate level, we can formulate total power dissipation as the sum of three major components: switching loss, short-circuit loss, and leakage loss. The following equation defines power consumption in a qualitative form

$$P = \alpha CV_{DD} V_{swing} f + I_{sc} V_{DD} + I_{leakage} V_{DD}. \quad (2.2)$$

The first term represents the dynamic power consumption caused by the charging and
discharging of the capacitance load on each gate’s output. It is proportional to the frequency of the system’s operation, $f$, the activity of the gates in the system, $\alpha$, the total capacitance seen by the gate’s outputs, $C$, and the voltages $V_{DD}$ and $V_{swing}$ (referenced to $V_{SS} = 0$), which in most cases the literature approximates them as equal. The second term is due to the direct-path short circuit current $I_{sc}$, which arises when both the NMOS and PMOS transistors are simultaneously active, conducting current directly from supply to ground. This $I_{sc}$ is dependent on the frequency of the commutations and a more detailed study of it is given in [3], [17]. Finally, the leakage current $I_{leakage}$, which arises from substrate injection and subthreshold effects, is primarily determined by fabrication parameters.

It is important to remember that the energy per operation is independent of the clock frequency. With this in mind, reducing the frequency lowers the power consumption but does not change the energy required to perform a given operation. Depending on the specifications, different measures must be considered. For instance, when studying supply line sizing, the peak power $P_{peak}$ is important. When addressing battery requirements or cooling, the average power dissipation $P_{av}$ is a more significant measure

$$P_{peak} = i_{peak}V_{supply} = max[p(t)]$$  \hspace{1cm} (2.3)

$$P_{av} = \frac{1}{T} \int_{0}^{T} p(t)dt = \frac{V_{supply}}{T} \int_{0}^{T} i_{supply}(t)dt,$$  \hspace{1cm} (2.4)

where $p(t)$ is the instantaneous power, $i_{supply}$ is the current being drawn from the supply voltage $V_{supply}$ over the interval $t \in [0, T]$, and $i_{peak}$ is the maximum value of $i_{supply}$ over that interval.

### 2.2.1 Power-Delay Product

The propagation delay and the power consumption of a gate are related—the propagation delay is mostly determined by the speed at which a given amount of energy can be stored on the gate capacitors. The faster the energy transfer (or the higher the power consumption), the faster the gate. For a given technology and gate topology, the product of power consumption and propagation delay is generally a constant. This product is called the power-delay product (or PDP), and can be considered as a quality measure for a switching device.
where $P_{av}$ is the average power dissipation and $t_p$ is the propagation delay of the critical path. The PDP presents a measure of energy, as is apparent from the units (W x s = Joule). Here, the PDP is simply the average energy consumed per switching event.

### 2.2.2 Energy-Delay Product

The scaling of $V_{dd}$ is beneficial from the energy point of view but may have serious side effects on the delay. This means that using the energy as the metric is not sufficient. A more relevant metric has been proposed [18] as it accounts for both energy and delay by using the product of the energy per operation and the delay per operation. This metric can be used as the basis for design optimization and comparison between different systems. The energy-delay product (or EDP) is defined as

$$EDP = PDP \cdot t_p = P_{av} t_p^2 = \frac{C_L V_{DD}^2}{2} t_p.$$  \hfill (2.6)

It is important to observe the voltage dependence of the EDP. For low supply voltages, the energy is minimum but the delay is not. Increasing the supply voltage decreases the delay but at the expense of the energy as shown in Figure\textsuperscript{1} 2.1. This indicates that an optimum operation point should exist as it can be determined from the energy-delay product.

### 2.3 Power Reduction in CMOS Digital Circuits

The dominant term in modern circuits is the switching component, which suggests that reducing the supply voltage is the most effective way to decrease power consumption due to being a quadratic term. Other means for reducing switching power consumption include: reduction of the voltage swing in all the nodes, reduction of the switching probability, and reduction of the load capacitance. This thesis assumes that the power

\textsuperscript{1}Figure taken from [16].
consumption in CMOS is dominated by the dynamic power resulting from the charging and discharging of the capacitance

\[ P = C_{\text{total}} V_{DD}^2 f, \]  

(2.7)

where \( C_{\text{total}} \) denotes the total capacitance of the circuit, \( V_{DD} \) is the supply voltage, and \( f \) is the clock frequency of the circuit.

### 2.3.1 Voltage Scaling

Reduction of the supply voltage is the key to achieving low-power operation since it plays an important role in the power dissipation of CMOS digital circuits as shown in Equation (2.7). However, when power supply reduction is implemented, several important issues must be addressed so that system performance is not sacrificed. One of these issues is the fact that since the core voltage is being reduced, the input and output signal levels of the circuit must be compatible with the peripheral circuitry, in order to maintain correct signal transmission. Thus, an important consideration is the increase of delay after a supply voltage reduction. A first-order approximation of the propagation delay \( t_p \) is given by

\[ t_p = \frac{C_{\text{crit}} V_{DD}}{k(V_{DD} - V_t)^2}, \]  

(2.8)
where $C_{\text{crit}}$ denotes the capacitance to be charged/discharged in a single clock cycle, i.e., the capacitance of the critical path, $V_{DD}$ is the supply voltage, and $V_t$ is the threshold voltage. The parameter $k$ is a function of technology parameters $\mu$, $\frac{W}{L}$, and $C_{ox}$.

For example, scaling the supply voltage from 5 to 3.3 V reduces the power dissipation by 56% with a propagation delay increase of 186%. For certain types of low-power applications this reduction in performance cannot be tolerated. Fortunately, there are architectural techniques to address this issue for systems where the throughput is more important than the speed. Parallel processing and pipelining are two of these techniques for lowering power consumption in systems where sample speed need not be increased [15].

### 2.3.1.1 Parallel Processing for Low Power

The single functional block shown in Figure 2.2 illustrates the architectural techniques to reduce power consumption. Both the input and output vectors are sampled through register arrays, driven by a clock signal CLK. The critical path in this logic block (at a voltage supply of $V_{DD}$) allows a maximum sampling frequency of $f_{CLK}$; in other words, the propagation delay is equal to $t_p = \frac{1}{f_{CLK}}$.

One of these architectural techniques is parallel processing which can decrease the power consumption of a system by allowing the supply voltage to be reduced. In an $L$-parallel system, the charging capacitance does not change while the total capacitance is increased by $L$ times (see Figure 2.3). In order to maintain the same rate, the clock period of the $L$-parallel circuit must be increased to $LT_{\text{orig}}$, where $T_{\text{orig}}$ is the propagation delay of the original circuit given by Equation (2.8). This means that $C_{\text{crit}}$ is charged in time $LT_{\text{orig}}$ rather than in time $T_{\text{orig}}$. In other words, there is more time to charge the same capacitance as shown in Figure 2.4. This idea can be utilized to reduce the supply voltage to $\beta V_{DD}$ [19].

The propagation delay of the original circuit is given by Equation (2.8) and the propagation delay of the $L$-parallel system is shown below

$$LT_{\text{orig}} = \frac{C_{\text{crit}}\beta V_{DD}}{k(\beta V_{DD} - V_t)^2}.$$  \hspace{1cm} (2.9)

Equating the propagation delays of both architectures, $\beta$ can be obtained. Once $\beta$ is
calculated, the power consumption of the $L$-parallel system is given by using Equation (2.10).

\[
P_{par} = (LC_L)(\beta V_{DD}^2)f \frac{f}{L}
= \beta^2 C_L V_{DD}^2 f
= \beta^2 P_{orig}
\] (2.10)

where $P_{orig}$ is the power consumption of the original system given by Equation (2.7). The power consumption of the $L$-parallel system has been reduced by a factor of $\beta^2$ in comparison with the original system. However, this reduction comes at the expense of increased area—the replication of $L$ times the logic of the original circuit plus the overhead of extra routing, resulting in a trade-off not suitable for area-constrained designs.

2.3.1.2 Pipelining for Low Power

Pipelining has been used to increase the sample speed of processing architectures [20]. In the case of low-power applications, speed is of second importance; therefore, pipelining can be used to maintain the throughput while reducing the supply voltage.
Consider an $M$-level pipelined system, where the critical path is reduced to $\frac{1}{M}$ of its original length and the capacitance to be charged in a single clock cycle is reduced to $\frac{C_{crit}}{M}$ (see Figure 2.5). It must be taken into account that the total capacitance of the circuit does not change. If the frequency is kept constant, only a fraction of the original capacitance, $\frac{C_{crit}}{M}$, is being charged or discharged in the same amount of time that was previously needed for the capacitance $C_{crit}$ (see Figure 2.6). This implies that the supply voltage can be reduced to $\beta V_{DD}$, where $\beta$ is a positive constant less than 1. The power consumption of the pipelined system will be

$$P_{pip} = C_L \beta^2 V_{DD}^2 f = \beta^2 P_{seq}. \quad (2.11)$$

Hence, the power consumption of the pipelined system has been reduced by a factor of $\beta^2$ when it is compared to the original system.

After pipelining the filter, the capacitance of the critical path is reduced, and if we wanted to maintain the same throughput—by operating the pipelined circuit at the same clock rate as the original system—we can equate the propagation delay of both
architectures. The propagation delay of the pipelined filter is given by

\[ T_{\text{pip}} = \frac{C_{\text{exa}}}{M} \beta V_{DD} \frac{\beta V_{DD} - V_t}{k} \]  

Once \( \beta \) is obtained, the power consumption of the pipelined system can be calculated using Equation (2.11). The power savings achieved after using pipelining are comparable to those obtained after using parallel processing with the advantage of much lower area overhead. While the power reduction is significant, it should be noted that the supply voltage cannot be reduced indefinitely since it is lower bounded by the process parameters. It is worth mentioning that pipelining is a special case of cutset retiming; more specifically, pipelining applies to graphs without loops. For more information on retiming, the reader should consult [21], [19].

### 2.3.2 Reduction of Switching Activity

The dynamic power consumption of CMOS digital circuits depends on the node transition factor \( \alpha \) (also called switching activity factor). This factor is the effective number of power-consuming voltage transitions experienced by the output capacitance per clock cycle. This node transition factor depends on the Boolean function performed, the logic family, and the input signal statistics. Switching activity in CMOS digital circuits can be reduced by algorithmic optimization, by architecture optimization, by proper choice of logic topology, or by circuit-level optimization.

Algorithmic optimization depends heavily on the application and on the characteristics of the data such as dynamic range, correlation, and statistics of data trans-
mission. The representation of data can have a significant impact on switching activity at the system level. An example is the use of sign-magnitude representation instead of the conventional two’s complement representation for signed data. A change in sign will cause transition of the higher-order bits in the two’s complement representation, whereas only the sign bit will change in sign-magnitude representation.

A different way to reduce switching activity is based on delay balancing and the reduction of glitches. In multi-level logic circuits, the propagation delay from one logic block to the next can cause spurious signal transitions, or glitches, as a result of critical races or dynamic hazards. If all input signals of a gate change simultaneously, no glitching occurs. A glitch can occur if input signals change at different times (see Figure 2.7).
Another very effective design technique for reducing the switching activity in CMOS logic circuits is the use of conditional, or gated clock signals. This approach is used to put the circuits in a sleep mode when they are idle. If certain logic blocks in a system are not immediately used during the current clock cycle, temporarily disabling the clock signals of these blocks will save switching power that would be otherwise wasted (see Figure 2.8).

2.3.3 Reduction of Switched Capacitance

Energy consumption is proportional to switching capacitance. This capacitance can be broken into two categories, the capacitance in dense logic (which includes the transistor parasitic and wire capacitances at the output of the gates) and the capacitances of busses and a clock network (which is mainly wire capacitance).

A comparison between the three logic families, static, dynamic, and the pass-gate (or CPL) [15] shows that the best logic family in terms of the energy required for a transition at a given amount of delay is the CPL. This comes at no surprise since
the CPL has the least transistor count for a given Boolean function implementation. This implies that the parasitic capacitances (gate oxide and source/drain diffusion capacitances) will be reduced. Therefore, CPL seems more attractive for low-power applications.

Transistor sizing is imperative for speed and power. Conventional design strategies focus on speed. The delay specifications were satisfied by sizing the transistors. For low-power design, the rule is to size the transistors on the critical path only such that the speed requirements are met. The transistors on the noncritical paths should be kept at the least possible dimensions. The name of the game for low-power design is to *use minimum-size transistor as much as possible* [16].

### 2.4 Pipelining Adaptive Digital Filters

As discussed earlier, pipelining is a helpful technique for lowering the supply voltage without decreasing throughput. It was also stated that pipelining is useful only for graphs without loops. Hence, pipelining of adaptive digital filters is an arduous task due to the presence of long feedback loops. Look-ahead transformation techniques [19] can be applied, but the resulting systems are not practical for hardware implementation [22]. Look-ahead transformations maintain the exact input-output behavior in frequency shaping filters, but in adaptive filter this input-output mapping is not that important since the coefficients continue to adapt until they converge. The relevant metrics in adaptive filtering are the misadjustment and the rate of convergence or adaptation time. The misadjustment provides a quantitative measure of the amount by which the final value of the *mean-squared error* (or MSE), averaged over an ensemble of adaptive filter, deviates from the minimum mean-squared error that is produced by the Wiener filter; in other words, the optimal solution [23]. The rate of convergence is defined as the number of iterations required for the algorithm, in response to stationary inputs, to converge “close enough” to the optimal solution.

In this section, the relaxed-look ahead transformation technique [19] is presented to help in the pipelining of the adaptive filters with little of no increase in hardware at the expense of minimum degradation in the adaptation behavior. The relaxed-look ahead transformation is based on certain approximations of the look-ahead technique. There are three forms of relaxed-look ahead including *product, sum, and delay*. 
2.4.1 Relaxed Look-Ahead Techniques

Consider the 1st-order time-varying recursion given by

\[ y(n + 1) = a(n)y(n) + u(n) \]  

(2.13)

with varying coefficients \(a(n)\). Using look-ahead, \(y(n + M)\) can be expressed in terms of \(y(n)\) as follows:

\[
y(n + M) = \prod_{i=0}^{M-1} a(n + M - 1 - i) y(n) \\
+ \sum_{i=1}^{M-1} \left[ \prod_{j=0}^{i-1} a(n + M - 1 - j) \right] u(n + M - 1 - i) \\
+ u(n + M - 1).
\]

(2.14)

The computation structure corresponding to Equation (2.14) is shown in Figure 2.9 for \(M = 4\). In general, the look-ahead transformation creates \(M - 1\) extra delays in the recursive loop which can be used to pipeline the multiply-add operation in the loop. For example, the iteration period of the structure in Figure 2.9 is now limited by \(T_m + T_a/4\). This transformation does not alter the input-output behavior. This has been achieved at the expense of the look-ahead overhead introduced by the 2nd term in the RHS of Equation (2.14), which is dependent on the level of pipelining \(M\). Depending on the constraints, this overhead may be unacceptable. However, under certain circumstances we can substitute approximate expressions to simplify the terms in the RHS of Equation (2.14). This is referred to as relaxed-look ahead.

2.4.1.1 Product relaxation

If the magnitude of \(a(n)\) is close to unity, then \(a(n)\) can be replaced by \((1 - \varepsilon(n))\), where \(\varepsilon(n)\) is close to zero. Then we have the following approximation

\[
\prod_{i=0}^{M-1} a(n + 1) \approx a(n + M - 1)^M = (1 - \varepsilon(n + M - 1))^M \\
= 1 - M\varepsilon(n + M - 1) \\
= 1 - M(1 - a(n + M - 1)).
\]

(2.15)
Furthermore, if \( u(n + i) \) is close to zero, then

\[
\prod_{j=0}^{i-1} a(n + M - 1 - j) u(n + M - 1 - i)
\]

can be approximated as

\[
u(n + M - 1 - i).
\]

Therefore, Equation (2.14) can be approximated as

\[
y(n + M) = (1 - M(1 - a(n + M - 1)))g(n) \\
+ \sum_{i=0}^{M-1} u(n + M - 1 - i). \tag{2.16}
\]

Hence, Equation (2.16) is the result of application of an \( M \)-level relaxed-look ahead with product relaxation to Equation (2.13). The computation structure corresponding to the 1st-order recursion in Equation (2.13) with 4-level product-relaxed pipelining is shown in Figure 2.10. Since \( a(n) \) is assumed to be close to 1, then \( \prod_{j=0}^{M-1} a(n+M-1-j) \) can also be approximated as \( a(n + M - 1) \). Then another approximation of Equation (2.14) can be obtained as illustrated below.
\[
y(n + M) = a(n + M - 1)y(n) + \sum_{i=0}^{M-1} u(n + M - 1 - i).
\] (2.17)

\[
M - 1 \prod_{i=0}^{M-1} a(n + M - 1 - i)y(n) + Mu(n).
\] (2.18)

If \( u(n) \) is also close to zero, then \( Mu(n) \) can be approximated by \( u(n) \) to obtain

\[
y(n + M) = \prod_{i=0}^{M-1} a(n + M - 1 - i)y(n) + u(n).
\] (2.19)

Thus, Equations (2.18) and (2.19) are the results of application of \( M \)-level relaxed-look ahead with sum relaxations to Equation (2.13). The computation structure of the 1st-order recursion with 4-levels of sum-relaxed pipelining is shown in Figure 2.11.

2.4.1.3 Delay relaxation

Consider the recursion
Figure 2.11: A 4-level sum-relaxed look-ahead pipelined 1st-order recursion

\[ y(n) = y(n-1) + a(n)u(n) \]  

(2.20)

and its M-level look ahead pipelined version

\[ y(n) = y(n-M) + \sum_{i=0}^{M-1} a(n-i)u(n-i). \]  

(2.21)

The delay relaxation involves the use of delayed input \( u(n-M') \) and delayed coefficient \( a(n-M') \) in Equation (2.21). This approximation is based on the assumption that the product \( a(n)u(n) \) is more or less constant over \( M' \) samples [24]. In general, this assumption is a reasonable approximation for stationary or slowly varying product \( a(n)u(n) \). Thus, Equation (2.21) is approximated as:

\[ y(n) = y(n-M) + \sum_{i=0}^{M-1} a(n-M' - i)u(n-M' - i). \]  

(2.22)

These relaxed-look ahead techniques may be applied individually or in combination to derive different architectures. Each of these architectures will have its own behavior, which will depend upon the approximations made and their validity. The relaxed-look ahead is a transformation technique in the stochastic sense, since the average output profile maintained while the input-output behavior has been modified.
2.5 A Survey of Power Optimization Techniques

A prediction made in 1965 by Gordon Moore, cofounder of Intel, stating that the number of transistors occupying a square inch of integrated circuit material would double each year since the invention of the integrated circuit has been maintained and still holds true today. This trend is expected to continue at least through the end of the decade. However, today’s researchers face the challenge of working within the physical limits of atomic structure for scaling transistors while managing both power and heat. Furthermore, frequency growth has dropped to 15 to 20 percent because of power limitations [25]. The challenge of the computer architecture community is to device innovative ways of delivering continuing growth in system performance while simultaneously solving the power problem. In response to these power and thermal challenges, researchers are aggressively delving into conventional and unconventional technologies. Instead of riding on the steady frequency progress of the past decade, improvements will be driven by integration at all levels in conjunction with hardware-software optimizations.

In this survey the focus is on the architectural and micro-architectural level of power optimization techniques. Topics like technology, logic, and system-level optimizations are beyond the scope of this paper.

2.5.1 Memory Systems

From the earliest days of computing, researchers have wanted unlimited amounts of fast memory. This has been addressed in part by utilizing a memory hierarchy and with help from the fast pace of semiconductor technology, which has increased the level of integration exponentially with time. In today’s industry, the cost per memory bit is extremely low, and the amount of memory available is no longer the main issue. Now, the major topics regarding memory systems are related to performance and power consumption. The memory system consumes a significant amount of power of the total power consumption of microprocessors [26]. Memory systems have two sources of power loss. The first source is represented by the frequency of memory accesses; the second source is derived from the leakage current represented in the power equation.

The on-chip L1 and L2 caches of the 21164 DEC Alpha chip dissipate 25% of the total power of the processor [27]. The StrongARM SA-110 processor from DEC, which targets specifically low-power applications, dissipates about 27% of the power in the
I-Cache [28]. In the Pentium Pro processor, the instruction fetch unit (IFU) and the I-Cache contribute 14% to the total power consumed [29].

The reason for the high-power consumption in the I-Cache subsystem is that the execution rate of a processor depends critically on the rate at which the instruction stream can be fetched from the I-Cache. The I-Cache should therefore be able to provide the data path of the machine with a continuous stream of instructions and has therefore very high switching activity. Bellas et al [30] are focusing on developing methods for reducing the energy dissipated in the on-chip caches since energy dissipated in caches represent a substantial portion in the energy budget of today’s processors. Furthermore, this portion is likely to increase in the near future. They propose a method that uses an additional minicache located between the I-Cache and the CPU core that buffers instructions that are nested within loops and are continuously otherwise fetched from the I-Cache. This mechanism is combined with code modifications, through the compiler, that greatly simplifies the required hardware, eliminates unnecessary instruction fetching, and consequently reduces the switching activity and the dissipated activity. Ko et al [31] present the characterization and design of energy efficient, on-chip cache memories. Their paper reveals that memory peripheral interface circuits and bit array dissipate comparable power; hence, to optimize performance and power in a processor’s cache, they propose a multidivided module (MDM) cache architecture to conserve energy in the bit array as well as the memory peripheral circuits.

While the bulk of the power dissipated is dynamic switching power, leakage power is also beginning to be a concern. Chipmakers expect that in future chip generations, as feature sizes shrink, the dominant component of this power consumption will be leakage. Kaxiras, Hu, and Martonosi [32] examine methods for reducing leakage power within the cache memories of the CPU by discussing policies and implementations for reducing cache leakage by invalidating and “turning off” cache lines when they hold data not likely to be reused. A similar approach is examined in [26], where according to Kim et al, during a fixed period of time, the activity in a data cache is only centered on a small subset of the lines. This behavior can be exploited to cut the leakage power of large data caches by putting the cold cache lines into a state preserving, low-power drowsy mode. They investigate policies and circuit techniques for implementing drowsy data caches.

In the case of embedded systems, one solution consists of mapping the most frequently accessed addresses onto the on-chip SRAM to guarantee power and performance
efficiency. This option is especially effective when memory access patterns can be pro-
filed and studied at design time. Benini, Macii, and Poncino [33] propose an algorithm
for the automatic partitioning of on-chip SRAM in multiple banks that can be indepen-
dently accessed. By utilizing a dynamic execution profile of an embedded application
running on a given processor core, they synthesize a multi-banked SRAM architecture
optimally fitted to the execution profile. Su and Despain [34] present a case study of
performance and power trade-offs in designing on-chip caches for the microprocessors
used in portable computing applications.

2.5.2 Buses

Interconnections heavily affect power consumption since it is the medium of most elec-
trical activity. Buses are a significant source of power loss, especially interchip busses,
which are often very wide. The standard PC memory bus includes 64 data lines and
32 address lines, and each line requires substantial drivers. A chip can expend 15% to
20% of its power on these interchip drivers [35].

According to Benini et al [36] the power dissipated by system-level busses is the
largest contribution to the global power of complex VLSI circuits. Therefore, the
minimization of the switching activity at the I/O interfaces can provide significant
savings in the overall power budget. They present encoding techniques suitable for
minimizing the switching activity of address buses by targeting the reduction of the
average number of bus line transitions per clock cycle. In a similar manner, Aghaghiri
et al [37] introduce an approach to decrease the switching activity of an address bus.
They present a new address bus encoding technique. Their method is based on the
limited weight encoding and transition signaling. The resulting code, ALBORZ, can
be made adaptive to make the required encoder/decoder hardware smaller and achieve
higher reduction in the switching activity.

Kumar, Zyuban, and Tullsen [38] examine the area, power, performance, and
design issues for the on-chip interconnects on a chip multiprocessor, attempting to
present a comprehensive view of a class of interconnect architectures. Their paper
shows that the design choices for the interconnections have significant effect on the rest
of the chip, potentially consuming a significant fraction of the real estate and power
budgets.

To increase the level of integration and the performance, system-on-a-chip is widely
deployed in today’s designs. In such designs, communication resources are allocated to connect the on-chip modules for data exchange. Hsieh and Pedram [39] propose a splitbus architecture to improve the power dissipation for global data exchange among a set of modules. Their proposed split-bus architecture can be extended to multi-way split-bus when a large number of modules are to be connected.

2.5.3 Processor

Chip-level design space includes two major options: how we trade power and performance within a single processor pipeline (core), and how we integrate multiple cores, accelerators, and off-load engines on chip to boost total chip-level performance [25]. In addition to fixing and scaling pipeline depth, additional enhancements to increase power efficiency at the microarchitectural level are possible. The computer architecture research community is exploring new architectures that are aware of power, energy, and thermal challenges and able to manage them dynamically while running applications. These architectures are described below and include multi-core microprocessors, clustered microarchitectures, and other power-optimized microarchitectures.

With single-core performance improvements slowing, multiple cores per chip can help continue the exponential growth of chip-level performance. This solution exploits performance through higher chip, module, and system integration levels, and optimizes for performance through technology, system, software, and application sinergies [25]. Kumar et al [40] propose a single-ISA heterogeneous multi-core architecture as a mechanism to reduce processor power dissipation. They assume a single chip containing a diverse set of cores that target different performance levels and consume different levels of power. During an application’s execution, system software dynamically chooses the most appropriate core to meet specific performance and power requirements. Their initial results demonstrate a five-fold reduction in energy at a cost of only 25% performance. Several microprocessors, including Alpha 21264 and POWER4 use a compacting latch-based issue queue design which has the advantage of simplicity of design and verification. The disadvantage is its high power dissipation. Different issue queue power optimization techniques that vary not only in their performance and power characteristics, but in how much they deviate from the baseline implementation are explored in [41]. Li and Martínez [42] examine the power-performance implications of running parallel applications on chip multiprocessors (CMPs). They develop an analytical model
that puts together parallel efficiency, granularity of parallelism, and voltage/frequency scaling, to establish a formal connection with the power consumption and performance of a parallel code running on a CMP. Zyuban and Kogge’s work [43] attempts to bring the power issue to the earliest phases of microprocessor development. They investigate power-optimization techniques of superscalar microprocessors at the microarchitecture level by identifying major targets for power reduction. Then, they develop an energy-efficient version of a multicluster microarchitecture that reduces energy in the identified critical design points with minimal performance impact.

Leakage power of execution units is a major concern in current and future microprocessor design. Hu et al [44] explore the potential of architectural techniques to reduce leakage through power-gating of execution units. Their paper focuses on hardware mechanisms, such as workload-driven, dynamic power-gating. They explain the microarchitecture and circuit-level design parameters that need to be considered in investigating the overall power savings potential in such an approach. Another paper that can be compared to this research is the one presented by Manne, Klauser, and Grunwald [29]. Their goal is to control speculation and reduce the amount of unnecessary work in high-performance, wide-issue, superscalar processors. They introduce a hardware mechanism called pipeline gating to control rampant speculation in the pipeline and present inexpensive mechanisms for determining when a branch is likely to mispredict, and for stopping wrong-path instructions from entering the pipeline; therefore, limiting speculation and reducing energy consumption. A different approach is the one presented by Srinivasan et al [45] where they introduce an optimization methodology that starts with an analytical power-performance model to derive optimal pipeline depth for a superscalar processor.

Recently, very long instruction word (VLIW) architectures have been proposed for high-performance embedded systems as an interesting alternative to more conventional CPUs, to balance performance with hardware complexity and scalability. An approach to the design of embedded VLIW processor architectures based on the forwarding (or bypassing) hardware, which provides operands from interstage pipeline registers directly to the inputs of the functions units is proposed in [46].
# 2.5.4 Dynamic Voltage Scaling

Dynamic frequency scaling (DFS) is a technique that seeks to reduce power consumption by changing the processor frequency based on the requirements of the executing application. For fixed-duration tasks, especially waiting, this usually results in a proportional reduction of energy use. For other tasks, because of the corresponding increases in execution time, DFS could either save or waste energy depending on second order effects. To guarantee energy savings, the processor voltage must be changed at the same time as the frequency. This technique is referred to as dynamic voltage-frequency scaling (DVFS), or as dynamic voltage scaling (DVS) [47].

There are many techniques that can be used to control the domain frequencies. Off-line algorithms may be best suited for applications which can be hand-tuned, which will exhibit run-time characteristics similar to the off-line analysis, and which are likely to repeat this behavior whenever they are executed. On-line algorithms may be best suited for situations where the applications that will be executed are not known or controllable. An on-line algorithm that reacts to the dynamic characteristics of the application to control domain frequencies is proposed in [48]. Hong et al [49] present a paper whose goal is to develop the design methodology for the low-power, core-based, real-time SOC based on dynamically variable voltage hardware. In their study they develop techniques that treat voltage as a variable to be determined, in addition to the conventional task scheduling and allocation. They also address the selection of the processor core and the determination of the instruction and data cache size and configuration so as to fully exploit dynamically variable voltage hardware.

Flautner’s approach [50] is somewhat different. The aim of his performance-scaling technique is to take advantage of power-saving features on processors such as the Intel XScale and Transmeta Crusoe that allow the frequency of the processor to be reduced with proportional reduction in voltage. He uses episode-length information combined with a predictor as the basis for an automatic mechanism to estimate the optimum performance-level for program execution.

# 2.5.5 Software

Software does not consume power, but the execution and storage of software requires energy consumption by the hardware. In this section we review some of the techniques used in different levels of software applications to reduce the power consumption at
diverse levels and on different components.

Power optimization at the architectural and software levels has attracted the interest of a number of researchers. A model that views power from the standpoint of the software that executes on a microprocessor and the activity that it causes, rather than from the traditional hardware standpoint is proposed in [51]. A brief review of compiler techniques for power minimization is presented in [52] and [53]. The approach presented in [30] relies on profile data from previous runs to select the best instructions to be cached. The compiler maximizes the number of basic blocks that can be placed in the L-Cache by determining their nesting and using their execution profile. The resulting hardware is very simple and most of the task is carried out by the compiler; thus, reducing the power requirements of the extra cache.

A different approach is given by [54] in which processor stalls can be used to increase the throughput by temporarily switching to a different thread of execution, or reduce the power and energy consumption by temporarily switching the processor to low-power mode. Shrivastava et al present code transformations to aggregate processor free time and use it to profitably switch the processor to low-power mode. Viswanath, Abraham, and Hunt [55] propose instruction-driven slicing, a technique for annotating microprocessor descriptions at the Register Transfer Level (RTL) in order to achieve lower power dissipation. The technique automatically annotates existing RTL code to optimize the circuit for lowering power dissipated by switching activity. Program code compression is an emerging research activity that is having an impact in several production areas such as networking and embedded systems. This is because the reduced-sized code can have a positive impact on network traffic and embedded system costs such as memory requirements and power consumption. A survey of code-size reduction methods is presented in [56].

2.5.6 Power Management Techniques in the Industry

This section examines the power optimization and management techniques that the industry is currently adopting. The description of the low-power techniques utilized in three different processors, the Intel Pentium M, the Transmeta Efficeon, and the IBM Blue Gene, is discussed.

The Pentium M carefully balances performance enhancing features with several powersaving features that increase the battery lifetime. It uses advanced power-aware
performance features including the innovative branch predictor, the dedicated stack manager, the micro-operation fusion, and the Intel Pentium M processor bus. To save energy, the Pentium M integrates several techniques that reduce the total switching activity, these include hardware for predicting idle units and inhibiting their clock signals, buses whose components are activated only when data needs to be transferred, and a technique called execution stacking which clusters units that perform similar functions into similar regions so that the processor can selectively activate the parts of the circuit that will be needed by an instruction. Also, the processor includes low leakage transistors in the caches. It supports an enhanced version of Intel’s SpeedStep technology that selfmanages voltage and frequency stepping [57].

Transmeta’s Efficeon processor saves power by means of its LongRun2 power manager. LongRun2 Technologies is a suite of advanced power management, leakage control and process compensation technologies that can diminish the negative effects of increasing leakage power and process variations in advanced nanoscale designs. This manager addresses these challenges with advanced algorithms, innovative circuits, process techniques, and software and manufacturing optimization methods. Further details are not known since they are proprietary. The Efficeon also saves power by shifting many of the complexities in instruction execution from hardware to software. The Efficeon relies on a code morphing software (CMS), a layer of software that reserves a small portion of main memory (typically 32 Mb) for its translation cache of dynamically translated x86 instructions into the processor’s native VLIW instructions. This reduces the on-chip area and its accompanying power dissipation [58].

Optimizing future supercomputing applications will depend on delivering the best performance for a given power budget. To deliver increasing aggregate performance for supercomputing workloads, the main challenge for today’s system engineers is addressing system power. For the Blue Gene/L system, a primary design constraint was power-performance efficiency in a high-density-computing system that would fit in the form factor of air-cooled racks in a standard machine room. The results on real-world applications from IBM’s engineers demonstrate that exploiting thread-level parallelism with lower-power cores offers significantly better power-performance characteristics than using higher-frequency cores with high power consumption. Additionally, exploiting data-level parallelism improves power-performance efficiency. The key to achieving application efficiency is to ensure good software parallelization efficiency [59].
2.5.7 Power-Performance Analysis Tools

Elevating power to a first-class constraint must be a priority early in the design stage when designers make architectural trade-offs as they perform cycle-accurate simulation. This presents a problem because designers can make accurate power determination only after they perform chip layout. However, designers can usually accept approximate values early in the design flow, provided they accurately reflect trends. For example, if the architecture changes, the approximate power figure should reflect a change in power in the correct direction [35]. Several research efforts are under way to insert power estimators into cycle-level simulators. Researchers at different locations [60],[61] have developed power estimators based on the SimpleScalar simulator. In [62] a complete system power simulator, called SoftWatt is presented. The tool models the CPU, memory hierarchy, and a low-power disk subsystem and quantifies the power behavior of both the application and operating system. The PowerTimer toolset [63] has been developed for use in early-stage microarchitecture-level power-performance analysis of microprocessors. The key component of the toolset is a parameterized set of energy functions that can be used in conjunction with any given cycle-accurate microarchitectural simulator. From the viewpoint of functional use, PowerTimer is similar to prior academic research tools such as Wattch [60], the most used research power-performance simulator within the academic community, and SimplePower [61].

In the embedded processors domain, there is SimBed [64], an execution-driven simulation testbed that measures the execution behavior and power consumption of embedded applications and RTOSs by executing them on an accurate architectural model of a microcontroller with simulated real-time stimuli. A simulator for memory systems is presented in [65]. The simulator is based on energy characterization of memory systems including buses, bus drivers, and memory devices by a cycle-accurate energy measurement technique.
Chapter 3

Filter Architecture

3.1 Introduction

Adaptive digital filters are successfully implemented in a variety of areas such as echo cancellation, adaptive control, voiceband modems, digital mobile radio, acoustic beam-forming, channel equalization, and speech and image processing. The LMS algorithm is generally the most popular adaptation technique because of its simplicity and ease of computation. Different architectures of LMS adaptive filters can be considered depending on the primary design constraint whether it is area, performance, or power.

This chapter discusses the design of the filter architecture. The first section gives an overview of the least-mean square algorithm, its basic components, and the equations that describe it. Section 3.3 introduces the adaptive noise canceling configuration for the adaptive filter. The following section offers the modeling of the input signals. In Section 3.5 we apply relaxed-look ahead transformation techniques to the original filter which allow us to pipeline the filter and give shape to the architecture. The final section presents the proposed architecture and a comparison between the original and the pipelined architectures.

3.2 Least-Mean-Square Algorithm

This section is based on the book by Simon Haykin [23], and its intention is to introduce the reader into the fundamental concepts of adaptive filtering, specially, the least-mean-square algorithm.
3.2.1 Overview of the LMS Algorithm

The least-mean-square (LMS) algorithm is a linear adaptive filtering algorithm that consists of two basic processes:

1. A filtering process, which involves (a) computing the output of a transversal filter produced by a set of tap inputs, and (b) generating an estimation error by comparing this output to a desired response.

2. An adaptive process, which involves the automatic adjustment of the tap weights of the filter in accordance with the estimation error.

Thus, the combination of these two processes working together constitutes a feedback loop around the LMS algorithm, as illustrated in the block diagram of Figure 3.1. First, we have a transversal filter, or F-block, around which the LMS algorithm is built; this component is responsible for performing the filtering process. Second, we have a mechanism for performing the adaptive control process on the tap weights of the transversal filter, hence the designation weight-update block or WUD-Block in Figure 3.1.

Details of the F-block component are presented in Figure 3.2. The tap inputs \( u(n), u(n-1), \ldots, u(n-N+1) \) form the elements of the \( N \)-by-1 tap-input-vector \( U(n) \), where \( N - 1 \) is the number of delay elements; these tap inputs span a multidimensional space denoted by \( \mathcal{U}_n \). Correspondingly, the tap weights \( w_1(n), w_2(n), \ldots, w_{N-1}(n) \) form the elements of the \( N \)-by-1 tap-weight vector \( W(n) \). The value computed for the tap-weight vector \( W(n) \) using the LMS algorithm represents an estimate whose expected value approaches the Wiener solution \( w_0 \) (for a wide-sense stationary environment) as the number of iterations \( n \) approaches infinity.

During the filtering process the desired response \( d(n) \) is supplied for processing, alongside the tap-input vector \( U(n) \). Given this input, the F-block produces an output \( (n \mid U_n) \) used as an estimate of the desired response \( d(n) \). Accordingly, we may define an estimation error \( e(n) \) as the difference between the desired response and the actual filter output, as indicated in the output end of Figure 3.2. The estimation error \( e(n) \) and the tap-input vector \( U(n) \) are applied to the control mechanism, and the feedback loop around the tap weights is thereby closed.

Figure 3.3 presents details of the weight-update block. Specifically, a scalar version of the inner product of the estimation error \( e(n) \) and the tap input \( u(n-k) \) is computed
for \( k = 0, 1, 2, \cdots, N - 2, N - 1 \). The result so obtained defines the correction \( \delta w_k(n) \) applied to the tap weight \( w_k(n) \) at iteration \( n + 1 \). The scaling factor used in this computation is denoted by \( \mu \) in Figure 3.3. It is called the \textit{step-size parameter}.

Earlier we pointed out that the LMS algorithm involves feedback in its operation, which therefore raises the issue of \textit{stability}. In this context, a meaningful criterion is to require that

\[
J(n) \to J(\infty) \quad \text{as} \ n \to \infty
\]

where \( J(n) \) is the mean-squared error produced by the LMS algorithm at time \( n \), and its final value \( J(\infty) \) is a constant. An algorithm that satisfies this requirement is said to be \textit{convergent in the mean square}. For the LMS algorithm to satisfy this criterion, the step-size parameter \( \mu \) has to satisfy a certain condition related to the eigenstructure of the correlation matrix of the tap inputs.

The difference between the final value \( J(\infty) \) and the minimum value \( J_{\min} \) attained by the Wiener solution is called the \textit{excess mean-squared error} \( J_{\text{ex}}(\infty) \). This difference represents the price paid for using the adaptive (stochastic) mechanism to control the tap weights in the LMS algorithm in place of a deterministic approach as in the method of the steepest descent. The ratio of \( J_{\text{ex}}(\infty) \) to \( J_{\min} \) is called the \textit{misadjustment}, which is a measure of how far the steady-state solution computed by the LMS algorithm is away from the Wiener solution. It is important to realize, however, that the misadjustment \( \mathcal{M} \) is under the designer’s control. In particular, the feedback loop acting around the tap weights behaves like a \textit{low-pass filter}, whose “average” time constant is \textit{inversely} proportional to the step-size parameter \( \mu \). Hence, by assigning a small value to \( \mu \) the adaptive process is made to progress slowly, and the effects of gradient noise on the tap weights are largely filtered out. This, in turn, has the effect of reducing the misadjustment.

### 3.2.2 LMS Adaptation Algorithm

In the LMS adaptive algorithm, a weighted sum of all the observations \([19]\)

\[
\hat{d}(n) = W^T(n - 1)U(n)
\] (3.1)

is used as an estimate of the desired signal \( d(n) \), where
Figure 3.1: LMS algorithm block diagram

\[ W^T(n) = [w_1(n), w_2(n), \cdots, w_N(n)] \]  \hspace{1cm} (3.2)

is the weight vector and

\[ U^T(n) = [u(n), u(n-1), \cdots, u(n-N+1)] \]  \hspace{1cm} (3.3)

contains the past and current input samples, where \( N \) is the filter order. The estimation error is the difference between the desired signal and the estimated signal

\[ e(n) = d(n) - \hat{d}(n) = d(n) - W^T(n-1)U(n). \] \hspace{1cm} (3.4)

In the \( n \)-th iteration, the LMS algorithm selects \( W^T(n) \), which minimizes the square error \( e^2(n) \). Therefore, the LMS adaptive filters consist of an FIR filter block (F-block) with coefficient vector \( W^T(n) \) and input sequence \( U(n) \), and a weight update block (WUD-block).

To derive the weight update algorithm, the derivative of \( e^2(n) \) with respect to \( W^T(n-1) \) is calculated as follows

\[ \Delta_{W^T} (e^2) = \frac{\partial e^2}{\partial W^T} = -2dU + 2W^T U \cdot U = -2(d - W^T U)U = -2eU. \] \hspace{1cm} (3.5)
The update of the weight vector is thus written as

$$W(n) = W(n-1) - \frac{1}{2} \mu \nabla W^T(e^2(n)).$$  \hfill (3.6)

Finally, we obtain

$$W(n) = W(n-1) + \mu e(n)U(n).$$  \hfill (3.7)

The system level diagram is shown in Figure 3.1, where in every iteration the error $e(n)$ (3.4) is computed by the filter block in Figure 3.2 and the weight vector (3.7) is updated by the WUD block in Figure 3.3.

### 3.3 Adaptive Noise Canceling

Typically, the elimination of a sinusoidal interference is achieved with a fixed notch filter tuned to the frequency of the interference signal. In order to eliminate the interference without distorting the signal of interest, a very narrow notch must be implemented. However, if the notch is very narrow, the center of the notch may not fall exactly over the interference. For the application at hand, the power line interference drifts slowly in frequency, since it is not always a constant 60-Hz frequency. A simple method proposed by Glover [66] can be used when a reference for the interference is available.
The block diagram of an adaptive noise canceling system is shown in Figure 3.4. The primary input $x(n)$ supplies an information-bearing signal $s(n)$ and a sinusoidal interference $p(n)$, which are uncorrelated with each other. The reference input $r(n)$ is a sinusoidal signal with frequency $f_r$ and phase $\phi$ which supplies a correlated version of the sinusoidal interference. The value of $f_r$ is set to the frequency of the interference $f_0$. In this method, the reference is adaptively filtered to match the interfering sinusoid as closely as possible, allowing them to be subtracted out. This reference is

$$r(n) = C \cos(2\pi f_r n + \phi) \quad (3.8)$$

and is applied to an $N$-stage tapped delay line. $C$ is a constant deterministic amplitude usually different from the interference amplitude. The outputs of the taps are weighted and summed to give the adaptive filter output $y(n)$ which is estimated to match the interference $p(n)$ at the primary input. Let $d(n)$ denote the desired signal, which in this case is equivalent to the primary input $x(n)$. The error signal $e(n)$ is defined as
the difference between the desired signal \( d(n) \) and the filter’s output signal \( y(n) = \hat{p}(n) \)

\[
e(n) = x(n) - y(n) \\
e(n) = s(n) + p(n) - \hat{p}(n) \\
e(n) = \hat{s}(n)
\]  

(3.9)

where \( \hat{s}(n) \) is an estimate of the noise-free EEG signal and \( y(n) = W^T(n-1)U(n) \). The LMS algorithm is designed to minimize an instantaneous version of the mean-squared error (or MSE) given by \( E\{|e(n)|^2\} = E\{|\hat{s}(n)|^2\} \).

Figure 3.4: Adaptive noise canceling system

### 3.4 Input Signals Models

To be able to simulate the architecture accurately and analyze its characteristics—the way it is going to behave in a given environment—we must understand the traits of the input signals. This section deals with the models used to simulate the input signals to the system.

#### 3.4.1 EEG Signal Model

Considerable motivation exists for the development of an adequate model for spontaneous electroencephalographic (EEG) activity, so that the features of spontaneous
EEG can be better interpreted in the quantitative EEG analysis. In addition, EEG signal modeling is important to achieve a better understanding of the physical mechanisms generating these signals and to identify the causes of EEG signals change [67]. Modeling can also be used for predicting the future neurological outcome and for data compression. Simulation based on EEG signal model can be used to better demonstrate the effectiveness of a certain quantitative analysis method or EEG feature extraction system [68].

In this thesis the Markov Process Amplitude (MPA) model described by [69] will be the foundation for generating the artificial EEG signals which will serve as one of the inputs to the system. Figure 3.5 shows the block diagram of the MPA model proposed.

\[
y(n) = \sum_{j=1}^{K} a_j(n) x_j(n) = \sum_{j=1}^{K} a_j(n) \sin(2\pi m_j n + \phi_j),
\]

(3.10)

where \(a_j(n)\) is the model amplitude of the first-order Markov process, \(m_j\) is the dominant \(j\)th frequency, \(\phi_j\) is the initial phase, and \(n\) is the time index. The next estimate of the model amplitude \(a_j(n + 1)\) is defined as

\[
a_j(n + 1) = \gamma_j a_j(n) + \xi_j(n),
\]

(3.11)
where $\xi_j(n)$ is a random increment of Gaussian distribution with zero mean and variance $(\sigma^2_j)$, and $\gamma_j$ is the coefficient of the first-order Markov process which must satisfy the condition $0 < \gamma_j < 1$ for stability.

### 3.5 Application of Relaxed Look-Ahead Techniques

As discussed in Section 2.4, relaxed-look ahead transformation techniques can be used to pipeline adaptive digital filters, and utilize that gain in throughput to scale down the supply voltage in order to obtain a low-power consumption. In this section, we will explain the application of the relaxed-look ahead transformation techniques to the LMS algorithm [70] for EEG signals.

To derive the pipelined architecture, we start with the serial LMS (or SLMS) equations and attempt to apply look-ahead directly. Equations (3.7) and (3.4) completely define the SLMS. To create $D_2$ delays in the WUDL, we apply a $D_2$-step look-ahead to Equation (3.7) and express $W(n)$ in terms of $W(n - D_2)$. We start with (3.7)

\[
W(n) = W(n - D_2) + \mu \sum_{i=0}^{D_2-1} e(n - i)U(n - i). \quad (3.12)
\]

In order to apply strict look-ahead, $e(n)$ needs to be expressed as a function of $W(n - D_2)$ and then substituted into (3.12). This would make the resulting equation very complex and not feasible for hardware implementation. This is why we take (3.12) as the starting equation.

#### 3.5.1 Delay Relaxation

The delay relaxation introduces $D_1$ in the error-feedback loop (EFL), as shown

\[
W(n) = W(n - D_2) + \mu \sum_{i=0}^{D_2-1} e(n - D_1 - i)U(n - D_1 - i). \quad (3.13)
\]

The second term on the RHS of the equation is a summation of past and present gradient estimates. As said before, this relaxation is valid based on the assumption that the estimates do not change much over $D_1$ samples. The hardware overhead of this equation is $N(D_2 - 1)$ adders, where $N$ is the number of the filter taps.
3.5.2 Sum Relaxation

For large values of \( N \) and/or \( D_2 \), this hardware overhead may not be acceptable. Therefore, we apply sum relaxation to the equation by taking \( LA \) terms

\[
W(n) = W(n - D_2) + \mu \sum_{i=0}^{LA-1} e(n - D_1 - i)U(n - D_1 - i),
\]

(3.14)

where \( LA \leq D_2 \). By substituting \( W(n - 1) \) from the above equation in (3.4) we obtain

\[
e(n) = d(n) - W^T(n - 1)U(n) \\
= d(n) - \left[ W(n - D_2 - 1) \\
+ \mu \sum_{i=0}^{LA-1} e(n - D_1 - i - 1)U(n - D_1 - i - 1) \right]U(n).
\]

(3.15)

If we assume that \( \mu \) is sufficiently small and replace \( W(n - D_2 - 1) \) by \( W(n - D_2) \)

\[
e(n) = d(n) - W^T(n - D_2)U(n).
\]

(3.16)

3.6 The Architecture

The pipelined architecture is completely described by Equations (3.14) and (3.16). The hardware overhead is now \( N(LA - 1) \) additions. The pipelined architecture is shown in Figure 3.6 where we can observe that \( D_1 \) can be retimed [21], and used to fine-grain pipeline [19] the hardware operators, i.e., the adders and multipliers. The latches \( D_2 \) at the output of the adder can be utilized to fine-grain pipeline that operator in an actual implementation. Table 3.1 presents a comparison between the serial LMS and pipelined LMS algorithms. Here, \( ta \) and \( tm \) represent the propagation delays of a two operand adder and multiplier, respectively; and \( N \) is the filter length. As can be observed, the critical path delay for the serial LMS increases proportionately with the filter length. With the pipelined LMS the idea is to maintain the critical path delay as short as possible by increasing \( D_1 \) and \( D_2 \) accordingly. It should be noted that \( D_1 \) and \( D_2 \) cannot be increased infinitely since this will cause a detriment to the behavior characteristics of the filter.
Table 3.1: Comparison of Serial LMS and Pipelined LMS algorithms

<table>
<thead>
<tr>
<th></th>
<th>Serial LMS</th>
<th>Pipelined LMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical path delay</td>
<td>((N+1)t_a+3t_m)</td>
<td>(t_m) (without fine-grain pipelining) &lt; (t_a) (with fine-grain pipelining)</td>
</tr>
<tr>
<td>Algorithm</td>
<td>(W(n)=W(n-1)+\mu e(n)U(n)) (e(n)=d(n)-W^T(n-1)U(n))</td>
<td>(W(n)=W(n-D_2)+\mu \sum_{i=0}^{L_A-1} e(n-D_1-i)U(n-D_1-i)) (e(n)=d(n)-W^T(n-D_2)U(n))</td>
</tr>
</tbody>
</table>

Figure 3.6: The pipelined architecture
Chapter 4

Design Flow and System Implementation

4.1 Introduction

*Application-Specific Integrated Circuits* are specialized circuit blocks or entire chips that can be designed specifically for a certain application or for an application domain. As these designs are custom-made, they can operate under more severe constraints than possible with standard IC parts. The long and expensive development process associated with their design has been for a long time its main disadvantage. However, this is changing lately, in part due to the improvement of the design methodologies and the increased use of automated synthesis tools, which has resulted in the proliferation of *fab-less* design companies that offer their designs as reusable blocks known as *intellectual property* blocks.

This chapter presents a design methodology highly integrated into a commercial CAD environment. An explanation of each of the phases in the design flow is offered, along with the corresponding tool which eases the process of developing an ASIC. The following section discusses and goes into the details about the implementation of a pipelined adaptive filter already described in the previous chapter. The last section deals with the characterization methodology used in the estimation of power, delay, and area for each of the functional blocks present in the design.
4.2 Design Styles

ASIC design starts with an initial concept of the required IC part. Early in this product conceptualization phase, it is important to decide the design style that will be most suitable for the design and validation of the eventual ASIC chip [71]. A design style refers to a broad method of designing circuits which uses specific techniques and technologies for the design implementation and validation. The design style determines the specific design steps to be carried out and the use of library parts for the ASIC. Design styles are classified into custom and semi-custom designs.

Custom designs involve the complete design to be hand-crafted in order to optimize the circuit for performance, area, or power for a given application. The semi-custom design style limits the circuit choices or primitives and used predefined blocks which cannot be further fine-tuned. This design style allowed the creation and proliferation of CAD tools for design and optimization which reduces the design times and facilitates the development. Figure 4.1 shows how semi-custom designs can be classified into two major classes: cell-based design and array-based design.

Cell-based designs can be based on standard-cell design, in which basic primitive cells are designed once, and are available in a library for each process technology or foundry used. Technology mapping is the process in which the design is mapped by the CAD tools to the cells contained in a library. Another way of using cell-based designs is by using cell generators to synthesize primitive building blocks from a functional or behavioral specification. This is the method used in this thesis to implement the system into an ASIC.

Array-based designs use a prefabricated matrix of non-connected components known as sites. These sites are wired together to perform a given function. Array-based circuits can be pre-diffused or pre-wired, also known as mask programmable and field programmable gate arrays (MPGAs and FPGAs). In this thesis work, FPGAs are utilized for circuit prototyping and verification purposes given their relative ease of design which leads to low cost and time overheads.

4.3 Design Flow

The design flow defines the approach used to take a design from an abstract concept through the specification, design, testing, and manufacturing steps. The waterfall model
has been the traditional model for ASIC development. In this model, the design goes through several steps while it is constantly refined at each level of abstraction.

The design process starts with the development of a specification, which may include requirements analysis, architecture design, and verification of the specification. This design is later coded at the register transfer level (RTL) in hardware description languages such as VHDL or Verilog. The functionality of the RTL is then verified against the initial specifications which are used as the golden model for verifying the design at every level of abstraction. The RTL is then synthesized into a gate-level netlist which is run through a timing verification which verifies that the ASIC meets the timing constraints specified. Afterwards, during the layout phase a floorplan for the chip is developed, in which to place the cells and route the interconnects, after which the chip is manufactured and tested. Figure 4.2 shows a typical ASIC design flow.

As can be seen, this design flow has its disadvantages; the main problem is that as the complexity of the system being designed increases, the design becomes more error prone. One explanation to this problem is that the requirements put upfront are not properly tested until a working prototype of the system is available, which in some cases is really late in the design flow. This leads to errors discovered late in the design process; therefore, the design must undergo through major redesign steps which involves costly reruns through the design flow. Another possible explanation is the fact that as the design goes from one level of abstraction to the next, a description of the design must be reentered. This represents an enormous overhead as the system...
Figure 4.2: Waterfall design flow for ASIC development

becomes increasingly complex; hence, more difficult to manage.

To address these issues, this thesis makes use of the design flow shown in Figure 4.3. The design flow is based on a Simulink environment and borrows some ideas from the *Chip-in-a-day* design flow [72]. The following sections give a brief explanation of the activities that are carried out in every phase of the design flow. An overview of the utilized tools in this thesis work is presented as well.

### 4.3.1 Algorithm Design

This is the first part of the design flow, where a system specification can be captured and analyzed. The algorithm of a design is represented in a graphical model which allows us to make decisions such as the number of pipeline stages, the number of the components in the data path, etc. The algorithm can be modeled in the form of a *signal-flow graph* or in a high-level language such as MATLAB. The functional specification is then simulated for a wide set of inputs to verify that the requirements are met.
4.3.1.1 Matlab/Simulink

MATLAB® is a high-performance language for technical computing [73]. It provides an integration between computation, visualization, and programming in an easy-to-use environment. Simulink® is an environment for multidomain simulation and model-based design [73]. It provides an interactive graphical environment and a customizable set of block libraries that can be used in the modeling and simulation of communications, controls, signal processing, video processing, and image processing.

The Simulink design environment was chosen since it provides an intuitive graphical description for both algorithm developers and hardware engineers, avoiding in this way, design re-entry which represents an immense overhead. For example, an algorithm can be entered in a graphical block form (i.e. signal-flow graph) providing an abstract representation of such algorithm. The modeling can be further expanded by providing real world data to the design such as the number of bits used for the representation of a module, the word-size of the data path, etc. while staying in a familiar and intu-
itive environment. After the design has been verified, the hardware engineers can use the Simulink model to generate an HDL description automatically, along with the test vectors utilized in the verification of the algorithm which can be used at refined levels of implementation.

4.3.2 RTL Model

After the algorithm design has been completed and verified against a set of specified requirements, the functional specification is converted automatically into a register transfer level (RTL) model. This RTL model uses register-level components like adders, multipliers, registers, etc. to represent the structural model of the design along with the interconnections in a technology-independent form. This RTL model is then simulated generally with event-driven simulation to verify the functionality and to obtain a rough estimate of the timing performance of the system. Once the RTL model has been thoroughly tested, it is used as the *golden model* to compare the results of the following phases against.

4.3.2.1 System Generator

Xilinx System Generator® for DSP is a high-level tool for designing DSP systems using FPGAs [74]. The tool provides system modeling and automatic HDL code generation from Simulink and MATLAB. This HDL code generation is suitable for use in an FPGA since it contains FPGA-specific primitives. The system modeling is done by utilizing the Xilinx blockset that contains functions for signal processing, arithmetic, memories, and digital logic. This blockset allows the construction of bit-accurate and cycle-accurate models of an FPGA circuit in Simulink. For verification and prototyping purposes, it allows the use of hardware co-simulation interfaces that make it possible to incorporate an FPGA directly into a Simulink simulation. Also, the tool provides a way to automatically generate a test bench in an HDL along with the test vectors utilized in the Simulink simulations which serve as a guide for performing simulations at a later phase. Figure 4.4 shows an example of the design environment in Simulink using System Generator and the Xilinx blockset.
4.3.3 Gate-Level Model

Once the RTL model satisfies the verification process, it is refined to the logic gate level using logic synthesis tools which implement the components with gates or combination of gates using a standard-cell library-based methodology. This phase also has a simulation part where functionality is tested along with other important constraints such as performance, area, and power. In this verification phase we start having a closer estimation of the timing performance of the design under test by using static timing analysis tools to identify the critical paths in the design.

4.3.3.1 LeonardoSpectrum

LeonardoSpectrum® is the tool utilized in this phase of the design flow. The tool offers design capture, VHDL and Verilog entry, register-transfer-level debugging for
logic synthesis, constraint-based optimization, timing analysis, and schematic viewing [75]. Leonardo is used in this thesis to automate the process of logic synthesis of the HDL code generated by System Generator after being modified and manually translated to technology-independent VHDL code. In this way, Leonardo understands the code and maps it to a specific ASIC technology producing a ready to place-and-route netlist. Also, Leonardo is the tool of choice to give an estimation of the delay of the critical path, as well as the number of gates contained in each module of the system.

LeonardoSpectrum works in three different modular levels of which Level 3 is the only one that supports ASIC logic synthesis. To target ASIC technologies, an available synthesis library for the process must be present in LeonardoSpectrum’s format (.syn). Since this was not the case, the XlibCreator Development Kit was used since it is a product that provides a development environment for ASIC libraries. This tool is designed to create Mentor Graphics ASIC synthesis libraries from a Synopsis .lib format. The synthesis library in .lib format must be provided by the ASIC vendor. Figure 4.5 shows the basic logic synthesis process as explained earlier.

![Logic synthesis process](image)

Figure 4.5: Logic synthesis process

Once the netlist is available in Verilog format, the next step is to import it into the ICFLOW® by using IC Studio® [75] and the HIT-Kit® [76] for switch-level simulation, power and area estimation, and physical layout. ICFLOW is the name Mentor Graphics gives to the design flow and the associated tools it sells for the design of nanometer ICs. IC Studio is Mentor’s integrated IC design environment which includes Design Architect IC (DAIC), IC Station, Calibre, Eldo, Mach TA, ModelSim, ADiT, among others. The HIT-Kit is a utility kit consisting of software programs and
libraries which contain full frontend (symbol, schematic, simulation models) and back-end (placement outlines, full layout) information for the development of digital, analog, and mixed signal circuits in a Mentor Graphics design environment. The process design kits included in the HIT-Kit cover the whole design phase from behavioral simulation, synthesis for digital parts, schematic capture, mixed-mode simulation, layout, verification, and backannotation.

4.3.4 Physical Layout

The last major section of the design flow examines the physical implementation of the design in silicon. The gate-level netlist is converted into a physical layout, by floorplanning the chip area, placing the cells, and routing the interconnections. The post-layout simulation is performed once the layout has been completed. This is done after extracting the parasitic capacitances from the layout and backannotating them to the original schematic or netlist. After the results from the simulations are satisfactory and conform to the system specifications, the layout is used to generate the set of masks required for chip fabrication.

4.3.4.1 IC Station

The IC Station tool suite is a Mentor Graphic’s software utilized to perform physical layout. It consists of several applications; however, the most important ones used in this thesis are IC Station SDL®, AutoCells®, ICassemble®, ICgraph®, ICrules® and ICtrace®. Since we already have a logic source (i.e. netlist), schematic driven layout (SDL) can be used to perform physical layout based on information from the netlist. This is the functionality provided by IC Station SDL. Autocells is an automatic placement and routing toll for large circuit designs that are comprised of standard cells [77]. ICassemble provides a set of features for floorplanning, top-level assembly, and interactive routing. ICgraph supports a set of editing functions for accurate polygon editing and it is useful when correcting the DRC errors. ICrules and ICtrace are used for design rules checking (DRC) and layout versus schematic (LVS) errors, respectively.

In order to understand the steps required to perform automated layout of a chip, Figure 4.6 displays the automated layout design flow [78].

Library creation is the process of creating a library of standard cells that will be used in the design. This library should contain all the cells in the layout, including
standard, block, external, etc. If this library is not created, the instantiation of layout
blocks that have been created in a bottom-up hierarchical approach will not be rec-
ognized by IC Station. **Floorplanning** is the process of estimating the chip area that
will be used for each standard cell or block in the design. A **floorplan** is a topological
structure comprised of rows and shapes used as guides for placing cells in design. **Place-
ment** is the process of placing standard cells and blocks into the floorplan. **Routing**
places and interconnects signal and power paths between standard cells and blocks.
**Compaction** minimizes the size of a completed layout. **Design verification and pattern
generation** are concerned with the determination of whether the layout conforms to
the design rules (DRC), corresponds to the logic source (LVS), performs satisfactorily
(PEX), and then transfer the mask layout to patterns that can be used to fabricate the
chip.

### 4.3.5 Simulation

After entering a design or translating a design through the phases of the ASIC design
flow, a verification phase must be enforced to determine if the design correctly imple-
ments the intended function or to discover whether the previous phase introduced any
errors or not. This verification and analysis process is achieved through simulation,
which evaluates how a design behaves. Simulation is a well understood process, and in this thesis we make use of three of them at different levels of abstraction and for different purposes.

Figure 4.7: Typical simulation process

4.3.5.1 ModelSim

ModelSim® is a verification and simulation tool for VHDL, Verilog, and mixed-language designs [75]. It is used for the pass/fail verification of the design under test at two levels of abstraction: RTL model, and gate-level model. Figure 4.7 represents the traditional simulation process which is applicable to the RTL model simulation. The HDL description is provided by System Generator and then manually translated to be technology-independent. The simulation input and output vectors and the test bench files are also provided by System Generator. The HDL description of the design is read into the simulator tool along with the input vectors. The simulator produces output vectors which are captured and evaluated against a set of expected values. If the output values match the expected values, the simulation passes; otherwise, it fails and the design must be corrected.

After synthesis takes place, the design is resimulated to ensure that the functionality has not been corrupted by the synthesis process. For the gate-level process, Figure 4.8 illustrates the verification process after synthesis.

The simulation inputs include a gate-level, technology-dependent version of the design, and the ASIC vendor simulation library. The simulation library contains a
model for each circuit in the library that describes the circuit behavior.

4.3.5.2 Eldo

Eldo® is an SPICE simulator for detailed and accurate analog simulation [79]. This tool is being used for the switch-level simulation of some of the smaller blocks in the system (i.e. adders and registers) to estimate their power and performance. Figure 4.9 shows the input files that must be provided for an Eldo simulation run and the output files that Eldo produces.

The SPICE-compatible netlist is provided by the netlisting capability of Design Architect IC. The simulation commands are supplied according to the measuring, plotting, and other commands necessary for the simulation run. The .cir file is Eldo’s main
control file and is generated by Design Architect IC when specifying the simulation commands for the design. The outputs are the simulation log file which includes results and error messages, and the .wdb file which is the simulation results file viewed with the EZwave waveform viewer.

4.3.5.3 Mach TA

Mach TA is a fast simulation tool utilized for timing analysis of analog and digital circuits. Mach PA is the power analysis tool of the product. These tools are used for the switch-level simulation and power estimation of the multiplier blocks present in the system. It was decided this way since Eldo could not handle such blocks at a reasonable speed. Mach TA takes as inputs SPICE netlists produced by DAIC, a subset of the simulation commands available in Eldo—which may be executed in a .do file—and test vector files (.tv).

4.4 System Implementation

This section deals with the application of the proposed design methodology to the pipelined architecture from Section 3.6. The intent is to illustrate a more explicit procedure by showing the exact steps followed to achieve an actual implementation. The design starts with the design entry in Simulink, and ends with the adaptive digital filter being realized as an intellectual property (IP) core ready to be used as part of a larger system.

4.4.1 Step 1: Design Entry

This first step describes the modeling of the algorithm in Simulink/MATLAB v7.2.0.232 (R2006a). The objective is to obtain a bit-true cycle-accurate model of the design by using the Xilinx Blockset.

Figure 4.10 presents a block-level view of the algorithm model. This is the pipelined implementation of the LMS algorithm with filter order of 8, $D_1 = 11$, $D_2 = 1$, and $LA = 1$. The functionality is implemented with Xilinx System Generator (XSG) blocks. The most used blocks are adders, multipliers, registers or delays, subtracters, and gateways, as can be seen from the figure. Also, every Simulink model that contains any element from the XSG library must contain at least one System Generator block.
where it is possible to specify how code generation and simulation should be handled. Each functional block from this library can be configured with user-defined parameters. Figure 4.11 shows the parameter dialog box for a multiplier where the user can modify these options. All the blocks are implemented from behavioral HDL description with fixed-point 2’s complement arithmetic in a $Q$ format of 16.14. The quantization is set to rounding and the overflow to saturate.

### 4.4.2 Step 2: Algorithm Simulation

In order to verify the functionality of the design, we must provide stimuli to the design and check the outputs. This is done in Simulink with the configuration shown in Figure 4.12. The configuration comprises the EEG model described in Section 3.4.1, an interference block, and a reference signal as inputs. The output is conditioned to allow the comparison of the signals. After the results are satisfactory, an FPGA emulation can be employed to allow the design to run in hardware. System Generator provides hardware co-simulation interfaces that make it possible to incorporate an FPGA directly into a Simulink simulation [80]; hence, accelerating the simulation. For this feature, the XUP Virtex-II Pro from Digilent was used. After installing the hardware co-simulation board with the files provided by the vendor, the configuration shown in Figure 4.13 was utilized. In this figure we can observe the configuration used to compare the behavior of the design in a software simulation against the hardware simulation.

### 4.4.3 Step 3: RTL Model

After verifying the functionality of the design in an FPGA, we proceed with the code generation which is provided by System Generator v9.1.01. The code generation is achieved by selecting HDL Netlist; checking the Create Testbench option; and clicking on Generate. The Create Testbench option is very important for the reason that it produces the test benches that will be used to simulate the design later in the design flow, along with the test vectors used in the Simulink simulation. The target FPGA part is not significant, as long as we choose to implement the code in behavioral form from each of the blocks in the design. In this block, we generate two sets of code one for each language (i.e. VHDL and Verilog). The reason for doing this is that ahead in the design flow we will simulate in ModelSim utilizing VHDL for the RTL
Figure 4.10: Simulink model of the pipelined LMS algorithm
model simulation since this is the language accepted by the synthesis tool. Then, the synthesis tool generates a netlist in Verilog format that must be verified in the gate-level simulation because the importing feature of Design Architect IC works only with this format of netlists.

Once we have the VHDL and Verilog descriptions of the system, we proceed with a manual translation of the code in order to get rid of some of the FPGA dialect present in the HDL description, which may obstruct the functionality of the synthesis tool when producing netlists for ASIC technology. Most of the FPGA-specific primitives can be removed from the code and will not affect the description because the blocks have been implemented in a behavioral format. Another advantage of the translation is that the description becomes easier to read and manage since System Generator creates an entity and architecture declarations for each module. In our case, all the adders implement the same function, as well as the multipliers; therefore, a single declaration is more than enough. In this way, the top-level structural description only instantiates the computational units, which are just a few after the translation. A complementary way to simplify the description is by eliminating the signals that are not used. For example, the adders and multipliers do not need a clock signal, unless they are being
fine-grain pipelined which is not the case here. Therefore, it is possible to remove the clock, clock enable, and clr signals of these units which would help later to make the routing process a little less painful. Appendix A contains the VHDL entity declarations and architecture bodies of the modules after translation, excluding the conv and clock packages and the top-level structural description which can be inferred from Figure 4.10. These packages are generated each time the code generation functionality of System Generator is executed; therefore, there is no need to include them here.

4.4.4 Step 4: RTL Model Simulation

As the description of the design has been translated and certain signals have been removed, the test benches generated by System Generator must be modified accordingly. ModelSim SE v6.3c is used in this part of the design flow. The design is compiled, and later simulated with the use of the test benches which compare the output signals with the expected results from the Simulink simulation. This is basically a pass/fail verification. The results from the testbench are shown in Figure 4.14. In this figure we can observe that the verification has been successful since no errors were found.

4.4.5 Step 5: Gate-level Model

The following step after verifying the functionality of the RTL model is to generate the gate-level model of the design. In this phase, LeonardoSpectrum v2007a_37 is introduced. Before we start with the actual logic synthesis, we have to go back to
Figure 4.5 to recall the inputs and outputs of the process. We already have the HDL description, and the design constraints, but we are missing the ASIC synthesis library. There are only two design constraints that Leonardo accepts: area and delay. In our case, we opted for area since the smaller the gates used, the smaller the transistor contained in them. If the transistors are smaller, then the capacitance being switched every cycle is reduced; therefore, the power consumption is decreased. The ASIC synthesis library is supplied by the vendor in the form of a Liberty formatted file (.lib). It is therefore necessary to convert this library into a format supported by Leonardo which must be a .syn file. Thankfully, Mentor Graphics provides a utility known as XlibCreator Development kit for ASICs [81]. XlibCreator v2005b is used, and the only two commands that were used to produce a usable ASIC library are the following:

```
sysgen c35_CORELIB.lib c35_CORELIB.lgn
lgen c35_CORELIB.lgn c35_CORELIB.syn
```

In order to use the c35_CORELIB.syn file produced, we must register it in Leonardo as explained in [82]. This process must be done only once, and the resulting library should be placed in a folder where every user of Leonardo can reach it. The c35_CORELIB.lib file can be obtained directly from the ASIC vendor which in this case is AMS.
Once the library is registered we can follow with the synthesis process. The basic process is as follows: we load a library; read the design files; then, we analyze them; pre-optimize them; and elaborate them into a RTL (generic-gate) design after which we optimize the modules and map them to a technology-dependent design. In Leonardo we can get an idea of the number of gates used for every module and the critical path of the system. The final step is to write the reports, and generate a netlist in Verilog format which will be the input to the gate-level simulation and the physical layout step. A TCL script automating the whole process is presented in Appendix D. The reader should consult [82] for a more detailed explanation of the process.

### 4.4.6 Step 6: Gate-level Simulation

After synthesizing, a gate-level simulation is performed to make sure that the functionality of the design is intact. In order to perform the simulation, we must compile the AMS simulation libraries; then, compile the gate-level verilog netlist and the test bench; and run the simulation. Again, this is just a pass/fail simulation.

The compilation of the AMS simulation libraries is straightforward as it is done like any other code compilation. The netlist is in Verilog format; therefore, the ASIC simulation library to be compiled should be in this format. The commands performed on the library are as follows

```
vlib c35_CORELIB
```
vmap c35_CORELIB c35_CORELIB
vlog c35_CORELIB.v -work c35_CORELIB

As in the case with Leonardo’s ASIC synthesis library, this library should be compiled only once and put on a network location where everyone can have access to it. To use the library, it is only needed to map the logical c35_CORELIB name to the physical location using `vmap`. After running the test bench, the results should match those from the RTL-model simulation as seen in Figure 4.14.

4.4.7 Step 7: Physical Layout

After the logic synthesis has completed, the resulting Verilog netlists are imported into AMS-specific place-and-route design flow [83]. Since we already have the top-level description of the system, the basic design methodology here is to import each and every single one of the modules declared in the netlist. Then layout, and verify them separately. Afterwards, instantiate them in the final layout with the top-level description. This methodology provides a relief when laying out and verifying the complete layout.

Once we create a project and start the design environment IC Studio, the first tool used is Design Architect IC v2006.1 [84] which is where we import the Verilog netlists, and convert them to Mentor Graphic’s EDDM format. This is achieved by creating a library and selecting it, after which we select `File → Import → Verilog`. In the open dialog box, we select `Schematic/Symbol` and the Verilog file to be imported. Also, we choose the `map file` supplied by AMS which is located at:

$AMS_DIR/mentor/c35/verilogin_cellmapfiles/c35b4_digital.cellmap

The only thing left to do is to create a .do file with the description shown below in it and select it from the `Advanced Options → Dofile` box.

```bash
$setup_partition(@sheet_count,80,1,'''','@prefix,@first,'''SG_SHEET''',\ @a,'''sg''',@all_bits);
```

This file sets schematic generation partitioning to partition the schematic onto 1 sheet, instead of the default of 80 instances per sheet. This eases the physical layout process considerably. After having all the `cells` imported in the project, we create
two viewpoints for each of the cells by opening the schematic view. Design Architect opens showing the schematic of the cells where we can modify the circuit if required to. Here, we select HIT-Kit Utilities → Create Viewpoint and in the dialog box we pick the right technology (c35b4), and viewpoint level which in this case are both device and apar level. The apar-level viewpoint is used for creating a layout cell; the device-level viewpoint will be utilized later for LVS verification. After doing this operations for every cell, we go back to IC Studio to make sure that the viewpoints have been generated.

The layout views can be created now from the apar-level viewpoints. This is done from the view window, where we create a layout view of type block with a connectivity source specified from the vpt_c35b4_apar design configuration. After creating the layout view, IC Station v2006.1 [78] opens showing two windows. The window to the left is the layout window; the window on the right is the logic window where we can see the schematic. The process and rules files are already set correctly.

To create the physical layout for each of the modules (adder, multiplier, registers, etc.), we attach the layout library TECH_C35B4/DIGITAL_LIBS by selecting File → Library → Attach. This file is provided by AMS and contains the layouts for each of the standard cells in the 0.35µm CMOS process. Then, we execute the following command:

\$sdl_auto()

In the prompt bar we choose Placement Method → Unplaced and click ok. Now, we open the Hierarchy Window from the File → Open menu. Here, we select the top level item and select Set XY where X and Y are set to 0 so the tool calculates the area by counting the number of instances.

To actually place the cells into the coreblock, we use AutoCells v5.13_1.1. We select the block, peek into it, and set it as the context. In the Place & Route palette, we choose Run in the AutoCells section. The AutoCells interface opens, and we set the Auxiliary PAR file to

\$AMS_DIR/mentor/ic_station/c35/autocells/Flip_Merge.PAR

the Alternate Technology Directory to

\$AMS_DIR/mentor/ic_station/c35/autocells/c35b4.dir
the *Alternate L Database* to

\$AMS\_DIR/mentor/ic\_station/c35/autocells/CORELIB.L

and the *Name Mapping File* to

\$AMS\_DIR/mentor/ic\_station/c35/autocells/CORELIB.map

We run AutoCells and when it finishes, we read the results in. Now we can clearly see the placement and routing of the blocks in the layout. We just go back to the top-level of the layout and snap the block to the grid using *Edit → Snap → To Grid*.

The DRC is performed by selecting the *ICrules* palette and clicking on *Check*. This will check the design and the geometry rules specified in the rules file. Most of the design errors are corrected by means of polygon editing with ICgraph. The explanation of every design rule is out of the scope of this thesis; therefore, it is recommended that the reader consult the design rules file provided by the vendor. The LVS check is done by using the *ICtrace(M)* palette and supplying the name of the source file, which is the device-level viewpoint created before and specifying EDDM as the source type. It is important to make sure that the correct layout ports are defined. This is achieved by placing the port name over the ports with text labels with layer *MXPIN* if the port is on layer *METX*. For example, if the name of the port is “VDD” and it is a shape drawn with layer *MET1*, then the label should read “VDD” and it should be drawn with layer *M1PIN*.

Now that every cell has its layout view, we can proceed with the physical layout of the top-level design. After we import the netlist, create the apar-level viewpoint, and create the layout view, we find ourselves in IC Station. The first thing we must do is create a layout library from the cells that we previously generated. If this library is not created, the layout cells of the adders, multipliers, and registers cannot be used. We select *File → Library → Create* and provide a library and layout library name. Then we edit it. In the *Edit Library* dialog box, we browse to the library where the cells are located and click ok. This library is read, and the *Cell List* is updated with the names of the cells. We highlight the ones we are interested in, and click on *Set Site/Cell Types*. We make sure that the *Cell Types* are set to *Block* and proceed. Now, the layout library is created, saved, and ready to be used.
We continue by attaching this new library to the design and executing the $sdl\_auto()$ command. We open the Hierarchy Window and observe that the modules and logic have been recognized. Then, we select the buffers and ties and create a hierarchy partitioning around them. After which, we run AutoCells on this block. The next step is to place all the modules on the layout and create a floorplan. After all the cells are placed, the routing can begin. For this phase, ICBlocks is used and we invoke it by clicking on All in the Auto Route section which is in the Place & Route palette. We set the options according to our needs, and specify level 1 (POLY1) as a restricted level for routing. The routing begins and if we have overflows, we have to route them. There is an option to compact the layout; therefore, reducing the final chip area. The final step would be to run DRC and LVS checks on the design and fix all the errors. Figure 4.15 shows a preliminary physical layout of the pipelined adaptive filter described. This figure closely resembles the configuration and position of the basic modules in the architecture shown in Figure 4.10. The large blocks in the upper and lower part of the layout are the multiplier modules. The delay registers vary in size due to the number of pipeline stages required. The adders are the smallest of the modules and can be identified since they do not have a visible name.

![Figure 4.15: Preliminary physical layout for the pipelined adaptive filter](image-url)
4.5 Characterization Methodology

Given the enormous complexity and the high number of transistors that the digital adaptive filter contains, a characterization methodology is presented to simplify the simulations and the estimates for power consumption, delay, and area. A great advantage to our favor is the fact that the adaptive filter architecture is highly regular in that the computational units are only a few and are instantiated several times but their behavior remains constant. For example, a 16-bit adder can be designed once and instantiated 10 times for a 10-tap filter. Since the complexity of the basic design units is very low (i.e. registers, adders, subtracters, multipliers), a full characterization over a range of word sizes is possible.

The approach for block-level characterization is shown in Figure 4.16. The methodology starts with the use of Simulink/System Generator to produce technology-independent HDL code for each of the basic design units for different word sizes from 8 to 16 bits. Then, LeonardoSpectrum produces technology-dependent gate-level netlists. Here we obtain a first estimation of the timing performance of the design by calculating the critical path. After the Verilog netlist is generated, the design is imported into Design Architect IC where we create a device viewpoint of the design and feed Eldo or Mach TA with the SPICE netlist. Here we obtain the power consumption and the delay estimation. The Verilog netlist is also an input to the physical layout tool IC Station where we obtain the area estimation.

![Block-level characterization diagram](image)

Figure 4.16: Block-level characterization
4.5.1 Power and Delay Estimation

The power estimation follows the same methodology as shown above. Estimates for power are refined with switch-level accuracy by simulating in Eldo or Mach TA. Figure 4.17 shows the circuit configuration utilized for the measurement of the power consumption and the propagation delay. As can be seen, 10fF capacitors are at the outputs and a 10 MHz toggling frequency is used with an input slope of 1 ns.

Figure 4.17: Circuit configuration used for power estimation

Switching-level characterization can be very time consuming especially if using Eldo, which gives more accurate results at the expense of speed of calculation. For the simulation of registers and adders, Eldo was used; for multipliers, Mach TA was used which is faster but, accordingly, less accurate. Power estimation is time consuming for another reason: the power consumption depends on the input vectors as well as the switching activity inside the block. An accurate estimation requires long simulations over a large number of cycles since to obtain a true average of the power consumption, all possible inputs must be exercised. This is virtually impossible; therefore, the inputs of the circuit were excited with 50 normally distributed pseudo-random vectors at the clock frequency previously stated. This was done for every single basic design unit for different word sizes at different supply voltages from 0.8 to 3.4V in steps of 0.2V. The product of the average of the current flowing through the main power supply and the supply voltage gave us the power consumption. To achieve this, the .MEAS AVG command is used. For the propagation delay we measured the time duration between the input crossing the 50% level of the supply voltage and the time at which the output reached the 50% level. The command used was:
In the end, it does not matter how accurate the simulations are, the numbers may change if the characterized blocks are placed in a different environment; therefore, this power and delay characterization should be understood as a rough estimate. Figure 4.18 shows the configuration used in Simulink for the creation of the random vectors. The random numbers have a mean of 0 and variance equal to 1 with initial seeds of 0 and 1.

The blocks were simulated before and after logic synthesis in ModelSim with the vectors described above. However, Eldo and Mach TA do not accept vectors in the format that System Generator provides them. These tools accept a format known as LSIM test vectors which is also a sort of test bench in that it provides the circuit stimulus and response. The MACH TA application can read and execute the LSIM format test vectors and compare the circuit responses with the test vector file. To address this conversion between formats we used two tools: ModelSim and VTRAN® from SourceIII. Figure 4.19 shows the methodology followed to perform the translation.

First, the test bench from System Generator along with the .dat files are provided to ModelSim. In ModelSim we use the `vcd dumpports` command which produces an extended VCD (EVCD) file with variable changes in all states and strength information and port driver data. The command will capture port driver changes unless filtered
Appendix B presents a script used for the automation of the compilation, simulation, and extraction the port data of a 16-bit adder. The EVCD file needs some reformatting that is achieved with VTRAN. A command file must be written to instruct VTRAN how the EVCD file should be processed. Appendix C illustrates the script used for the same 16-bit adder. The output is an LSIM-formatted file ready to be used as input to an Eldo or MACH TA simulation with the `.tvinclude` command. The same LSIM file is used to perform verification for specific modules.

### 4.5.2 Area Estimation

The area estimation methodology starts by importing the Verilog netlist into Design Architect IC and creating an apar-level viewpoint. This viewpoint allows us to automate the physical layout process by creating a layout view from IC Studio. Once IC Station is open, we have the layout window on the left and the source logic window on the right. The next step is to attach a layout library for the standard cells to be recognized. After that, we execute the `$sdl_auto()` command and in the Placement Method section we select *unplaced*. Then, we open the hierarchy window from *File → Open → Hierarchy Window* and verify that the standard cells have been recognized. Now, we select all core cells and in the *Plan & Place Palette* we click on *PrtHi*; we name the new core block and we place it on the layout by clicking on *Plc*. We move the cursor into the layout window and place the block. In the hierarchy window, we select the block, and by making the layout window active, we click on *Draw* in the *Place & Route Palette*. We draw a rectangle as outline for the layout box of this block. The only thing left to do is to measure the block with the *Add Ruler* button. The
product of the base and height is the area. Figure 4.20 shows how to use the rulers to measure area.

Figure 4.20: Measuring area with the ruler
Chapter 5

Results

5.1 Introduction

This chapter presents the results obtained from simulations in detailed form utilizing the methodologies described in the previous chapter. The first section introduces the results from the characterization of the basic units such as multipliers, adders, and registers. These results are later used as the basis from which power consumption, performance, and area of the architectures are estimated. The second section defines a set of adaptive filter architectures. The last section presents an evaluation of the performance in terms of convergence speed and misadjustment. Also, an estimation of power and energy for each of the architectures is introduced along a comparison of the power savings obtained.

5.2 Characterization of the Basic Units

This section presents the results from the characterization methodology from Section 4.5 applied to the basic units that conform the adaptive filter architectures. These units are the multipliers, adders, and registers. The subtracters were omitted here given their resemblance to the adders in terms of area, delay, and power consumption. Also, given that each architecture makes use of only one of them in their entire structure, it would be not practical to perform an entire characterization.

As stated in Chapter 4, the results presented here were simulated in Eldo or Mach TA, depending on the circuit complexity. For example: the adders, and registers were
simulated in Eldo, whereas the multipliers were simulated in Mach TA. This was in order to obtain a reasonable simulation time since Mach TA is faster than Eldo at the expense of accuracy.

The configuration for the simulations was already given by Figure 4.17 for the case of a 16-bit multiplier. As we can observe, at the outputs we connected 10fF capacitors. The toggling frequency was 10 MHz with an input slope of 1 ns. The results are presented for each block for supply voltages from 0.8 to 3.4 V in steps of 0.2 V; the intermediate voltages are calculated by linear interpolation. For the average power consumption calculation, Eq. (2.4) was used. The propagation delay was measured as the time duration between the input crossing the 50% level of the supply voltage and the time at which the output reached the 50% level. LeonardoSpectrum provided the critical path. The power-delay product was obtained using Eq. (2.5). All the multipliers and adders presented here were implemented with quantization set to rounding and overflow set to saturation. The technology of choice to implement the basic units is the 0.35 \( \mu \text{m} \) process from AMS where BSIM3 models are used for the derivation of the results.

All the simulations were run on a Sun Blade 1500 workstation with a 1.5-GHz UltraSPARC IIIi processor with 2 GB of DDR memory. The operating system used was Solaris 9.

5.2.1 Multipliers

Multiplication is an important and fundamental function in arithmetic operation. They usually contribute significantly to the time delay and take up a great amount of the silicon area available for the system. Also, their contribution to the power consumption is important and this must be taken into account when planning the power budget for a DSP system.

Figure 5.1 shows the power consumption of the multiplier block for different word sizes. The toggling frequency is 10 MHz. For different frequencies, the power consumption curves can be linearly scaled according to the needs without much hassle and distortion of the curves. The maximum power consumption—which is given at 3.4 V—for a 16-bit multiplier is around 3500 \( \mu \text{W} \); for an 8-bit multiplier, the delay is 600 \( \mu \text{W} \).

The propagation delay of the critical path for the multipliers is given in Figure 5.2.
where the delay is a function of $V_{DD}$. We can observe that the propagation delay starts to increase dramatically as the supply voltage approaches $2V_t$, where the typical values of the threshold voltages of the nMOS and the pMOS transistors are $V_{t,n} = 0.50V$ and $V_{t,p} = -0.65$ with minimums of $V_{t,n} = 0.40V$ and $V_{t,p} = -0.55$. The minimum delay for a 16-bit multiplier was 36 ns at 3.4 V. The delay for an 8-bit multiplier at 3.4 V was around 20 ns.

![Figure 5.1: Power consumption of a multiplier block for different word sizes](image)

The power-delay product is an estimation of the energy consumed by the block during a switching event. Figure 5.3 shows the average energy consumed by the multiplier block during a switching event.

Table 5.1 exhibits the area and number of transistors for each multiplier in terms of the word size. The CMOS technology used was c35b4 which is a 0.35-µm 4-metal-layer technology from AMS. A fully automatic place and route approach was used as indicated in the previous chapter.

### 5.2.2 Adders

Addition is an obligatory operation that is crucial to processing the fundamental arithmetic operations [86]. It is by far the most frequently used operation in general-purpose
Figure 5.2: Propagation delay of a multiplier block for different word sizes

Figure 5.3: Power-delay product of a multiplier block for different word sizes
Table 5.1: Area and number of transistors of a multiplier block

<table>
<thead>
<tr>
<th>No. of Bits</th>
<th>Area</th>
<th>No. of Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>26544 µm²</td>
<td>2140</td>
</tr>
<tr>
<td>10</td>
<td>42091 µm²</td>
<td>3326</td>
</tr>
<tr>
<td>12</td>
<td>59088 µm²</td>
<td>4814</td>
</tr>
<tr>
<td>14</td>
<td>81304 µm²</td>
<td>6574</td>
</tr>
<tr>
<td>16</td>
<td>103895 µm²</td>
<td>8618</td>
</tr>
</tbody>
</table>

systems and in application-specific processors.

The power consumption estimation is given in Figure 5.4. Again, this characterization was done for supply voltages from 0.8 to 3.4 V and a toggling frequency of 10 MHz. If we compare the values of these power estimates against those of the multipliers, we can observe that the power consumption of the 16-bit multiplier is around 30 times that of the 16-bit adder at 3.4 V which gives us an idea of the size and capacitance of the multiplier.

Figure 5.5 shows the propagation delay of the critical path of the adder. This propagation delay is the carry ripple step which involves the propagation from the least-significant bit to the higher bit position.

![Figure 5.4: Power consumption of an adder block for different word sizes](image)

Figure 5.4: Power consumption of an adder block for different word sizes
Figure 5.5: Propagation delay of an adder block for different word sizes

An energy estimation is provided in Figure 5.6. This figure displays the energy consumption of the adders for different sizes and at different supply voltages. It is worth remembering that the average energy consumed is independent of the toggling frequency. The ratio of energy between the 16-bit adder and the 8-bit adder is not significant as it is close to two. This tells us that the difference between the number of transistors, or charging capacitance, between one and the other is not large, as can be observed from Table 5.2. This table shows what we stated before. The growth of the carry ripple chain is one of the causes for such an increase in the number of transistors. Another cause is the saturation and rounding logic added to handle the increase in the number of bits.

Table 5.2: Area and number of transistors of an adder block

<table>
<thead>
<tr>
<th>No. of Bits</th>
<th>Area</th>
<th>No. of Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4232 $\mu m^2$</td>
<td>316</td>
</tr>
<tr>
<td>10</td>
<td>5153 $\mu m^2$</td>
<td>368</td>
</tr>
<tr>
<td>12</td>
<td>5892 $\mu m^2$</td>
<td>444</td>
</tr>
<tr>
<td>14</td>
<td>7031 $\mu m^2$</td>
<td>520</td>
</tr>
<tr>
<td>16</td>
<td>8179 $\mu m^2$</td>
<td>596</td>
</tr>
</tbody>
</table>
5.2.3 Registers

The register blocks were implemented as simple D-type flip flops using AMS’ HIT-Kit. Figure 5.7 presents the power consumption estimation when running at a clock frequency of 10 MHz. If we observe carefully, we can notice that the power consumption for these registers scale linearly with the number of bits used. For example, the power consumption of the 16-bit register doubles that of the 8-bit register since it has twice the number of transistors.

Figure 5.8 displays the single propagation delay for all register sizes. The delay behaves in this way for all registers since they only have one level of logic, i.e. one D-type flip-flop. For practical matters, this propagation delay is often neglected, and will not be considered here in the future sections.

The energy consumed by the registers is presented in Figure 5.9. For example, if we define the maximum clock frequency at which a 16-bit register can be operated at 3.4 V as \( \frac{1}{2 \times T_{pd}} \), the power consumption can be approximated by linearly scaling the power curve to this frequency. Since the delay for this register at this voltage is 0.477797 ns, the power consumption would be around 9.33 mW. Using the equation for the PDP, the energy per operation for a 16-bit register would be close to 4.46 pJ as can be seen in Figure 5.9.

Figure 5.6: Power-delay product of an adder block for different word sizes
Figure 5.7: Power consumption of a register block for different word sizes

Figure 5.8: Propagation delay of a register block for different word sizes
Table 5.3 indicates the number of transistors and area for each register size. Again, this structure is highly regular and the number of transistors scales linearly with the number of bits. The area estimation is different since routing and interconnections have to be considered in the layout. Therefore, the area of the 16-bit register more than doubles that of the 8-bit register.

Table 5.3: Area and number of transistors of a register block

<table>
<thead>
<tr>
<th>No. of Bits</th>
<th>Area</th>
<th>No. of Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2715 , \mu m^2$</td>
<td>208</td>
</tr>
<tr>
<td>10</td>
<td>$4035 , \mu m^2$</td>
<td>260</td>
</tr>
<tr>
<td>12</td>
<td>$4117 , \mu m^2$</td>
<td>312</td>
</tr>
<tr>
<td>14</td>
<td>$5307 , \mu m^2$</td>
<td>364</td>
</tr>
<tr>
<td>16</td>
<td>$7031 , \mu m^2$</td>
<td>416</td>
</tr>
</tbody>
</table>
5.3 Architectures

In this section we define five different adaptive filter architectures that were compared in terms of their performance and power consumption. These architectures are defined by the equations presented in Table 3.1 which were explained in Chapter 3. For the sake of convenience and to provide a common ground, we considered 8-tap filters implemented with block units with a word size of 16 bits.

The first of the aforementioned architectures is the Serial LMS architecture (SLMS8). The SLMS8 architecture is shown in Figure 5.10. This is the simplest of the architectures and is being used as the benchmark all other architectures must be compared against.

Figure 5.11 presents one of the Pipelined architectures considered. The design was pipelined with $D1 = 1$, $D2 = 1$, and $LA = 1$. We named this architecture PLMS8 A.

The PLMS8 B is a pipelined architecture described by the equations from Section 3.5. For this configuration, we chose $D1 = 11$, $D2 = 1$, and $LA = 1$ such that the critical path is determined by one multiply operation.

A more complex design is the one shown in Figure 5.13. The architecture has been pipelined with a great array of delay registers such that the addition and multiply operations can be fine-grain pipelined in an actual implementation. This was done in
Figure 5.11: Pipelined LMS architecture with D1=1, D2=1, and LA=1

Figure 5.12: Pipelined LMS architecture with D1=11, D2=1, and LA=1
this way to study the behavior of such pipelining approach and its implications on the performance of the architecture.

![Figure 5.13: Pipelined LMS architecture with D1=42, D2=3, and LA=1](image)

The last architecture is quite similar to the one described before (PLMS8 C). Figure 5.14 shows the \textit{PLMS8 D} design. The difference between this configuration and the previous one is the inclusion of new hardware in order to improve the architecture performance.

5.4 Evaluation and Discussion of Results

5.4.1 Performance Evaluation

The configuration utilized for the simulation and performance evaluation of the architectures is presented in Figure 4.12. The algorithm described by Equations (3.4), (3.7), (3.14), and (3.16) was simulated with inputs generated by the EEG model from Section 3.4.1 and a simulated power line interference. The performance of the architecture was evaluated in terms of the convergence speed and misadjustment.

The stationary Markov process amplitude EEG model described by Equations (3.10) and (3.11) was used to simulate an EEG signal. To generate this signal, \( K = 2 \) dominant frequencies located at 3 Hz and 12.5 Hz were established. The model
parameters were set to $\gamma_1 = 0.98$, $\gamma_2 = 0.99$, and $\sigma_1^\xi = \sigma_2^\xi = 0.01$. The MSE presented in the following results is obtained as

$$MSE = |s - \hat{s}|^2,$$  \hspace{1cm} (5.1)

where $s$ and $\hat{s}$ are $N$-dimensional vectors of the interference-free and estimated signal. For the results presented here $N = 3000$.

The results shown in Figure 5.15 are for a normalized EEG signal to [-1, 1] and a power line interference signal with amplitude $A = 0.1$ with a frequency of $f_0 = 60Hz$. The step-size parameter is kept constant at $\mu = 0.02$. The experiment is used to analyze the rate of convergence and the misadjustment of the adaptive filter architectures described in the previous chapter.

As can be seen from the figure, the SLM8S and the first two pipelined architectures maintain a close resemblance in terms of the convergence speed (in number of iterations) and misadjustment, as the architectures converge after approximately 1500 iterations at practically the same misadjustment.

The convergence speed of the PLMS8 C architecture is slower and with a much
larger MSE. This is due to the inclusion of latches in the weight-update loop. However, the slower convergence of this architecture can be rectified by using sum relaxation as explained in Chapter 3 with $LA = 2$. Given that this implies an increase in the computation time of the error-feedback loop, we need to increase the value of $D1$ from 42 to 45. We called this architecture PLMS8 D. As was noticed before with PLMS8 A and B, increasing $D1$ hardly affects the convergence speed. The MSE plot for the PLMS8 D shows the exceptional improvement in the rate of convergence and misadjustment. Figure 5.16 presents a different view of the same MSE plots for all architectures. The purpose of this figure is to understand the difference in MSE values for each of the architectures by overlapping the signals. Figure 5.17 shows a closer look at the beginning of the adaptation behavior of the architectures in order to realize the latency present due to the inclusion of $D1$ latches. The delayed response of the pipelined architectures is clearly visible.

Figure 5.15: Comparison of the filter performance for different architectures
Figure 5.16: Comparison of MSE for different architectures

Figure 5.17: Comparison of the filter latency
5.4.2 Power Estimation

In Chapter 2 we stated that voltage scaling was the key to achieve low power consumption. We also established that reducing the power supply voltage, inevitably increases the propagation delay as is demonstrated by Equation (2.8) and the characterization results previously presented. The same chapter introduced two architectural techniques that allowed voltage scaling without affecting the performance. Pipelining for low power operation is the technique applied to the architectures reviewed in the previous section.

Let’s assume that the SLMS8 architecture needs to run at its maximum allowable sample speed. In this case, we are considering a maximum possible rate of \( f_{\text{max}} = \frac{1}{2\tau_{pd}} \). The propagation delay of the critical path of the SLMS8 architectures is determined by 9 adders and 3 multipliers (remember that a subtracter is being equated to an adder). Therefore, the \( T_{pd_{SLMS8}} = 230.73\text{ns} \) at 3.4 V. If we were interested in maintaining the same clock speed, we could pipeline the SLMS8, and scale down the supply voltage to a value that sustains the same propagation delay present before the pipelining.

The first pipelined architecture introduced, PLMS8 A, pipelines the serial architecture in such a way that halves the propagation delay. This allows the design to run at a supply voltage of 1.95 V which preserves the propagation delay; hence, the clock frequency.

The PLMS8 B architecture is pipelined such that the propagation delay is constrained by the delay of a multiply operation. The delay of a multiplier, as seen in Figure 5.2, is 32.96 ns at 3.4 V. If we decrease the value of the supply voltage to 1.20 V, the propagation delay of the filter becomes 236.17 ns which is close enough to the \( T_{pd_{SLMS8}} \).

Designs PLMS8 C and D have a propagation delay of \( T_{pd_{PLMS8C,D}} = \frac{T_{pd_{MULT}}}{6} \). Therefore, their supply voltage is calculated using linear interpolation from Figure 5.2.

Table 5.4 presents a power, energy, and area comparison for the five architectures already described. All the architectures are running at approximately the same clock frequency according to their propagation delay. The power consumption reported in this table has been scaled down to the clock frequency used in these examples from a 10 MHz frequency used in Figure 5.1. Also, the area is not considering routing and interconnections.

Design SLMS8 is running at a supply voltage of 3.4 V and its average power consumption is 13660.98 µW. The energy per operation is given by 3151.99 pJ, and the
Table 5.4: Power and energy comparison for five adaptive filter implementations

<table>
<thead>
<tr>
<th></th>
<th>Delay</th>
<th>$V_{DD}$</th>
<th>Power</th>
<th>PDP</th>
<th>Area</th>
<th>Power Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLMS8</td>
<td>230.73 ns</td>
<td>3.40 V</td>
<td>13660.98 $\mu$W</td>
<td>3151.99 pJ</td>
<td>2051761 $\mu$m$^2$</td>
<td>0.00%</td>
</tr>
<tr>
<td>PLMS8 A</td>
<td>229.19 ns</td>
<td>1.95 V</td>
<td>3514.30 $\mu$W</td>
<td>805.46 pJ</td>
<td>2079885 $\mu$m$^2$</td>
<td>74.27%</td>
</tr>
<tr>
<td>PLMS8 B</td>
<td>236.17 ns</td>
<td>1.20 V</td>
<td>1055.7 $\mu$W</td>
<td>249.32 pJ</td>
<td>2501745 $\mu$m$^2$</td>
<td>92.27%</td>
</tr>
<tr>
<td>PLMS8 C</td>
<td>237.49 ns</td>
<td>0.85 V</td>
<td>535.41 $\mu$W</td>
<td>127.16 pJ</td>
<td>3936069 $\mu$m$^2$</td>
<td>96.08%</td>
</tr>
<tr>
<td>PLMS8 D</td>
<td>237.49 ns</td>
<td>0.85 V</td>
<td>569.59 $\mu$W</td>
<td>135.28 pJ</td>
<td>4226493 $\mu$m$^2$</td>
<td>95.83%</td>
</tr>
</tbody>
</table>

area is 2051761 $\mu$m$^2$. If we observe well, the power consumption of the serial design is the largest of all. We used this value to calculate the power savings obtained when comparing the power consumption of the other architectures against the SLMS8. As can be expected, the SLMS8 has the smallest layout area.

The PLMS8 A and B have an average power consumption of 3514 $\mu$W and 1055.7 $\mu$W, while running at supply voltages of 1.95 V and 1.20 V, respectively. Designs C and D with a supply voltage of 0.85 V consume very little energy per operation regardless of their increased area: only 127.16 pJ for design C and 135.28 pJ for design D with power savings around 96%. Their area is approximately twice the area of the SLMS8. Nevertheless, their supply voltage is dangerously low which does not make these architectures trustworthy for reliable operation. Therefore, architectures PLMS8 A and B with power savings of 74.27% and 92.27%, respectively and with an increase in area that can be neglected, would make a suitable replacement for the SLMS8 if low power operation is required.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

The acquisition, conditioning, and processing of electroencephalographic (EEG) signals has resulted beneficial in the monitoring and diagnosis of certain clinical situations, such as epilepsy. One of the problems when dealing with EEG signals is that most often they are corrupted with artifacts and electromagnetic interferences. The development of an ambulatory wireless integrated system would be a valuable tool for characterizing seizures-like events outside the EEG laboratory. The inconvenience of this endeavor is that it must operate with the added constraint of a small power budget. The aim of this thesis was to evaluate power consumption rates for different LMS adaptive digital filter architectures. This filter was used to attenuate the power line interference in the recording of EEG signals.

The design flow utilized in this thesis is based on Simulink. This tool allowed the simplification of the whole process by providing a unified framework in which algorithm modeling, implementation details, and hardware co-simulation could exist at the same time. The advantages of this approach were basically rapid hardware evaluation of complex signal processing algorithms in an FPGA, and the automation of code and test bench generation when used along System Generator. To understand the design flow, a pipelined filter was implemented using all the design steps, from design entry to physical layout.

Given the large number of transistor contained in the adaptive filters, a characterization methodology was presented in order to facilitate the power consumption, delay, and area estimation of the filter architectures. Since the LMS filter displays a high
regularity in that the computational units are only a few, a full characterization could be applied to the basic functional units present in the system.

This thesis work analyzed the performance and power consumption of five digital signal processing architectures implementing the LMS algorithm. These architectures were pipelined with relaxed look-ahead techniques in order to be able to scale down the supply voltage. Their performance in terms of convergence speed and misadjustment was capable of attenuating the power line interference in artificially generated EEG signals. It was shown that if the filters were to be run at their maximum allowable clock frequency, the power consumption and energy per operation needed for the pipelined architectures were substantially lower when compared to a serial or conventional LMS adaptive filter. The power consumption savings can be estimated at around 75% and up, while the area overhead can be neglected for all practical purposes.

### 6.2 Future Work

This work has provided a framework and a design flow as side contributions which can be helpful to future projects. Other areas of opportunity for future works are presented below:

- The filters can be implemented with a variable step-size parameter as proposed in [4]. The actual step-size implementation can be pipelined with relaxed look-ahead techniques.
- Since the design is powered by a reduced supply voltage, a level-converting circuit can be implemented to allow the internal low-voltage signals to interface to standard CMOS voltages.
- The power consumption and delay estimations could be further refined by extracting the parasitic capacitances from the layout views, back-annotating, and re-simulating the designs.
- In case the adaptive filter finds its way to the IC foundry, a more robust physical verification tool like Calibre from Mentor Graphics could be used to check for DRC and LVS errors.
- If the filters needed to be run at a lower clock frequency and power consumption, subthreshold operation might be considered [6].
Appendix A

VHDL Description of the Pipelined Adaptive Filter

-- 16-bit Adder

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;

entity adder16 is
  port (  
a : in std_logic_vector((16 - 1) downto 0);  
b : in std_logic_vector((16 - 1) downto 0);  
s : out std_logic_vector((16 - 1) downto 0));
end adder16;

architecture behavior of adder16 is
  signal a_17_32: signed((16 - 1) downto 0);
  signal b_17_35: signed((16 - 1) downto 0);
  signal cast_69_18: signed((17 - 1) downto 0);
  signal cast_69_22: signed((17 - 1) downto 0);
  signal internal_s_69_5_addsub: signed((17 - 1) downto 0);
signal internal_s_83_3_convert: signed((16 - 1) downto 0);
begin
a_17_32 <= std_logic_vector_to_signed(a);
b_17_35 <= std_logic_vector_to_signed(b);
cast_69_18 <= s2s_cast(a_17_32, 14, 17, 14);
cast_69_22 <= s2s_cast(b_17_35, 14, 17, 14);
internal_s_69_5_addsub <= cast_69_18 + cast_69_22;
internal_s_83_3_convert <= std_logic_vector_to_signed
  (convert_type(signed_to_std_logic_vector
   (internal_s_69_5_addsub), 17, 14, xlSigned, 16, 14, xlSigned,
    xlRound, xlSaturate));
s <= signed_to_std_logic_vector(internal_s_83_3_convert);
end behavior;

-- 16-bit Mu Constant

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;

entity const_mu is
  port (  
    op : out std_logic_vector((16 - 1) downto 0));
end const_miu;

architecture behavior of const_miu is
begin
  op <= "0001100110011010";
end behavior;
-- 16-bit Delay
-- This is the template for higher order delays used in the design

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;

class delay1 is
    port (       
        d : in std_logic_vector((16 - 1) downto 0); 
        q : out std_logic_vector((16 - 1) downto 0); 
        clk : in std_logic); 
    end delay1;

architecture behavior of delay1 is
    signal d_1_22: signed((16 - 1) downto 0);
    type array_type_op_mem_20_24 is array (0 to (1 - 1)) of signed
        ((16 - 1) downto 0);
    signal op_mem_20_24: array_type_op_mem_20_24 := ( 
        0 => "0000000000000000");
    signal op_mem_20_24_front_din: signed((16 - 1) downto 0);
    signal op_mem_20_24_back: signed((16 - 1) downto 0);
    begin 
        d_1_22 <= std_logic_vector_to_signed(d);
        op_mem_20_24_back <= op_mem_20_24(0);
        proc_op_mem_20_24: process (clk)
            is 
                variable i: integer;
            begin 
                if (clk'event and (clk = '1')) then 
                    op_mem_20_24(0) <= op_mem_20_24_front_din;
                end if;
        end process;
    end begin;
end process proc_op_mem_20_24;

op_mem_20_24_front_din <= d_1_22;
q <= signed_to_std_logic_vector(op_mem_20_24_back);
end behavior;

-- 16-bit Multiplier

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;

entity mult16 is
  port (  
a : in std_logic_vector((16 - 1) downto 0);
b : in std_logic_vector((16 - 1) downto 0);
p : out std_logic_vector((16 - 1) downto 0));
end mult16;

architecture behavior of mult16 is
  signal a_1_22: signed((16 - 1) downto 0);
signal b_1_25: signed((16 - 1) downto 0);
signal mult_46_56: signed((32 - 1) downto 0);
signal product_55_3_convert: signed((16 - 1) downto 0);
begin
  a_1_22 <= std_logic_vector_to_signed(a);
b_1_25 <= std_logic_vector_to_signed(b);
mult_46_56 <= (a_1_22 * b_1_25);
product_55_3_convert <= std_logic_vector_to_signed
  (convert_type(signed_to_std_logic_vector(mult_46_56), 32, 28,
  xlSigned, 16, 14, xlSigned, xlRound, xlSaturate));
p <= signed_to_std_logic_vector(product_55_3_convert);
end behavior;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;

entity sub16 is
  port (  
    a : in std_logic_vector((16 - 1) downto 0);
    b : in std_logic_vector((16 - 1) downto 0);
    s : out std_logic_vector((16 - 1) downto 0));
end sub16;

architecture behavior of sub16 is
  signal a_17_32: signed((16 - 1) downto 0);
  signal b_17_35: signed((16 - 1) downto 0);
  signal cast_71_18: signed((17 - 1) downto 0);
  signal cast_71_22: signed((17 - 1) downto 0);
  signal internal_s_71_5_addsub: signed((17 - 1) downto 0);
  signal internal_s_83_3_convert: signed((16 - 1) downto 0);
begin
  a_17_32 <= std_logic_vector_to_signed(a);
  b_17_35 <= std_logic_vector_to_signed(b);
  cast_71_18 <= s2s_cast(a_17_32, 14, 17, 14);
  cast_71_22 <= s2s_cast(b_17_35, 14, 17, 14);
  internal_s_71_5_addsub <= cast_71_18 - cast_71_22;
  internal_s_83_3_convert <= std_logic_vector_to_signed
    (convert_type(signed_to_std_logic_vector
     (internal_s_71_5_addsub), 17, 14, xlSigned, 16, 14,
      xlSigned, xlRound, xlSaturate));
  s <= signed_to_std_logic_vector(internal_s_83_3_convert);
end behavior;
Appendix B

ModelSim Script

# ModelSim script to automate the compilation, simulation, and
# extraction of port data of a 16-bit adder.
# Inputs: conv_pkg.vhd, adder16.vhd, adder16_tb.vhd
# Outputs: adder16_tb.evcd
# Author: Luis Valencia
# ModelSim SE v6.3c Mentor Graphics Corporation www.mentor.com

-- vcom.do
vlib work
vmap work work

vcom -nowarn 1 conv_pkg.vhd
vcom -nowarn 1 adder16.vhd
vcom -nowarn 1 adder16_tb.vhd

-- vsim.do
vsim -L work -t ps adder16_tb
set NumericStdNoWarnings 1
run 0
set NumericStdNoWarnings 0
run 6250.000000 ns

-- vcd_dumpports.do
restart -f
vcd_dumpports -file adder16_tb.evcd /adder16_tb/dut/a \  
     /adder16_tb/dut/b \  
     /adder16_tb/dut/s \ 
run 6250.000000 ns
quit -sim
Appendix C

VTRAN Script

{

##########################################################################
# This is the vtran command file.                                    #
# Translation: EVCD < to > LSIM                                      #
# Original File: "adder16_tb.evcd"                                   #
# Target File:  "adder16_tb.tv"                                     #
# Command File: "evcd2lsim.vtran"                                   #
# Author: Luis Valencia                                            #
# VTRAN 8.0 (C) 2007 Source III, Inc. www.sourceiii.com            #
##########################################################################
}

{

##########################################################################
# ORIGINAL VECTOR FILE BLOCK                                          #
# The OVF_BLOCK contains commands which tell VTRAN how to read the  #
# vector data in the Original Vector File.                          #
##########################################################################
}

OVF_BLOCK
BEGIN
  SCRIPT_FORMAT verilog_vcd; {# Unquoted}
  ORIG_FILE "adder16_tb.evcd";
}
MAX_UNMATCHED 200 [verbose];
INPUTS a[15:0], b[15:0];
OUTPUTS s[15:0];
END

{

######################################################################
# PROCESSING BLOCK
# This is an optional command block in the VTRAN command file. It
# can be used to specify special processing and transformations that
# VTRAN should perform on data in the OVF during translation.
######################################################################

PROC_BLOCK
BEGIN
STATE_TRANS pure_inputs
'D'->'0', 'U'->'1', 'n'->'X', 'N'->'X', 'd'->'0', 'u'->'1',
'L'->'0', 'H'->'1', 'l'->'0', 'h'->'1', 'T'->'Z', 'x'->'X',
'? '=' 'X', 'A'->'0', 'a'->'0', 'B'->'1', 'b'->'1', 'C'->'X',
'c'->'X', 'f'->'Z', 'F'->'Z';

STATE_TRANS pure_outputs
'L'->'0', 'H'->'1', 'l'->'0', 'h'->'1', 'T'->'Z', 'x'->'X',
'D'->'0', 'U'->'1', 'n'->'X', 'N'->'X', 'd'->'X', 'u'->'X',
'? '=' 'X', 'A'->'1', 'a'->'X', 'B'->'0', 'b'->'X', 'C'->'0',
'c'->'1', 'f'->'Z', 'F'->'X';

STATE_TRANS bidir_inputs
'D'->'0', 'U'->'1', 'n'->'X', 'N'->'X', 'd'->'0', 'u'->'1',
'? '=' 'X', 'A'->'0', 'a'->'X', 'B'->'1', 'b'->'1', 'C'->'X',
'c'->'X';

STATE_TRANS bidir_outputs
'L'->'0', 'H'->'1', 'l'->'0', 'h'->'1', 'T'->'Z', 'x'->'X',
'? '=' 'X', 'A'->'1', 'a'->'X', 'B'->'0', 'b'->'X', 'C'->'0',
'c'->'1', 'f'->'Z', 'F'->'X';
TIME_OFFSET 1000
END

{
# The commands in the TVF_BLOCK tell VTRAN how to format the vectors #
# in the Target Vector File. The most obvious commands needed here #
# are the SIMULATOR command which specifies for which target #
# simulator the vectors should be formatted, and the TARGET_FILE #
# command which specifies the name of the target vector file.  #

TVF_BLOCK
BEGIN
    SIMULATOR lsim;
    TARGET_FILE "adder16_tb.tv";
    SCALE 1000
END
END
Appendix D

LeonardoSpectrum Script

# LeonardoSpectrum script to automate the process of logic synthesis#
# Inputs: Registered ASIC library, conv_pkg.vhd, plms_8_11_1_1.vhd  #
# Outputs: Area report  #
# Delay report  #
# Verilog netlist  #
# Top-level verilog netlist  #
# Verilog netlists for every module  #
# VHDL, SDF, EDIF, NCF, and LPRF files  #

load_library c35_CORELIB
set_working_dir C:/Thesis/work/results/plms_8_11_1_1/leonardo
set resource_sharing FALSE

read -technology "c35_CORELIB"  { C:/Thesis/work/results/ 
/plms_8_11_1_1/leonardo/source/conv_pkg.vhd C:/Thesis/work/results/ 
/plms_8_11_1_1/leonardo/source/plms_8_11_1_1.vhd }

pre_optimize -common_logic -unused_logic -boundary \ 
-xor_comparator_optimize

pre_optimize -extract
set register2register 100.000000
set input2register 100.000000
set register2output 100.000000
set optimize_for area
set report brief
set -hierarchy preserve
set effort quick
set hierarchy_auto FALSE
set hierarchy_preserve TRUE
set hierarchy_flatten FALSE
set modgen_select none
set macro TRUE
set chip FALSE

optimize .work.plms_8_11_1_1.structural -target c35_CORELIB -macro\  -area -effort quick -hierarchy preserve

report_area C:/Thesis/work/results/plms_8_11_1_1/leonardo/reports\ /area.txt -cell_usage -hierarchy -all_leafs

set propagate_clock_delay TRUE

report_delay C:/Thesis/work/results/plms_8_11_1_1/leonardo/reports\ /delay.txt -num_paths 1 -show_schematic 1 -longest_path\  -show_input_pins -clock_frequency

set novendor_constraint_file FALSE
auto_write -format Verilog plms_8_11_1_1.v
set novendor_constraint_file FALSE
auto_write -format Verilog -single_level plms_8_11_1_1_top.v
set novendor_constraint_file FALSE
auto_write -format Verilog -module plms_8_11_1_1.v
set novendor_constraint_file FALSE
auto_write -format VHDL plms_8_11_1_1.vhd
set novendor_constraint_file FALSE
auto_write -format SDF plms_8_11_1_1.sdf
set novendor_constraint_file FALSE
auto_write -format NCF plms_8_11_1_1.ncf
set novendor_constraint_file FALSE
auto_write -format EDIF plms_8_11_1_1.edf
set novendor_constraint_file FALSE
auto_write -format Preference plms_8_11_1_1.lprf
Bibliography


