This is to certify that the
thesis entitled

LOW ENERGY HARDWARE FOR SENSOR SIGNAL CALIBRATION AND COMPENSATION

presented by

PRASANNA BALASUNDARAM

has been accepted towards fulfillment
of the requirements for the

Master of Science	degree in	Electrical and Computer Engineering

Major Professor's Signature

January 15, 2004

Date

MSU is an Affirmative Action/Equal Opportunity Institution

LOW ENERGY HARDWARE FOR SENSOR SIGNAL CALIBRATION AND COMPENSATION

By

Prasanna Balasundaram

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

ELECTRICAL AND COMPUTER ENGINEERING

2004

ABSTRACT

LOW ENERGY HARDWARE FOR SENSOR SIGNAL CALIBRATION AND COMPENSATION

By

Prasanna Balasundaram

When semiconductor sensors transfer signal from one domain to the other, an accurate output is not reported due to inherent physical properties of sensor materials and the problems in sensor manufacturing. Non-linearity, offset and cross sensitivity are typical phenomena observed in the sensor outputs, requiring calibration and compensation processes to obtain meanigful information from the sensors. This thesis makes use of advances in semiconductor industry to develop a correction engine in the form of an Application Specific Integrated Circuit (ASIC) that efficiently calibrates and compensates sensor data. The correction engine uses floating point hardware to perform the error correction prescribed by the IEEE 1451.2 standard. The configurable correction engine is capable of performing the error correction operations to suit the energy demands of the battery powered sensor microsystem, either with high accuracy, or with ultra-low energy expenditure. Energy efficient cell library, compact multipliers and adders reduce the power consumption in the computations. Novel value prediction scheme and an efficient rounding mode are employed to increase the effectiveness in spending the energy. The hardware correction engine facilitates the development of key microsystems for medical, commercial and industrial applications using simple, low-cost sensors that otherwise would not provide reliable data.

To Michigan State University

ACKNOWLEDGEMENTS

I would like to start by giving my thanks to Dr. Andrew Mason for providing continuous moral support in doing the thesis work. I am greatful to the freedom that he gave me throughout the thesis work. Without his encouraging words and guidance, I would not have finished this work. I like to thank Dr. Michael Shanblatt, Dr. Nihar Mahapatra, and Dr. Peixin Zhong for serving the thesis committee in their busy schedule. I like to thank the department chair person Dr. Satish Udpa, and the graduate co-ordinator Dr. Donnie Reinhard for their continuing support during the Masters degree program. I like to thank Mr. Fredrick Hall and other unix administrators for answering my requests immediately, even on holidays. I like to thank Mr. Peter Semig for clarifying my doubts and providing tool kit support. His involvement in getting the technical documents from cadence Sourcelink was very useful in the library development process. Jichun Zhang, Junwei Zhou, and Kartik Vaidyanathan of Advanced Micro Systems and Circuits Lab provided good encouragement during the last couple of years. I like to thank Matthew Guthaus, Eric Marsman, and Robert Senger of University of Michigan, who gave technical support when I was developing the library. I like to thank the *comp.cad.cadence* usenet group, especially Andrew Beckett of Cadence Design Systems for providing valueable suggestions when I faced problems with CAD tools. I like to thank my friends Chandan Reddy, Shankarshna Madhavan, Arvind Ravisekar, Loganathan Anjaneyulu, Badrinarayanan Kasturi, Mahesh Arumugam, Narasimhan Swaminathan, Sunder Balakrishnan, Srinivasan Rakhunathan and many more for encouraging me to do challenging things in life. I learn a lot from them. Finally, I like to thank my Mom, without her blessings and kind heart, I would not have written this.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Semiconductor sensors play an important role in measuring a physical quantity for control applications. They take part in our every day activities and make life easier. The complexity associated with a sensor may vary from a simple room-temperature control to advanced motion control in an airplane. Advances in the Micro Electro Mechanical Systems (MEMS) industry lead us to the age where sensors are downsized to the order of micrometers. Modern sensors acquire information such as temperature, pressure, or humidity and transform them into another domain, where the information is processed, and suitable control action is taken to keep the physical quantity under control. The transformed domain is usually the electrical domain, since information processing and communication are easier and flexible using electronic circuits. When the sensors transfer the signal from one domain to the other, accurate output is not reported due to inherent nature of the problems in sensor manufacturing. This error must be corrected by proper means to interpret the quantity of the interest correctly. This thesis focuses on developing an Application Specific Integrated Circuit (ASIC) capable of performing the error-correction efficiently, making use of advances in the semiconductor industry.

1.1 Smart sensors and error correction

The era of semiconductor sensors began when the piezoresistivity of silicon was discovered. Consequently, devices such as photodiodes and Hall devices were developed. The main purpose of these devices is to generate an electric signal proportional to the physical quantity of interest. A practical sensor usually doesn't yield an ideal signal transfer curve,

but includes effects such as nonlinearity, offset, and non-unity gain. Commonly, the sensor signal is not only proportional to the physical quantity of interest, but also sensitive to other parameters such as temperature. Removing the effects of non-linearity, offset and gain is known as calibration while removing the effect of other physical quantities like temperature is known as compensation. These undesired effects should to be removed from the sensor signal so that the signal produced by the sensor can be interpreted correctly.

Traditionally these errors were corrected by laser trimming discrete components such as resistors and capacitors in the signal conditioning circuits. This type of error-correction required individual attention to each sensor; high material and labor costs increased the price difference between a calibrated and un-calibrated sensor. To reduce the cost of the error correction, more signal conditioning circuits were integrated with the sensors as the VLSI technology advanced. Mixed signal designs enabled the amplifiers and other passive components to become digitally programmable so that the parameters associated with the components can be modified as desired. The sensors became capable of communicating the sensor signal to a digital computer and sensor systems became smart once it started processing the information. Improvements in the semiconductor fabrication and packaging reduce the cost of the sensors; at the same time the intelligence of the sensors keeps ever increasing.

The integrated sensor module can be simplified as in Figure 1.2 where the major blocks include the analog interface, signal conditioning, amplifier, A/D converter, and a bus interface to communicate the data to a digital controller. The error-correction procedure can be performed in any stage after the sensor output is read by the signal conditioning circuit. The transfer curve of the amplifier can be controlled by modifying the passive components in the circuit to overcome the effects of nonlinerities. This involves using trimmable or

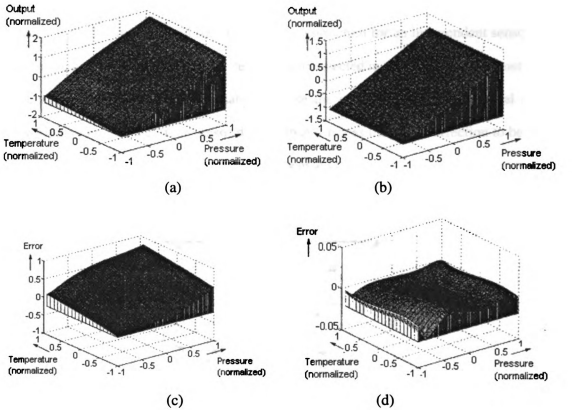


Figure 1.1: Output of the sensor before (a) and after calibration (b), error surface before (c) and after calibration (d) [6].

programmable resistors and capacitors. Their ability to control the calibration and compensation process becomes limited and each sensor needs individual attention, which increases the production cost of the sensor. The calibration and compensation can be done using programmable amplifiers [13] and Analog to Digital Converters(ADC) [18].

These circuits become very complex and less flexible. Single Chip ASIC with signal conditioning and error correction [11] and performing the error correction in the sensor module itself [12] offer a good solution for error correction for an independent sensor, but these implementations suffer if many sensors are connected in the form of a small network. Redundant sensors become unavoidable because the compensation of one signal in the network may require the other sensor data. Hence it forces the error correction to be done in a centralized place where all the data are processed.

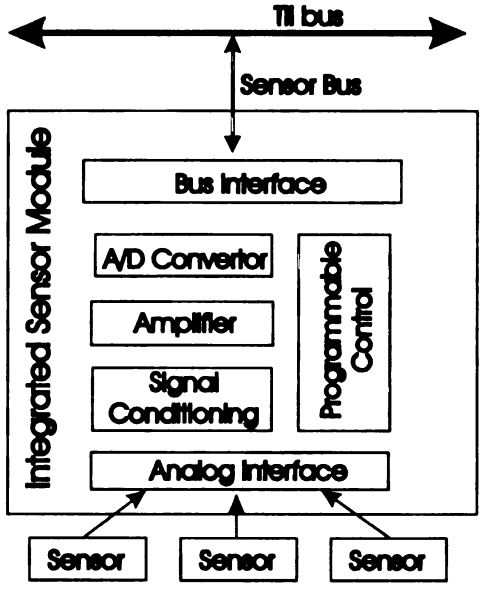


Figure 1.2: Components of an Integrated Sensor Module.

The error correction in a central digital controller in the digital domain offers flexible and accurate solution. The simplest method of error correction is to build a lookup table in the memory of the controller. The lookup table method is very fast, it requires huge memory in the controller and also offers only limited resolution. The other method of doing the error correction is to fit the sensor transfer curve as a multinomial and evaluate the multinomial when the sensor signal arrives. Sophisticated techniques such as spline functions [7] exist to determine the multinomial coefficients using minimal measurements. Usually these coefficients are expressed as floating point numbers to increase the range of representable values. Once the multinomial coefficients are determined, the co-efficients are stored in the non-volatile memory of the sensor. When the sensor is connected to the network, the non-volatile memory is read for performing error correction. Recalibration of the sensor can be done by simply rewriting the modified coefficients.

Widespread usage of single chip microcontrollers offer an attractive means for performing sensor signal calibration and compensation as they provide cost effective means of computation. Since many microcontroller cores don't support floating-point operations needed to perform the error correction, specific subroutines [5] are written to do the floating-point operations. The disadvantage in using subroutines method is that the error correction process for a single data point takes large amount of time, limiting the sampling rate of the sensor to a very low value. For example , running on a 2-4 MHz clock, a typical errorcorrection routine takes about 4-13 ms [4]. This engages the controller for a longtime that it wont be able to perform other control applications.

The accuracy of the error correction depends on how well the sensor signal transfer curve is represented using the multinomial approach. Experiments show that higher the order of the multinomials, lower the error bounds. But high accuracy comes at the cost of increasing the processing time needed by the microcontroller. An attractive alternative to

reduce the order of the multinomial and also to decrease the processing time is to segment the various regions of operation of the sensor and approximate the transfer curve by a lower order multinomial. In most cases, this doesn't considerably increase the error. Using subroutines to perform the error correction in the micro-controller consumes more energy due to the overhead when the controller handles interrupts. Hence, by developing a dedicated hardware that is capable of performing floating-point operations, the error correction can be performed with lower energy consumption leading longer battery life.

Since there are many ways to build sensor modules and to communicate with central digital controllers, many consumer products have appeared in the market. To regulate the products and to improve the portability of devices, IEEE 1451.2 [2] standardizes the communication between the smart transducer and the microcontroller and also the Trasnducer Electronic Data Sheet(TEDS). It defines how the data inside the sensor module are organized for internal programmable control. It also recommends digital error correction of the sensor signal using the piecewise linear multinomial approach. The standard also encourages that the error correction be performed using floating-point operations.

The sensor signal transfer curve is approximated by a multinomial in an n+1 -dimensional space, where n is the number of parameters the sensor signal data depends upon. For example, if a pressure sensor signal depends on the pressure channel data and the temperature channel data, then n will be equal to 2 and the space formed is a 3-dimensional space. The pressure and temperature data will form the X and Y-axes while the corrected pressure data will form the Z-axis. The TEDS allow the independent axes to be segmented into as many segments as needed to reduce the degrees of the multinomial. For example, if the pressure and temperature channel spectrum are divided into 2 and 3 segments, then the entire space will be divided in to 6 region of operation. In each region of operation, the transfer curve can be a multinomial of an arbitrary degree required. If we assume that the

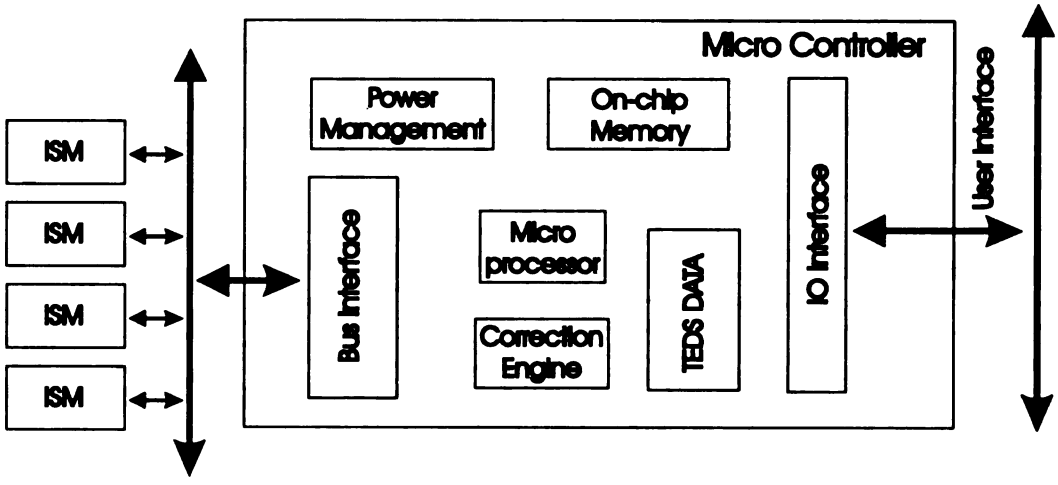


Figure 1.3: Architecture of the central digital controller.

actual sensor data depends on the square of the pressure channel data and linearly on the temperature channel data in a particular region of operation, then the signal transfer curve will be an expression involving 6 coefficients with all possible combination of pressure and temperature channel data raised to the powers of 0 through 2 and 0 through 1 respectively. The general expression for correction for a particular region of operation is given as

$$Y = \Sigma_{i=0}^{D(1)}\Sigma_{j=0}^{D(2)}\cdots\Sigma_{p=0}^{D(n)}C_{i,j\ldots p}[X_1 - H_1]^i[X_2 - H_2]^j\cdots[X_n - H_n]^p$$

1.2 Motivation

The error correction processes in smart sensors accounts for upto 50 % of the cost of the sensors and consumes more than 30% of the energy spent by the micro-controller. Sometimes the errors due to the nonlinerarities and crosssensitivities can change the actual output from 50% to 75% [6] of the true value. Hence automating the calibration and compensation process of the sensor and performing sensor signal error correction by utilizing least amount of resources will lower the cost of sensors and allow them to be used in many more

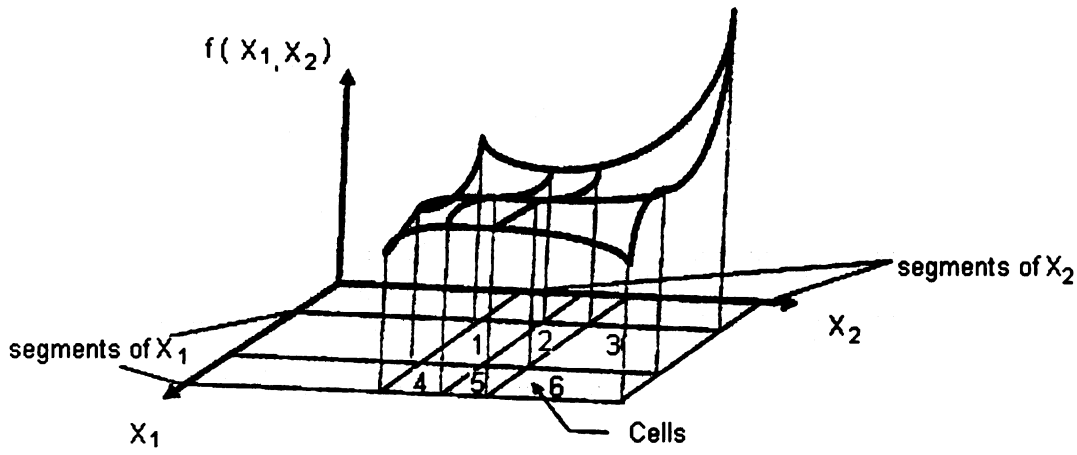


Figure 1.4: Segmenting the operational region of sensors [2].

applications in the future. The piecewise multinomial approach allows the sensor signal to be corrected irrespective of the nonlinearities and cross dependencies. To maintain the accuracy of the error correction process, it is desirable to do the correction process in floating point arithmetic. The correction process in an integer computation based microcontroller involve more energy expenditure due to the overheads in the subroutines and data communication process. Energy savings can be obtained using dedicated hardware to perform the floating-point operations for sensors that cross dependencies with less than two other physical quantities. It will be an optimal balance if we design the hardware to perform the correction upto the 3^{rd} order, and upto 4 input signals, and allow software programs to perform the correction if the needed order is higher. We can also obtain considerable energy savings if previously computed values are stored in on chip memory to avoid repeated calculations. If the contribution of a particular co-efficient is very small, energy savings can be obtained at the cost of slight loss of accuracy. Also for small perturbations in the input

signal, corrections can be obtained with less number of computations by applying Taylor's expansion of the multinomial at the operating point.

1.3 Goals

The main goal is to design a reconfigurable correction engine to perform calibration and compensation of sensor signals using floating-point hardware optimized for low-energy. At the lowest level, energy savings are considered while designing the library cells like logic gates and flip-flops [8, 15]. At an intermidiate level, modules such as adders, multipliers [16] and rounding units are optimized for low energy dissipation. At the architectural level, the required number of computations are optimized using a data analyzer. At the system level the previously computed values are analyzed for opportunities to reduce the number of operations. By designing the hardware from device to system level optimizing for energy savings yields low energy error correction hardware.

1.4 Organization

The rest of the thesis is organized as follows: Chapter 2 discusses design flow and the cell library generation process, Chapter 3 describes design issues in integer adder and multiplier units and floating-point rounding units , Chapter 4 highlights the top level deisgn of the correction engine and the hardware sorter, and Chapter 5 summarizes the results acheived during the project giving directions for future research.

CHAPTER 2

Standard Cell Library Design

As the feature size of integrated circuits (ICs) grow smaller, it enables the designers to pack more gates in the given area of the chip and acheive more functionality. At the same time, more burden is placed on the designer to ensure the reliability of the chip. Understanding the design flow and the tools used in the design process leads the circuit designer to make wise choices in the design to give maximum productivity. ASIC technology is proven to be cost effective for mid-high volume applications. This chapter focuses on the design flow and the library used in the correction engine design.

Bottom-up and top-down are the two widely used design flows for integrated circuit design. In the bottom-up design flow, the system is divided into sub-blocks and each module is designed considering timing constraints by varying the transistor size. Though this method gives the maximum efficiency interms of area and power constraints, this approach is not very efficient and some time impractical for designs involving thousands of transistors. In the top-down design flow, the functionality of a block is given a priority and it is expressed in hardware description language such as verilog. Once the design meets the functionality requirements, the synthesis tool develops a generic implementation and maps it to the target technology meeting the timing and area constraints. The physical design is performed by the Auto Place and Route (APR) tool after the synthesis process.

To design an integrated circuit with a specific process technology using the top-down method, a cell library is needed. The cell library contains information about the timing properties (to perform logic synthesis) and physical properties (to perform physical design) of the cells in the library. An in-house standard cell library was developed for designing

the correction engine in AMIC5N process for this project. The library contains a complete set of combinational cells, tristates, latches, and flip-flops, so that any digital design could be implemented using the top down method. This cell library helps the designer to produce integreated circuits with high reliability in a short period of time.

2.1 Library Design Flow

In this correction engine design, Silicon Ensemble and Build Gates were used for performing logic synthesis and physical design respectively, since they are a part of the Cadence NCSU Design kit used in Michigan State University. Silicon Ensemble requires the physical information be represented in a Library Exchange Format (LEF) and Build Gates require the timing information be represented in Timing Library Format (TLF). This section explains the LEF and TLF generation processes.

2.1.1 LEF Generation

The LEF file contains information about the metals, vias, and poly layers for each cell in the library. The LEF file lacks information such as wells, and active layers, that are not relevant for the cell-based place and routing. Cadence Abstract Generator is used for Generating LEF file from the layout. A Design Planner Universal eXchange (DPUX) file containing the technolgy information is created for the Abstract Generator. The Layout of the cells are given as input for the Abstract Generator to generate the LEF file and the generated LEF file is verfied by a sample place and route run.

DPUX Generation

The recomended method for generating the DPUX file is to input an existing LEF file with technology information and ask the Abstract Generator to create this file. But in

the absence of the technology LEF file, the information is entered manually reading the technology file dumped by the Design Framework (dfII) environment.

Layout Generation

The layout for the cells in the library is created in Virtuoso Layout Editor passing the Design Rule Check (DRC) and Layout Versus Schematic (LVS) tests. Before designing the layout, the horizontal and vertical pitches (the distance between the center of two metal strips; the grid spacing is set to the corresponding pitch) of the cells are decided. The performance of the routing engine is enhanced if the ratio of the horizontal pitch and the vertical pitch is kept as a simple ratio. For the AMIC5N process library, a horizontal pitch of 8λ ($2.4\mu m$) and a vertical pitch of 10λ ($3\mu m$) are chosen. The following design considerations are kept in mind when designing the layout.

- The height and the width of all cells are kept in multiples of the vertical and horizontal pitch respectively.
- The offset (the distance by which the boundary of the cell extends beyond the grid) for the cells are kept at half of the corresponding pitch.
- Metal 1 and 3 layers are drawn horizontally and Metal 2 is drawn vertically.
- The input and output pins are kept in the intersection of horizontal and vertical grids to increase the efficiency of the router.
- Text/Pin lables are bound inside the shape pin to avoid exclusion of pins in the abstract generation step.
- Width of the power rails are kept as one vertical pitch ($3\mu m$).

Abstract Generation

The layout of the cells are given as input to the Abstract Generator to create the LEF file. The following steps are involved in this process.

- Input Technology: The DPUX file generated before is given as the input for the Abstract Generator in this step.
- Input Layout: The layout is exported from the dfII environment in GDSII format (stream) and imported to the Abstract Generator using a layer map table.
- Import Logical: The input and output pin information about the cells is given in a verilog file for the Abstract Generator.
- Pins Step: The text labels are mapped to the terminal lables in the layout and the place and route boundary is created.
- Extract Step: The Abstract Generator probes through the layout and finds the connectivity among the nodes using various layers. Antenna information (capacitance, inductance and resistance information) about the cells are also created in this step.
- Verify step: LEF view for the cell is created and a target place and route run is performed to make sure that the cell can be used by the place and route engine.
- Export LEF: The LEF file which can be directly used as input for Silicon Ensemble is exported from the Abstract Generator.

Veryfying LEF File

A verilog netlist using the cells in the library is created and tested with Silicon Ensemble. The density of the design is studied and the layout is modified to improve the output

of the APR engine. A row utilization of 40% is achieved with the cell library, quite reasonable for a 3 metal process, though a row utilization of 80-90% is common with a 6 metal process.

2.1.2 TLF Generation

The timing library format, TLF file represents the input-output characteristics such as delay and functionality for each cell in the library. The TLF clasifies the cells as combinational cells (the output of the cell depends only on the current inputs of the cell, but not on the previous outputs), tristates (in addition to logic high and low as in combinational cells, the output may be floating for these type of cells), latches (level sensitive device which stores data when write enabled), flip-flops (edge sensitive storage device) etc. Each type of cell contains some unique characteristics that help the synthesis tool to identify a particular class of cells and perform the logic synthesis using them. If an input signal applied to a cell changes its output, the cell has a path (or an arc) from the specific input to the output. The TLF file summarizes all of the possible paths. A more formal definition of the quantities involved in the TLF is summarized below with reference to Figure 2.1.

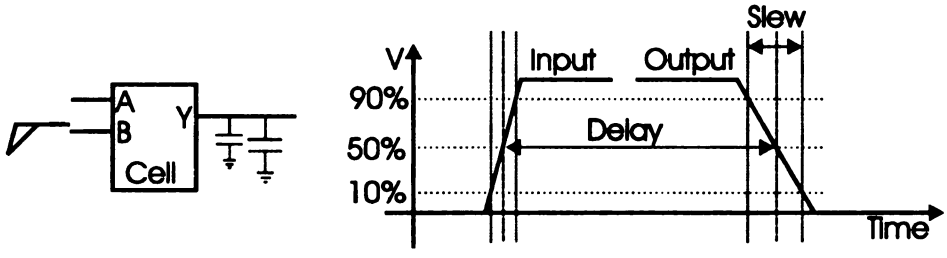


Figure 2.1: Parameters associated with a cell path [3].

- Slew: Time for an input/output signal to rise from V_{Th1} to V_{Th2} (V_{Th2} to V_{Th1} for a fall transition). Usually V_{Th1} and V_{Th2} are set as 10% and 90% of the vdd, and the input/output signal is approximated to a ramp when measuring this quantity.

- Delay: Time difference between the input and output crossing the mid-point in the transistion (V_T).

- Setup: The time for which the input signal has to be stable before the clock transition to ensure proper storage of data in the flip-flop/latch.

- Hold: The time for which the input signal has to be stable after the clock transition to ensure proper storage of data in the flip-flop/latch.

- Recovery: The time after which the asynchronous signal (set/reset) has to be applied to override the data stored by the clock signal.

- Removal: The time before which the asynchronous signal (set/reset) has to be applied to override the data stored by the clock signal.

Combinational Cells

In a combinational cell, the output state depends on the inputs, and change in an input can cause its output to change. The output slew and the delay (between input and output) depends on the input slew and the output load. Both the output slew and delay are represented as two dimensional timing tables (by taking the input slew and output load as independent axes) in the TLF file for all possible paths in the cell. In addition to that, the functionality of each pins and the area of the cell are represented in the TLF file. The static timing analysis (STA) tool uses the TLF file to determine the delay in the a circuit and hence the worst path in the circuit. Linear interpolation techniques are used by the STA tool if the output slew and delay information are nor readily available in the tables.

Parasitic capacitances as small as 0.001pF are extracted from the layout and a netlist is created. The timing characteristics are obtained by simulating the cell with various input slew and output load conditions. Spectre is used for simulating the transistor level circuits and perl scripts are used for reading the output generated by Spectre and report it to a TLF file. The scripts developed for this library can identify any combinational gates upto 3 inputs and report all the possible paths for the specified input slew and load conditions.

Tristates

Tristate devices are used in synthesizing bus structures in a design. The tristate devices differ from the combinational blocks as the output of the tristate devices can be in high impedance state (Z) in addition to logic high (1) and logic low (0). The timing information for the output transition from either 1 or 0 to Z is not critical since a Z output is not driving any other gates. Hence the timing table is filled with entries as zero. The time for an output to go from Z to 1 can be no greater than the time for the output to go from 0 to 1. Hence the worst case time for 0 to 1 and 1 to 0 is substituted for the change from Z to 1 and Z to 0 respectively.

Latches and Flip-flops

Latches and flip-flops are storage elements that contain additional parameters like setup and hold times. If they have asynchronous inputs like preset or clear, the recovery and removal times are included in the TLF file. The definition for setup and hold times mentioned in the Section 2.1.2 were modified slightly to determine those times using simulations. The delay for the data to get stored in the flip-flop after the rising/falling edge of the clock varies with the time the clock signal is applied after the data has settled. If the time difference between the clock edge and the data signal is decreased, the delay for the data to get stored in the flip-flop increases. The setup time is computed as the minimum time before which

the data has to be stable so that the storing delay does not increase beyond 5% of the normal operational delay. Hence if we determine the time difference between the data and the clock, such that the delay between the clock and the output is as close as to 105% of the normal delay time, that becomes the setup time for the transition with specified data and clock slews. From Figure 2.2, we see that the delay of the gate gradually increases as the data is moved closer to the clock and the data fails to latch beyond a limit (shown as the discontinuity).

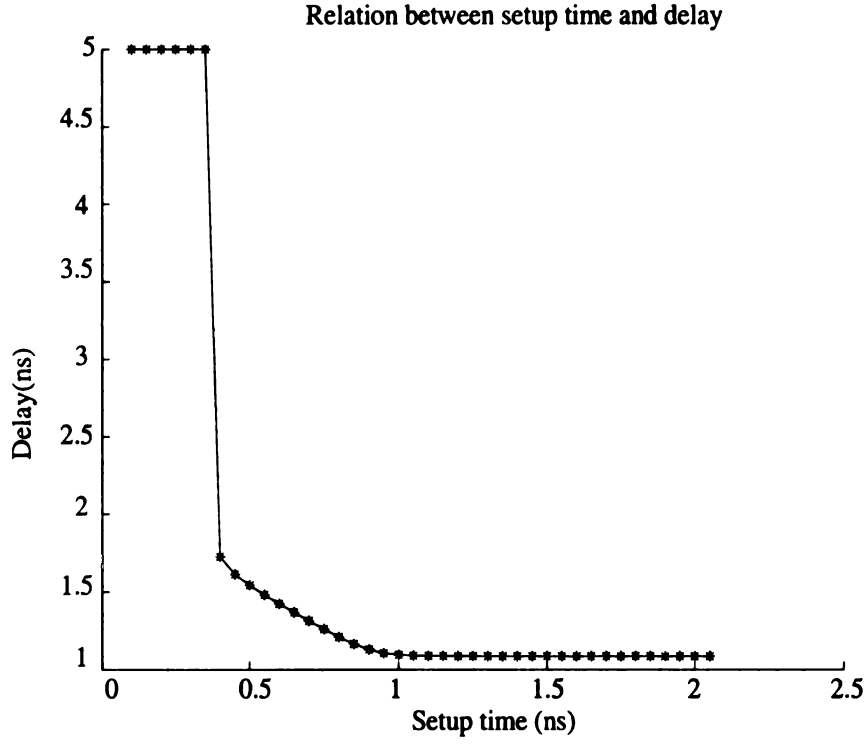


Figure 2.2: Delay of the Flip-flop for various setup times.

Setup time of the flip-flop is determined by performing spice simulations iteratively on the following basis. The normal storing delay is found by giving enough seperation

between the input and the clock signal without violating the setup condition (say 5 ns). Another simulation is performed with no seperation between the data and the clock signal, and hence violating the setup condition. Consequent simulations are performed by either increasing or decreasing the seperation between the data and the clock signal depending on whether a violation has occurred or not. The increment or decrement in the seperation is half of the time between a simulation without violation and a simulation with violation. In Figure 2.3, the setup time converges after 7-8 simulations performed in this manner. A similar argument is extended for determining hold time also.

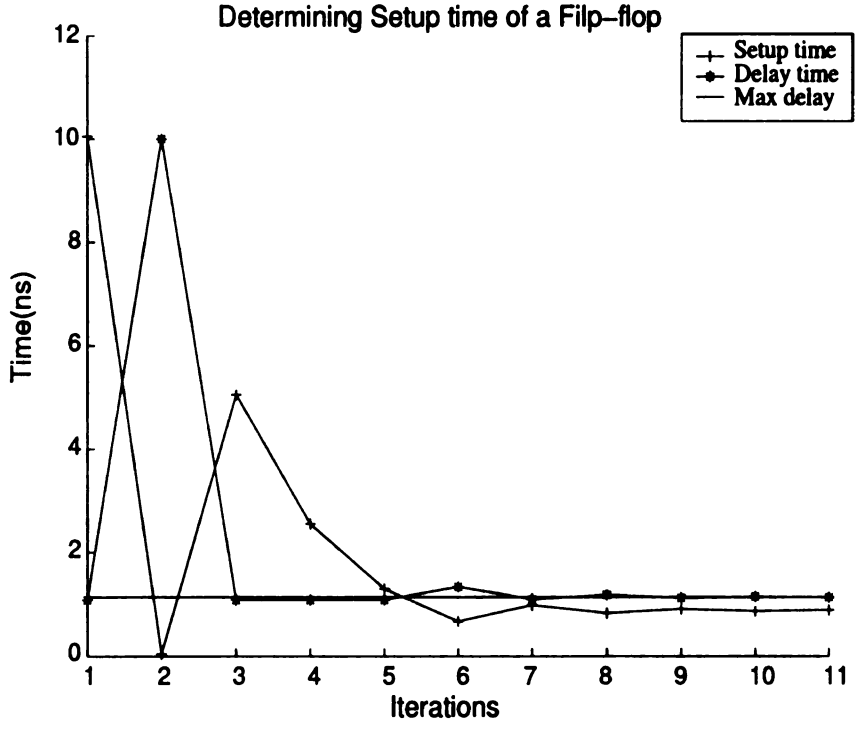


Figure 2.3: Determining Setup time by iterative simulations.

2.2 Results

The developed cell library has an inverter, a tristate inverter, a buffer, 2 and 3 input NAND, NOR, AND, and OR gates, a 2 input XOR gate, a 2:1 multiplexer, a latch, a flip-flop with reset and a flip-flop without reset. Five structures were considered for the flip-flops and two of the most efficient ones are chosen to be included in the library. The following sections summarize the physical properties of the cells in the library.

2.2.1 Combinational Cells

The Table 2.1 summarizes the logic function, width of the cell, number of transistors, input/output capacitance, and power consumption for a run at 40 MHz with all possible input changes.

2.2.2 Flip-flops

Since flip-flops are used extensively in the pipelined microprocessor and data correction unit, the energy demands of five different flip-flop structures (in-house flip-flop with and without reset (dff, dffr), push-pull isolation (ppi) flip-flop [8], transmission gate flip-flop (tgff), and a regular master-slave flip-flop (rdff)) were thoroughly analyzed (shown in Figure 2.4), and the two efficient structures were included in the cell library. When a simulation is performed, the spectre simulator dumps the raw data (the voltage at various nodes, and the current supplied by the sources in the circuit at each instance of the transient analysis) to a file. The file is read by a perl script [14] to store the data in arrays. The current supplied by the source (i_{vdd}) is multiplied by the value of vdd at each instance of time and integrated over the time period (T_s) of the simulation. Thus $\int vdd * i_{vdd}$ and

$$\frac{\int vdd * i_{vdd}}{T_s} \tag{2.1}$$

Table 2.1: Logical and Physical properties of cells developed for the correction engine design.

Cell	Function	Width (μm)	Delay (ns)	Power (μW)	Transistors	i/p cap (pF)	o/p cap (pF)
nor2	!(A| B)	9.6	0.165	1.738	4	0.711	1.975
nand2	!(A&B)	9.6	0.176	1.711	4	0.720	1.808
inv	(!A)	7.2	0.106	1.881	2	0.720	1.808
or2	(A| B)	12.0	0.416	3.042	6	0.711	1.778
and2	(A&B)	12.0	0.409	3.036	6	0.720	2.489
mux21	(S&B)| ((!S)&A)	14.4	0.104	3.481	6	1.690	1.571
dffr	D	62.4	1.027	6.630	34	0.519	1.854
fill		2.4	0.000	0.000	0	0.000	0.000
fill2		4.8	0.000	0.000	0	0.000	0.000
fill4		9.6	0.000	0.000	0	0.000	0.000
nand3	! (A&B&C)	12.0	0.370	1.736	6	0.762	3.405
nor3	! (A| B| C)	12.0	0.319	1.671	6	0.745	2.961
or3	(A| B| C)	14.4	0.597	2.363	8	0.745	1.838
and3	(A&B&C)	14.4	0.628	2.423	8	0.728	1.838
buf1	(A)	9.6	0.327	3.861	4	0.854	1.335
xor2	(A$\oplus$ B)	19.2	0.525	6.096	12	2.739	1.242
dlatch	D	24.0	0.000	34.413	14	0.552	1.571
tinv	!A	12.0	0.208	2.749	6	0.552	1.571
delay	A	74.4	0.000	10.078	8	24.859	1.803
buf2	A	14.4	0.000	4.788	4	0.720	1.803
nand4	!(A&B&C&D)	26.4	0.000	4.228	14	1.306	1.571
and4	(A&B&C&D)	24.0	0.634	3.620	12	1.306	2.065
or4	(A| B| C| D)	24.0	0.000	3.766	12	1.306	1.926
fill5		1.2	0.000	0.000	0	0.000	0.000
passgate	A	12.0	0.000	2.093	4	1.453	1.690
tiehi	1	7.2	0.000	0.000	2	0.000	0.477
tielo	0	7.2	0.000	0.000	2	0.000	0.329

and represents the energy and the average power dissipated throughout the simulation. In order to measure the the energy dissipation for a particular transition (for example, clk01q01 indicates the clock transition for storing 1 in the flip-flop overwriting the existing 0) the integration is started when the clock signal crosses the 10% of its complete swing when rising from 0 V to 3 V and stopped when the incremental energy spent is less than 1% of the total energy spent during the transition. In literature, usually the falling edge of a clock transition is ignored for power and energy reports [15] . From the Figure 2.4, we see that non data-storing edge of the clock also consumes a considerable amount of energy.

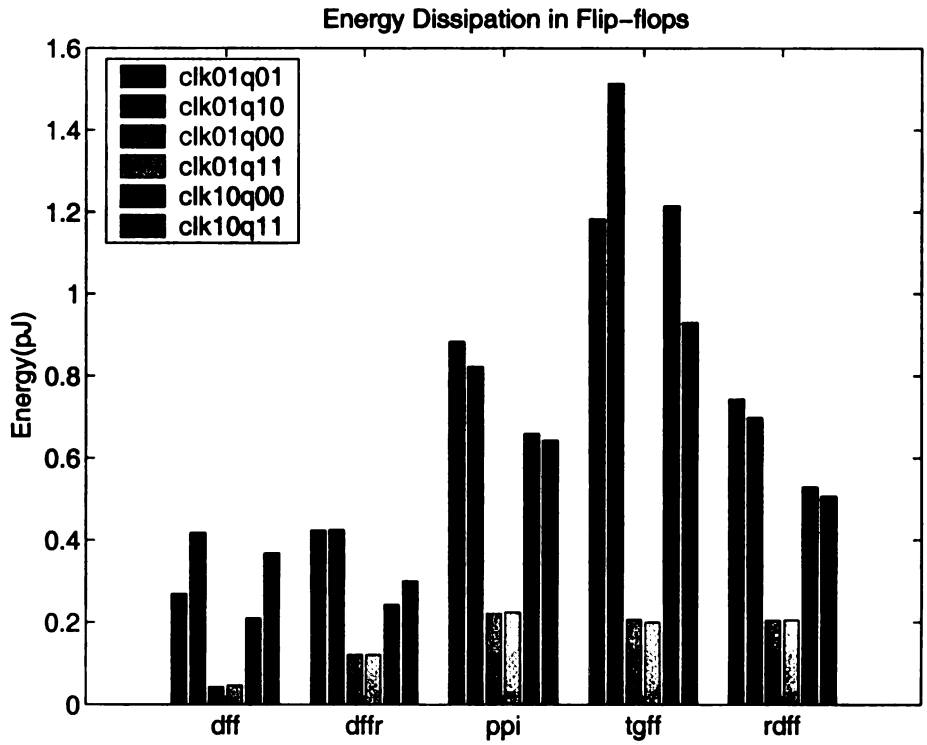
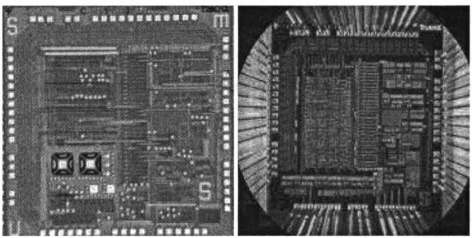


Figure 2.4: Energy expenditure for Flip-Flop operation.Non data-storing edge energy consumption, usually ignored in literature is included for analysis.

2.2.3 System Design

The cell library was used in the top-down design flow of the Universal Micro-Sensor Interface chip which performs data communication between a sensor node and a central microcontroller [19]. The first version of the circuit was designed using full-custom design methodologies while the second version was designed using the cell library. Figure 2.5 shows the layout generated for the bus interface using the full-cuatom design method (a) and the semi-custom design method (b). The library drastically reduced the design time of the bus interface design from months to weeks in the semi-custom design. However, it was achieved only at the loss of the design density. The layout occupied 1 mm x 1mm in the first version, and it occupied about 1.5 mm x 1 mm in the second version.



(a) (b)

Figure 2.5: Layout of UMSI - Full-Custom design (a), Semi-Custom Design(b). Drastic reduction of design time is experienced in (b) when compared to (a).

CHAPTER 3

Floating Point Unit

A digital computer processes either a logical 0 and 1 through its arithmetic and logic units. Integers other than 0 and 1 are represented in a string of 0's and 1's of a specified length. The integer representation can accommmodate 2^n (n-bits wide) discrete value in its spectrum. Floating point representation is used to represent the values between the integer values. Though the floating point quantities also can represent only 2^n discrete values (some times even fewer dicsrete values), the spectrum is very broad when compared to the integer representation. Floating point arithmetic operations can be carried out using integer arithmetic units, but with little modifications to the inputs such as alignment according to their exponents and post-processing the outputs such as normalization and rounding. This chapter discusses integer addition/multiplication units and floating point units used in the correction engine.

3.1 Integer Operations

In order to perform the floating point computations for the calibration and compensation engine, unsigned integer adders and multipliers are used. In this section a variety of adder/multiplier structures are studied and a suitable architecture is identified for the implementation.

3.1.1 Adders

The following adder structures were considered for the project: a ripple-carry adder, a carry-save adder, a carry-lookahead adder, and a manchester carry chain adder. The adders

are compared for the delay, area and energy dissipation. Since the ripple carry adder turned out to be the adder with least energy dissipation, it was chosen for the project.

3.1.2 Multipliers

A variety of multipliers are considered for the project including an array multiplier, booth encoded multiplier, and a Wallace tree multiplier with carry-save adder and ripple-carry adder as final adders. Since the Wallace tree multiplier with ripple-carry adder is the efficient structure in terms of energy, it was chosen for the multiplier.

3.2 Floating Point Operations

IEEE standard for Smart Transducer Interface for Sensors and Actuators [2] recommends that sensor signal processing be performed in floating point precision. A single precision floating point number is represented using 24 bits according to the IEEE floating point standards [1]. In Figure 3.1, the $f[22:0]$ represents the fraction bits (also known as significand), $e[30:23]$ represents the exponent bit and the leading s bit represents the sign of the number. Table 3.1 summarizes the value of the represented number for all the values of the fraction, exponent and the sign bits [9]. If the given floating point number is a normal number, then the significand will be greater than or equal to 1 and less than 2 due to the implicit presence of a leading 1. This is represented as $[1,2)$ in symbolic notations. However, denormalized numbers can have significands greater than 0 and less than 1 and hence represented in the interval $[0,1)$. The value of 127 (known as *bias*) is always added to the exponent in the IEEE representation to represent the negative exponents in unsigned representation. Sensor outputs usually contain noise in the sample that limit data precision to around 12 bits. Thus a precision of 24 bits in significand is not necessary for doing the correction engine operations, hence the significand width is reduced to 16 bits, deviating

Table 3.1: Floating point values for various exponent and significand combinations.

Bit Pattern	Value
$0 < e < 255$	$(-1)^s * 2^{e-127} * 1.f$ (normal numbers)
$e = 0; f \neq 0$	$(-1)^s * 2^{-128} * 0.f$ (denormal numbers)
$e = 0; f = 0$	$(-1)^s * 0$ (signed zero)
$e = 255; f = 0$	$(-1)^s * \infty$ (infinity)
$e = 255; f \neq 0$	*NAN* (Not-a-Number)

from the IEEE floating point standards. Figure 3.1 shows the floating point representation used in the correction engine design.

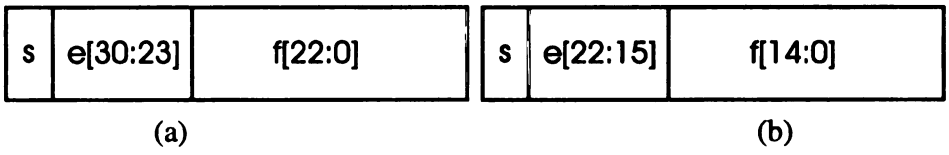


Figure 3.1: IEEE (a) and custom (b) representation of single precision floating point quantities.

The advantages of using floating point computation over integer computation are as follows:

- The floating point number has wider range of numbers that can be represented using the specified number of bits
- IEEE floating point representation handles exceptions precisely and degradation of tiny numbers is handled gracefully

- The exponent can be used as an indication of the magnitude of the represented number; before performing the actual multiplication and addition processes, the magnitude of the results can be predicted. This is used for isolating quantities of least significance to conserve energy while performing the computations.

3.2.1 Multiplication

The floating point inputs represented by $(s1, e1, f1)$ and $(s2, e2, f2)$ are multiplied to form the result (s, e, f). The output is computed by $s = s1 \oplus s2$; $e = e1 + e2$ and $f = f1 \cdot f2$. The significands of the normal and denormal numbers are in the range of $[1,2)$ and $[0,1)$ respectively. Hence the product can be in the range of [0,4) which has to be rounded to the range $[0,2)$. The following sections describe the steps taken to perform the multiplication operation.

Pre-normalization

The leading bit of the significand is not stored anywhere in the number and is said to be implicit. In the pre-normalization stage, the packed input number is unpacked to form the explicit significand for multilpication. A tentative exponent of the result is generated by summing the exponents of the inputs. Since the *bias* gets added twice in the addition process, the *bias* is subtracted once from the tentative exponent. If the the result is too large to be represented using the given single precision point, an exception is set in this stage to handle the overflow of the product.

Significand Multiplication

The 16-bit input significands are multiplied using the unsigned integer multiplier to produce the 32-bit product and the 32-bit result is rounded to 16 bits in the rounding stage.

Rounding

ince the multiplication result is 32 bits, it is more precise than the result that will be represented by the 16 bits. The process of converting the higher precision significand to a lower precision representable significand is defined as rounding. In other words, rounding is the process in which the higher order 16 bits are modified to represent the lower order 16 bits at the cost of accuracy. The result can take either the value of the higher order 16 bits (a) or the higher order 16 bits + 1 ulp (b) where ulp is the least representable quantity for the given bit width. Whether the result takes the value of a or b depends on how close the 32 bit precise result is located between a and b and the rounding mode.

In order to round the result to 16 bits, the 17^{th} bit (rounding bit) is examined. If the rounding bit is a 0, then the result takes the value of a, since the precise significand is closer to a rather than b. If the rounding bit is a 1, and at least one of the bits from 18 to 32 is one (the bits are *OR*ed together to make the *sticky* bit and used for decision making as the position of the 1 in bits 18 to 32 really does not change the decision making), then the result is rounded to the higher magnitude b. However if the rounding bit is 1 and the *sticky* bit is 0, then the 32 bit precise significand lies exactly between a and b, and the result is chosen to be either a or b depending on the rounding mode of operation. The IEEE floating point standard supports 4 rounding modes. The four rounding modes are explained in Figure 3.2 In each of the rounding mode, the result is chosen as follows:

- *Even*: Since the significands a and b are a ulp apart, one of them will be an even and the other will be odd. The even number among a and b is chosen as the result.

- $+\infty$: The significand closer to $+\infty$ is chosen as the result. This depends on the sign of the floating point product. If the sign of the floationg point result is positive, then

b is chosen as the result, since *b* is closer to $+\infty$. If the sign is negative, then *a* is chosen as the result as *a* is closer to $+\infty$.

- $-\infty$: The significand closer to $-\infty$ is chosen as the result in this rounding mode. That is, if the sign of the product is positive, *a* will be chosen as the result and if the sign is negative, *b* is chosen as the result.

- *Zero*: the significand closer to 0 is chosen as the result. Since *a* is closer to zero, *a* is chosen as the result in this mode irrespective of its sign.

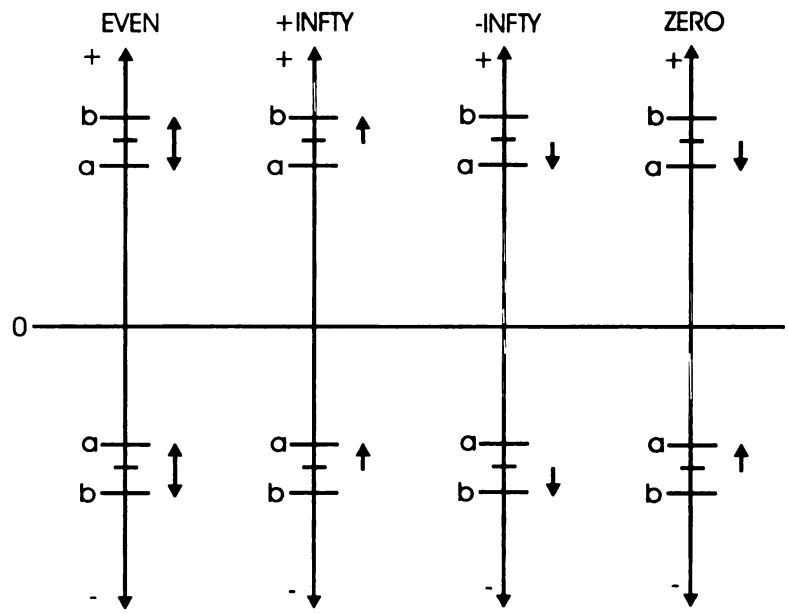


Figure 3.2: Four rounding modes used in the correction engine design.

If the value of the significand *a* is the largest one represented using 16 bits, then incrementing it by an *ulp* will produce a carry. In that case, the exponent is incremented and the result is chosen to be all zeroes. The process of chosing between the significands

a and b is defined as $significand - rounding$ and incrementing the exponent is defined as $post - normalization$. During the post-normalization process, if incrementing the exponent results in a carry, it is impossible to represent the result in the described precision and the hence $overflow$ occurs. The $overflow$ can occur in all rounding modes except for *Zero*, as the significand and the exponent are never incremented, which avoids exponent $overflow$.

As mentioned previously, if the input significands are in $[0,2)$, then the product of those two significands may be in $[0,4)$ and rounding is the process to represent the significand in $[0,2)$. If the product is in $[2,4)$, it is adjusted to be in $[0,2)$ by incrementing the exponent and shifting the imaginary floating point to the left before examining the rounding bit and proceeding to the significand rounding process. If the product of the multiplier is in $[2,4)$, the most significant 16 bits are considered as a and the rounding is carried out with 17^{th} bit as rounding bit and the *OR*ed value from bits 18 through 32 as *sticky* bit. If the product itself is in $[0,2)$, the most significand bit of the product is ignored and the next 16 bits are treated as a. The 18th bit becomes the rounding bit and *or*ed value from bits 19 through 32 becomes the *sticky* bit.

3.2.2 Addition

The multiplication operation is carried out by calculating the 32 bit result and then rounding it to 16 bits. But, the addition process can be performed with a 16-bit adder itself. The addition process is a little more complicated than the multiplication operation as the addition operation may turn out to be a subtraction depending on the sign of the operands. When the inputs are known, the smaller operand is identified by comparing the exponents and shifting the smaller operand to the right, such that both the inputs have equal exponents. Then the addition is performed using a 16 bit adder and the result is post-normalized to

form the final result. The following sections elaborate the floating ponit addition process employed in the design of correction engine.

Pre-normalization

In this stage, the packed floating point operands are processed and sent to the adder unit for addition. The operands are swapped such that the exponent difference between the operands are not negative. The significand of the second operand is complemented if the two operands differ in their signs. The second operand is shifted to the right by the exponent difference positions preserving the sign. The bit adjacent to the *lsb* is assigned as *guard* bit, the bit adjacent to the *guard* bit is assigned as *round* bit and the rest of the bits are *OR*ed together to form the *sticky* bit.

Significand Addition

The significands produced in the pre-normalization are added using a 16-bit adder to produce the sum and carry in this stage.

Rounding

Different rounding procedures are carried out depending on the input operands and the result.

- In the pre-normalization process, the operands are compared only for exponents (to find which operand is large and to shift the smaller operand's significand to the right). If the two operands have the same exponent and their signs are different, then the result produced will be exact and the result will contain leading zeroes. The result is shifted to the left till the *msb* of the result is 1, or the exponent become 0 giving rise to a de-normalized quantity.

- If the operands are of the same sign and no carry is produced during the addition process, the significand is rounded with the *guard* bit as the new *round* bit and the *round* and *sticky* bits are *OR*ed to make the new *sticky* bit. If the significand overflow results, the exponent is incremented. If an exponent overflow occurs, appropriate $overflow/infinity$ flags are set depending on the rounding mode.

- If the operands are of the same sign and a carry is produced in the addition process, the exponent is incremented and the $significand - rounding$ is carried out with the *lsb* of the sum as the new *round* bit and the *guard*, *round* and *sticky* bits are *OR*ed together to make the new *sticky* bit. Abnormal activities such as $overflow$ are detected and proper flags are set during the $significand - rounding$.

- If the operands are of different signs, a carry is produced in the adder and the *msb* of the sum is 0, the carry is discarded and the *gurad* bit is included in the significand and the $significand - rounding$ is carried out with the actual *round*, *guard* and *sticky* bits.

3.3 Implementation

Both the multiplier and the adder are implemented in three pipeline stages. The first stage performs the pre-normalization, the second stage performs the actual multiplication/add operation. The third stage performs the rounding operation. These stages initiate the operations at the rising edge of the clock and write the data in the internal registers which act as the input for the next stage. To avoid the spurious computations, each block is enabled by the previous stage. The floating point multipliers and adders were verfied with test vectors. The verification is carried out with the help of NC-Verilog simulator. Number of gates in each stage and the length of the critical path is listed in Table 3.2 summarizes the number of gates in each of the floating point multiplier/adder stages, the area of the

design , the number of gates in the critical path and the critical path delay. Since the critical path delay for the multiplier is 21.2 ns, it limits the clock frequency to around 40 MHz.

Table 3.2: Design results of the modules in the correction engine design. Shown is the #gates in the module, its area, number of gates in critical path(CP), delay in critical path, and the power consumption when operating at 40 MHz.

Block	#Gates	Area (μm x μm)	CP	CP delay (ns)	Power (mW)
Exp Predictor	999		49	19.02	2.417
Mul-Pre-norm	351	688.8 x 668.4	21	8.02	0.835
Multiplier	4293	1951 x 1951	30	21.2	10.216
Mul-Rounding	867	1015 x 1017	30	16.25	2.039
Add-Pre-norm	1303	1149 x 1127	28	14.96	2.670
Adder	358	702 x 702	16	6.43	0.801
Add-Rounding	1595	1394 x 1370	24	14.71	4.116

CHAPTER 4

Calibration and Compensation Engine

The calibration and compensation engine (correction engine for brevity) performs the error correction of sensor signal prescribed by the IEEE standards [2] using floating point hardware. Essentially the error correction is the process of evaluating the multinomial shown in Equation 4.1.

$$Y = \Sigma_{i=0}^{D(1)}\Sigma_{j=0}^{D(2)}\cdots\Sigma_{p=0}^{D(n)}C_{i,j\ldots p}[X_1' - H_1]^i[X_2' - H_2]^j\cdots[X_n' - H_n]^p \tag{4.1}$$

For example, in a given region of operation, the correction equation might be like Equation 4.2.

$$Y = C_{000} + C_{120}\cdot[X_1' - H_1]\cdot[X_2' - H_2]^2 + C_{213}\cdot[X_1' - H_1]^2\cdot[X_2' - H_2]\cdot[X_3' - H_3]^3 + C_{030}\cdot[X_2' - H_2]^3 \tag{4.2}$$

X_1', X_2', and X_3' are input signals like pressure channel data and temperature channel data, H_1, H_2, and H_3 are the offset values for the given channel, and $C_{000}, C_{120}, C_{213}$, and C_{030} are the gain coefficients. $D(1), D(2)$, and $D(3)$ take values of 2, 2, and 3, the maximum order a particular signal is raised to. The offset values are subtracted from the input signals, raised to appropriate powers, multiplied with other signals, and multiplied with the constant coefficient to form the partial sum term. The partial sum terms are accumulated to form the final output Y.

When the signals from the sensors arrive at the microcontroller, the microcontroller converts the sampled digital signal to a floating point value, compensates for the offset values and initiates the error correction operation (let us assume that the signals compensated with offsets are X_1, X_2, X_3 for simplicity). The correction engine performs the error

correction operation using its hardware resources and report the corrected signal to the microcontroller. The correction engine employs value prediction schemes to perform the error correction operation with high accuracy or with low energy expenditure, trading off one for the other. The architecture and the operation of the correction engine and the hardware sorter is explained in this chapter.

4.1 Correction Engine Architecture

The blocks of the correction engine and their connectivity is shown in Figure 4.1. The microprocessor core directly interacts with the shared memory and stores the necessary values to perform the calibration. The shared memory is 32-bits wide. The calibration coefficients are stored in this 32-bit memory location (shown in Figure 4.2). The lower 24 bits hold the calibration coefficient. The upper 8 bits are divided into 4 banks, each of 2 bits wide to store the orders of X_1, X_2, X_3, and X_4 respectively. This limits the upper bound for the number of independent signals to 4 and their corresponding orders to 3.

The *controller* reads the memory location where the first calibration coefficient is stored and computes the exponent of the partial sum (like $C_{213} \cdot X_1^2 \cdot X_2 \cdot X_3^3$) using the exponents of the input signals. This exponent is tentative, as it might be modified during the rounding in the multiplication process (Section 3.2.1). This tentative exponent is calculated for all the partial sum terms and fed to the sorter to arrange them in ascending order. The partial sum terms are evaluated starting from the lowest tentative exponent for evaluation. In order to evaluate the partial sum, inner porducts (like X_1^2, X_3^3) and cross products (like $((X_1^2 \cdot X_2) \cdot X_3^3)$) are required. These values are computed using the pre-normalization, multiplier and rounding blocks of the floating point multiplier and and stored in the memory. Consequtively, the partial sum is evaluated and passed to the accumulator which is reset at the beginning of the multinomial evaluation. Evaluating the partial sum starting from the

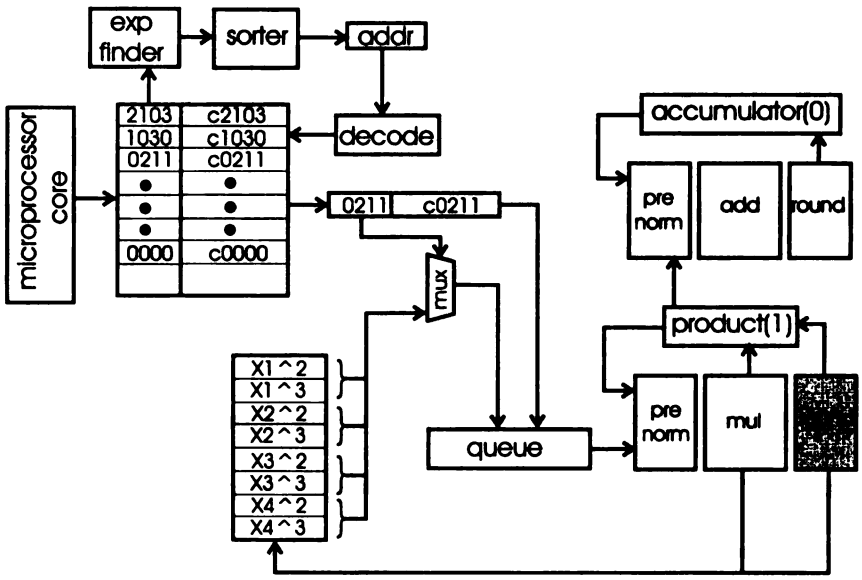


Figure 4.1: Architecture of the Correction Engine with microprocessor core and memory.

term of least significance avoids the tiny number to disappear in the floating point addition process (For example, if we want to add the numbers 10, 0.8, and 0.9 using the floating point adder, which is of 2 digits wide, accumulation starting from 10 will yield a result of 10 as the tiny quantities are lost when the operands are aligned. However if the accumulation is started from 0.8, 0.9, and followed by 10, the result will be 11, a more accurate result).

4.2 Correction Engine Operation

When the inputs from the sensors arrive, the region of operation in the sensor signal tranfer curve is determined by the microcontroller, and the error-correction coefficients are stored in the shared memory. The controller reads the first correction coefficient and the order of the input signals. The controller has a 4-input multiplexer for each input signal.

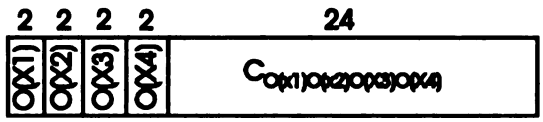


Figure 4.2: Memory word organization. The powers of the input signals are stored in the MSB, while the correction coefficient is stored in the 3 LSBs.

The input signals for the multiplexer are 0, e_{X_1} (exponent of the input signal X_1), $2 * e_{X_1}$ (left shifted once from e_{X_1}), and $3 * e_{X_1}$ (obtained by adding e_{X_1} and $2 * e_{X_1}$); the select signal is the order of the input signal ($O(X_1)$). The outputs of all four of the multiplexers are fed to a 5-input adder, where the other input comes from the exponent term of the coefficient, to form the tentative exponent of the partial sum term (Figure 4.3). The sign of the partial sum term is determined by evaluating $s_{X_1} \oplus s_{X_2} \oplus s_{X_3} \oplus s_{X_3} \oplus s_{C_{O(X_1)O(X_2)O(X_2)O(X_2)}}$. However the sign evaluated here is used only to set an appropriate $\pm\infty$ flag during the overflow at any stage of the shift or addition process.

The exponent of the partial sum terms are computed one by one at each clock cycle. Reading a value of a constant ($O(X_{1...4}) = 0$) indicates that the multinomial does not have any more terms to evaluate. The output of the tentative exponents are fed to the hardware sorter, which accepts all the inputs given in consecutive clock cycles. After getting the signal that the multinomial has reached a constant term, the sorter starts giving its output starting from the term of least significance. Inner porducts (like X_1^2, X_3^3) and cross products (like $((X_1^2 \cdot X_2) \cdot X_3^3)$) are evelauated folloewd by the partial sum term itself (like $C_{213} \cdot X_1^2 \cdot X_2 \cdot X_3^3$). Once the first partial sum is evaluated, it is fed to the adder (pre-normalize, add, and rounding stages) in the consecutive cycles. The other input for the adder comes from the accumulator of the correction engine.

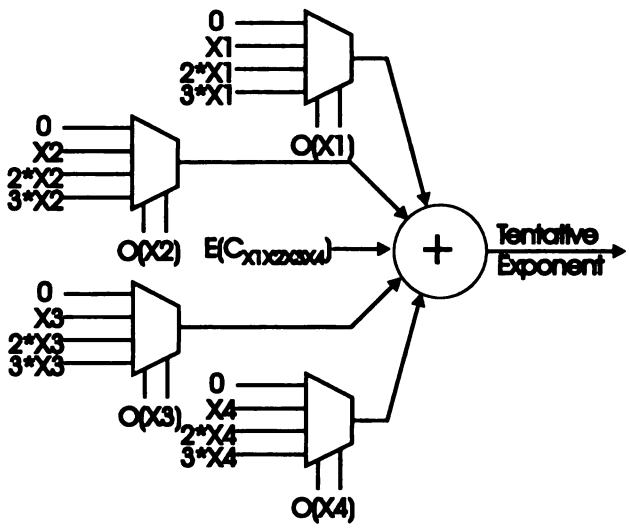


Figure 4.3: Tentative exponent generation with input exponents and correction coefficient.

4.2.1 Clocking

The clocking scheme for various blocks is an interesting one, as not all the blocks produce and consume data on consecutive cycles. Once the sorter produces an output about which partial sum term is to be evaluated first, the necessasy inner and cross products are generated to evaluate the complete partial sum term. During this computation phase, the sorter as well as the adder blocks are not clocked. If there is a data dependency, the Once a partial sum term is evaluated completely, the sorter is clocked to produce the next partial sum term and at the same time, the adder pre-normalize block is clocked. The add and rounding blocks are clocked in the consucutive cycles from the adder pre-normalize block is clocked, to complete the accumulation. The internal flip-flops holds the data values if the stages are not clocked.

4.2.2 Reconfigurability

The data correction unit is capable of adapting itself to a variety of operations in addition the error correction operation. It can be programmed to work as a dedicated integer/floating point multiply and accumulate unit without clocking the hardware sorter unit. It can adapt to a new rounding scheme where the rounding bit is forced to zero bypassing the entire post normalization blocks and complete with computations in fewer clock cycles. The microcontroller can take control of all hardware resources to operate as a general-purpose floating-point co-processor to perform filtering operations and data fusion algorithms.

4.2.3 Perturbation Analysis

Incremental theorem for functions [17] describe about the output of a function if there is a small change in the inputs. If $Y = f(X_1, X_2)$, and there is a change $\Delta X_1 = X_1 - X_{10}$ in the input X_1, then the output is given by $Y = Y_0 + \frac{\partial f}{\partial X_1} \cdot \Delta X_1$ if the perturbation in the input ΔX_1 is small. If one of the inputs change slightly during the operation, this principle can be used to compute the change in the output rather than computing the output again. Software assistance is needed for calculating the perturbation of the input signal and in determining whether the approximation is close enough to the actual output.

4.2.4 Energy Efficiency

Since the operating frequency of the correction engine is limited to 40 MHz, the power measurements are taken by simulating the design at 40 MHz. All possible input combinations are applied to each cell in the library, and the power dissipation is calculated. The number of times a particular gate is used, is counted and multiplied with the power

consumption. Performing this calculation for all the cells in the library, yielded the typical power consumption. The correction engine takes about 28 clock cycles to complete the multinomial shown if Equation 4.2, and consume about 4.47 nJ from the battery. it is assumed that the leakage power is very small when compared to the switching power.

The effect of having the hardware sorter comes to play, when the correction equation contain many partial sum terms. The hardware sorter reduces the number of computations in the multinomial by making a trade off in accuracy, and improves the energy efficiency of the correction engine. Usually, the data precision and speed performance are not of utmost importance in the sensor based battery powered system applications, when compared to the energy demands of the system. Hence this method will be an effective one, when compared to the general purpose floating point hardware units, where data accuracy is not compromised, and a certain level of performance is guaranteed. Most of the application specific controllers in sensor based microsystems support floating point operations in software rather than in hardware. These routine-driven softwares have computational and communication overheads in the correction engine process making them unfriendly for a system, which predominantely wants to stay in sleep mode to save battery life. This reconfigurable correction engine, as a single system meets the requirements of both high accuracy and low energy demands (not simultaneously though), making it an unique product in the growing environmental and bio sensor microsystems.

4.3 Hardware Sorter

The significance of a particular partial sum term in the error correction multinomial is approximately estimated from the exponent value of the input signals and the correction coefficient. The partial sum terms are rearranged in ascending order of their significane and evaluated and accumulated to form the final sum. This avoids the the error due to

alignments in the floating point addition operation. However, if the accumulation is started from the term of most significance, a coarse result can be achieved in fewer computations saving time and energy involved in the floating point computation scheme. Arranging the partial sum terms in an order is sorter is necessary to produce accurate or faster results. For the correction engine design, a hardware sorter working on the principle of assigning ranks to integers on the fly is developed. The following sections describe the algorithm and the working principle of the sorter.

4.3.1 Sorting Algorithm

In this scheme, each incoming integer is associated with a rank, which determines its position in a set of integers. When a new integer arrives to the network, it does not have a rank. The rank of the new integer among the existing integers is determined by comparing it against all the existing integers. Then rank of all the integers whose rank is greater or equal to the incoming integer is incremented to maintain the uniqueness of the rank. If the integers are called in the order of their ranks, a sorted list is produced. In this methodology, swapping the integers inside the network is avoided, the main contributor of increased processing time and energy dissipation. The disadvantage of the sorter is that the input/output are given/taken once in a clock cycle(i.e. it takes n clock cycles to sort n integers). However, since input for the sorter comes from the exponent predictor block and the output is fed to the Data Control block, all working in a pipeline, the designed sorter fits well for the correction engine design.

4.3.2 Sorter Architecture

The sorter (shown in Figure 4.4) is capable of sorting sixteen 8-bit wide unsigned integers given one integer in a clock cycle. It has sixteen 8-bit registers to store the integers and a comparator associated with each integer. The comparators compare the existing

integers with the incoming integer and produce a 0 if the existing integer is smaller than the incoming integer else it produces a 1. If we count the 0's from all the comparators, it determines the rank of the incoming integer. The result of all the comparators are stored in a 16-bit register, which has a special property that the result of any comparator can be stored in any bit position. The bit position of the comparator's result is determined the rank of the existing integer stored in the 4-bit index register. With this arrangement, the rank determination problem is reduced to a lead zero detection problem. The sorter has a lead zero detector for determining the rank of the incoming integer. Once the lead zeroes are detected and storeed as the rank for the incoming integer, the rank is compared against all the existing integer's ranks. The ranks higher than the incoming integer's rank are incremented. Hence the new integer is inserted in the array of ordered integers. Each integer has a data valid bit associated with it, so that the comparision is performed only when there data valid bit is enabled.

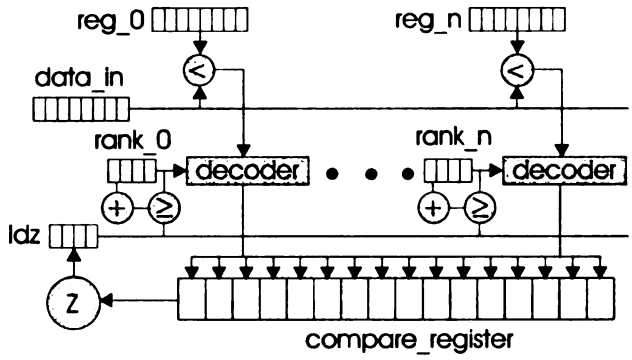


Figure 4.4: Architecture of the sorter

4.3.3 Sorter Operation

The sorter is initialized by setting the *input_rst* signal (Figure 4.5) which resets the *store_index* counter. In consecutive clock cycles, the *store_index* is incremented. Each time an integer arrives to the network, the incoming integer is stored in the register indicated by the contents of the *store_index*. The same integer is also stored in the *data_in* register to compare it with the already existing integers. The *reg_index* associated with the first integer is assigned to 0 ($4'b0000$ as the *reg_index* is 4-bit wide) and the *data_valid* bit associated with that register is set, so that the result of the compare operation is stored in the 0^{th} oposition in the *compare_reg*.

When the next integer arrives, the incoming integer is stored in the next register. The new integer is compared with the already existing integer. If the new integer is bigger than the old integer, the comparator will produce a 0. The leading zeros in the *compare_reg* is counted by the lead zero detection network. Since the leading zeros(1) is greater than the *reg_index* of the first integer, it is not modified this time. The leading zeros is stored in the *reg_index* of the second integer.

If a number in between the first and the second integer arrives to the network, the third integer is stored in the 3^{rd} register. The already existing integers are compared against the incoming integer. The comparator result of the first and second integers goes to the 0^{th} and 1^{st} bit positions. This time also the leading zeros in the *compare_reg* will be counted as 1 and the *reg_index*es with entries greater than or equal to the leading zeros will be incremented. Hence the *reg_index* of the first integer will not be incremented while the *reg_index* of the second integer will be incremented. The leading zeros will be stored in the *reg_index* of the third integer. Now if the integers are recalled by *reg_index* values, they will be sorted in ascending order. The sorter was tested with a typical input pattern and the functional simulation is shown in Figure 4.5.

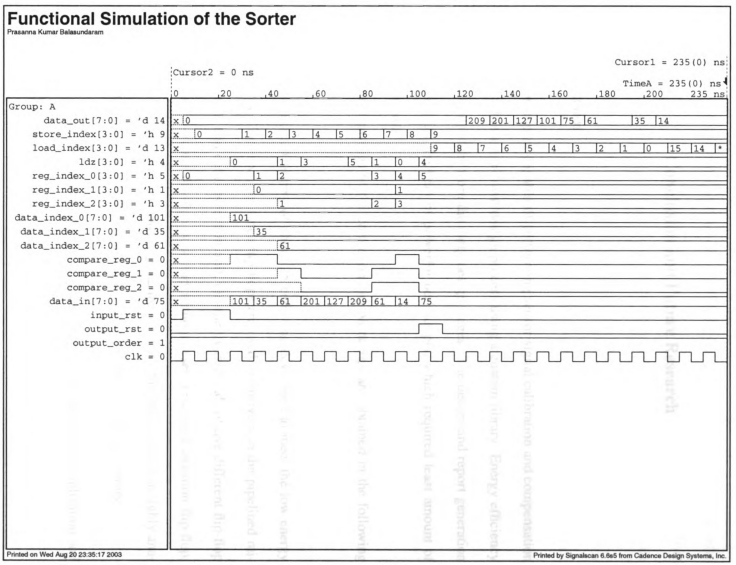


Figure 4.5: Functional Simulation of the Sorter

CHAPTER 5

Conclusion and Future Research

5.1 Conclusion

A correction engine capable of performing sensor signal calibration and compensation was implemented in a top-down design process using a custom library. Energy efficiency was given a priority right from the early design stages. The design and report generation were performed with the help of powerful perl scripts, which required least amount of manual interventions.

Energy/Power savings in the correction engine design are obtained in the following ways:

- An optimized cell library of about 25 cells was developed to meet the low energy constraints. For example, since flip flops are used extensively in the pipelined microprocessor and data correction unit, the energy demands of five different flip-flop structures (in-house flip-flop with and without reset, push-pull isolation flip-flop, transmission gate flip-flop and a regular master-slave flip-flop) were thoroughly analyzed and the two most efficient structures were included in the cell library.

- Efficient multipliers and adders are used to perform the integer multiplication and addition process.

- The expoenent term of the floating point quantity is exploited to order the partial sum terms even before the multiplication and addition process. This novel method saves energy in computing the small terms in the multinomial which does not contribute to

the final result. When the accuracy is of a concern, the accumulation can be started from the term of least significance even avoiding the error arising in the floating point alignment process. Thus the correction engine design meets the requirements of both the ends. Predicting the magnitiude of the data using the exponents of floating point values is an unique contribution to the scientific community.

- The design can adapt to a novel rounding scheme which forces the rounding bit to 0 and gains a clock cycle by shutting down an entire rounding block. This is an unusual way of gaining performance for the scientific community.

- Assigning ranks on the fly algorithm was used to re-order the terms in the multinomial according to their significance. The data movements in the sorter are kept to a minimum (no swapping between the contents of registers) to keep the energy dissipation under control.

- Perturbation calculations which minimize the order of the sum terms in the multinomial helps high order equations to be evaluated using hardware without software interventions contribute further to the low energy objective.

5.2 Future Research

- The interactions between the microcontroller and the correction engine can be well studied and optimized.

- Moving to a technology with more interconnect metal layers and smaller feature size Silicon on Insulator technologies will give designs of high densities and can further contribute to the low energy objective.

- Formal verification techiniques and Design for Test techniques will improve the reliability of the design process and identify the faults in the circuit and checking the consistency of the design in avrious stages of the design flow.

- Content Adressable Memory [10] locations for storing inner and cross products for a particular partial sum term in the multinomial would reduce evaluating the same terms twice.

- Right now, more burden is placed on the hardware to resolve the hazards and to schedule the resources. The load can be moved to the compiler to avoid the stall cycles in the operation.

APPENDIX A

Design Flow With AMSAC Library

The cell library has a Timimg Library Format (TLF) file to help the synthesis tool to perform logic synthesis, and an LEF file to perform the physical design. Both TLF and the LEF file can be segmented into two sections. In the first section, information pertaining to all the cells in the library is provided, and in the second section, information pertaining to a cell is given. New cells can be added to the library with little or no changes to the rest of the cells. A portion of the TLF file and the LEF file is provided in this appendix to get an insight in the library development process.

A.1 Using the TLF file

Ambit Buildgates can be invoked in any workstation in MSU by running these commands in a console/xterm window: *source $ SOFT/spr40* , followed by *ac_shell -gui&* (these commands are subjected to change, and contact the CAD support/unixadmin if case of any problems).

- The TLF file can be read by Ambit Buildgates using the command *read_tlf amsac_lib.tlf* (in ac_shell prompt) by keeping the *amsac_lib.tlf* in the Ambit Buildgates running directory.
- The verilog source file can be read using the command *read_verilog test.v.*
- After setting the timing constraints, a generic design can be designed using the command *do_build_generic -all* . Then the generic design can be mapped to the library using by running *do_optimize* in the ac_shell prompt.

- A gatelevel netlist in verilog format can be obtained from Ambit Buildgates by executing *write_verilog -hier test.vg*. This completes our synthesis process and the gatelevel netlist can be fed as input to the physical design tool.

A.2 Sample TLF file

```
/* Library for Synthesis --
Copyright reserved by Advanced Micro Systems and Circuits Laboratory
Michigan State University

Author: Prasanna Balasundaram

This file will be used for synthesis using Ambit Build Gates.

Version: 1.02  11/03/2003
-Added the load capacitance of the gates which read zero before;
if the cap values are 6+ digits, it was written from the netlist directly
else it was added from the thesis report.
Version: 1.01  Date Unknown
-All cells in the library are recognized by the synthesis tool.
-synthesized netlist functionally matches the source.
*/

header(
  library("amsac_lib")
  date("Tue Feb 25 11:15:35 2003")
  vendor("Michigan State University AMSAC Lab")
  environment("com1c_tt_n_n")
  technology("AMIC5N 0.3um")
  version("1.02")
  tlf_version("4.3")
)

Properties (

  temperature(25)
  voltage(3.0)

  /* multipliers and k-factors */

  proc_mult(1.0)
  temp_mult(1.0)
```

```
  volt_mult(1.0)

  /* threshold definitions */
  table_input_threshold (0.5)
  table_output_threshold (0.5)
  table_transition_start (0.1)
  table_transition_end (0.9)
//  for_cell(seq for_pin(input slew_limit(warn(2.0) error(2.0))))
//  for_cell(comb for_pin(input slew_limit(warn(2.0) error(2.0))))

  /* defaults */
  load_limit(100.0) /* max output load */
)

/* additional header data */

/* end of header section */
/* ------------------------------------------- */
cell (nand2
  /* cell properties */
  /* constraint models */
  /* timing models */
timing_model (td_a10_y01_b1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.1222 0.1337 0.1343 0.1617 0.1430 )
(0.1361 0.1429 0.1429 0.2113 0.1835 )
(0.1996 0.2084 0.1978 0.2129 0.2509 )
(0.3292 0.2655 0.3752 0.2848 0.4019 )
(0.3042 0.3046 0.3093 0.3701 0.5635 )
)))

timing_model (td_b10_y01_a1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.1085 0.0893 0.0969 0.1354 0.1427 )
(0.1285 0.1243 0.1309 0.1493 0.1777 )
(0.1473 0.1504 0.1539 0.2143 0.2463 )
(0.2005 0.2223 0.3214 0.3332 0.3019 )
(0.3478 0.2562 0.3107 0.3903 0.4563 )
)))

timing_model (td_a01_y10_b1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
```

```
(0.2064 0.2586 0.2580 0.2563 0.3498 )
(0.2216 0.2129 0.2363 0.2916 0.3588 )
(0.2783 0.2941 0.3120 0.3527 0.3823 )
(0.3127 0.3412 0.3309 0.4188 0.4606 )
(0.4270 0.3475 0.3460 0.4755 0.5042 )
)))

timing_model (td_b01_y10_a1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.1851 0.2189 0.2151 0.2682 0.3423 )
(0.2477 0.2350 0.2297 0.2719 0.3727 )
(0.3199 0.3340 0.3446 0.3898 0.4677 )
(0.4322 0.4413 0.4512 0.5526 0.6302 )
(0.4697 0.4872 0.5068 0.6789 0.8177 )
)))

timing_model (ts_a10_y01_b1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.1554 0.1535 0.1576 0.2070 0.2758 )
(0.1733 0.1669 0.1735 0.2198 0.2642 )
(0.2529 0.2584 0.2791 0.2998 0.3282 )
(0.3655 0.4362 0.3833 0.4058 0.4772 )
(0.4756 0.4764 0.6707 0.5750 0.6333 )
)))

timing_model (ts_b10_y01_a1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.0973 0.1421 0.1538 0.1502 0.2103 )
(0.1420 0.1521 0.1658 0.2124 0.2595 )
(0.2311 0.2463 0.2588 0.2917 0.2784 )
(0.2909 0.3753 0.3406 0.3748 0.4284 )
(0.5166 0.5321 0.5135 0.6225 0.6190 )
)))

timing_model (ts_a01_y10_b1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.3012 0.3316 0.3519 0.4498 0.4909 )
(0.2817 0.3222 0.3518 0.4358 0.5818 )
(0.3436 0.3455 0.3574 0.4121 0.5444 )
(0.4183 0.4128 0.4623 0.4862 0.5995 )
(0.4866 0.5302 0.5542 0.6594 0.7938 )
```

```
)))

timing_model (ts_b01_y10_a1
(spline (input_slew_axis 0.1 0.2 0.5 1 2)
(load_axis 0 1 2 5 10) (
(0.2913 0.3165 0.3539 0.4473 0.5960 )
(0.2817 0.3239 0.3446 0.3696 0.5802 )
(0.3448 0.3523 0.3788 0.4842 0.6753 )
(0.4516 0.4548 0.4933 0.5644 0.6153 )
(0.6158 0.6389 0.6183 0.7345 0.9436 )
)))

  pin(A pintype(input) capacitance(0.719819993816823))
  pin(B pintype(input) capacitance(0.719819993816823))
  pin(Y pintype(output) capacitance(1.80803994797362) Function(!(A\&B))
)

  /* path definitions */
  Path(A => Y 10 01 Delay(td_a10_y01_b1) Slew(ts_a10_y01_b1))
  Path(A => Y 01 10 Delay(td_a01_y10_b1) Slew(ts_a01_y10_b1))
  Path(B => Y 10 01 Delay(td_b10_y01_a1) Slew(ts_b10_y01_a1))
  Path(B => Y 01 10 Delay(td_b01_y10_a1) Slew(ts_b01_y10_a1))
)
```

A.3 Using the LEF file

Envisia Silicon Ensemble can be started from any workstation by running the following commands in the console/xterm window: *source $SOFT/dsmse53* , *source $SOFT/ic446* and *sedsm -m=96*.

- In the command prompt of the Silicon Ensemble, the LEF file is imported to the database using *INPUT LEF FILENAME "amsac_lib.lef" REPORTFILE "importlef.rpt";*

  .

- Special variables are set using the following commands:

  SET VAR INPUT.VERILOG.POWER.NET "vdd!";

  SET VAR INPUT.VERILOG.GROUND.NET "gnd!";

SET VAR INPUT.VERILOG.LOGIC.1.NET "vdd!";

SET VAR INPUT.VERILOG.LOGIC.0.NET "gnd!";

SET VAR INPUT.VERILOG.SPECIAL.NETS "vdd! gnd! clk";.

- A sample verilog file with all the cells in the library is created and imported to the database. These files need not have functional descriptions, but should have matching pins with the LEF. *INPUT VERILOG FILE "../verilog/amsac_lib.v" LIB "verilog_lib" ;.*

- The design (synthesized gate level netlist) is imported to the Silicon Ensemble by executing *INPUT VERILOG FILE "test.vg" LIB "cds_vbin" REFLIB "verilog_lib " DESIGN "cds_vbin.name_of_the_top_module:hdl" ; .*

- Floorplanning is performed by the command: *FINIT FLOOR rowu 0.35 rowsp 6000 blockhalo 2000 a 1 xio 30000 yio 30000 ;.*

- *IOPLACE AUTOMATIC STYLE EVEN ;* places the pins along the periphery in random. The IO constraint file can be modified to place the pins in the desired locations.

- If the design is not constrained much, running the commands *QPLACE NOCONFIG ;* and *WROUTE NOCONFIG ;* should complete the place and route process.

- The design can be exported in the LEF, GDSII and DEF (Design Exchange Format) formats using the following commands:

  OUTPUT GDSII MAPFILE amsac_lib.map FILE test.gds2 ;

  OUTPUT DEF FILENAME "test.def" ;

  OUTPUT LEF BLOCK FILENAME "test.lef" MACRONAME name_of_the_top_module ; .

- The design can be imported to dfII by importing the DEF, followed by importing the GDSII by keeping the layout view open. The source netlist for the Silicon Ensemble can be imported to a schematic and the LVS can be performed here.

A.4 Sample LEF file

```
VERSION 5.3 ;
NAMESCASESENSITIVE ON ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;
UNITS
  DATABASE MICRONS 1000 ;
END UNITS

LAYER nwell
  TYPE VIRTUAL ;
END nwell

LAYER active
  TYPE MASTERSLICE ;
END active

LAYER poly
  TYPE MASTERSLICE ;
END poly

LAYER cc
  TYPE CUT ;
  SPACING 0.9 ;
END cc

LAYER metal1
  TYPE ROUTING ;
  DIRECTION HORIZONTAL ;
  PITCH 3  ;
  WIDTH 0.9 ;
  SPACING 0.9 ;
  RESISTANCE RPERSQ 0 ;
  CAPACITANCE CPERSQDIST 0 ;
  CURRENTDEN 0 ;
END metal1
```

```
LAYER via
  TYPE CUT ;
  SPACING 0.9 ;
END via

LAYER metal2
  TYPE ROUTING ;
  DIRECTION VERTICAL ;
  PITCH 2.4  ;
  WIDTH 0.9 ;
  SPACING 0.9 ;
  RESISTANCE RPERSQ 0 ;
  CAPACITANCE CPERSQDIST 0 ;
  CURRENTDEN 0 ;
END metal2

....

MACRO nand2
  CLASS  CORE ;
  FOREIGN nand2 0.000 0.000 ;
  ORIGIN 0.000 0.000 ;
  SIZE 9.600 BY 21.000 ;
  SYMMETRY X Y  ;
  SITE CoreSite ;
  PIN A
    DIRECTION INPUT ;
    PORT
      LAYER metal1 ;
        RECT 1.800 8.400 3.300 9.600 ;
    END
  END A
  PIN gnd!
    DIRECTION INOUT ;
    USE GROUND ;
    SHAPE ABUTMENT ;
    PORT
      LAYER metal1 ;
        RECT 1.800 0.000 3.000 5.100 ;
        RECT 0.000 0.000 9.600 3.000 ;
    END
  END gnd!
  PIN B
    DIRECTION INPUT ;
    PORT
```

```
      LAYER metal1 ;
        RECT 6.300 11.400 7.800 12.600 ;
    END
  END B
  PIN vdd!
    DIRECTION INOUT ;
    USE POWER ;
    SHAPE ABUTMENT ;
    PORT
      LAYER metal1 ;
        RECT 1.800 14.250 3.000 21.000 ;
        RECT 6.600 14.250 7.800 21.000 ;
        RECT 0.000 18.000 9.600 21.000 ;
    END
  END vdd!
  PIN Y
    DIRECTION OUTPUT ;
    PORT
      LAYER metal1 ;
        RECT 4.200 3.900 5.400 16.950 ;
        RECT 4.200 3.900 7.800 5.100 ;
    END
  END Y
END nand2

END LIBRARY
```

BIBLIOGRAPHY

[1] *IEEE Standard for Binary Floating-Point Arithmetic ANSI/IEEE Std 754*. IEEE Press, 1985.

[2] *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*. IEEE Press, 1997.

[3] Cadence Design Systems. *Timing Library Format Reference*, October 2000.

[4] A.V. Chavan. An integrated high resolution barometric pressure sensing system. Technical Report SSEL-313, University of Michigan, 2000.

[5] Yoshikoru Yoshii et al. Integrated software calibrated cmos pressure sensor with mcu, a/d converter, d/a converter, digital communications port, signal conditioning circuit and temperature sensor. In *Proceedings of Transducers*, 1997.

[6] K.F. Lyahou, G. van der Horn, and J.H. Huijsing. A noniterative polynomial 2-d calibration method implemented in a microcontroller. *IEEE Transactions on Instrumentation and Measurement*, 46(4):752–757, 1997.

[7] O. Machul, D. Hammerschmidt, W. Brockherde, and B.J. Hosticka. A smart pressure transducer with on-chip readout, calibration and nonlinear temperature compensation based on spline-functions. In *IEEE Integrated Solid-State Circuits Conference*, pages 198–199, San Francisco, 1997.

[8] D. Markovic, B. Nikolic, and R.W. Brodersen. Analysis and design of low-energy flip-flops. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED'01*, pages 52–55, Huntington Beach, CA, August 6-7, 2001.

[9] S. Microsystems. Numerical computations guide, 1991.

[10] H. Miyatake, M. Tanaka, and Y. Mori. A design for high-speed low-power cmos fully parallel content-addressable memory macros. *IEEE Journal of Solid-State Circuits*, 36(6):956–968, June 2001.

[11] M.L.Dunbar. Single chip asics for smart sensor signal conditioning. In *Proceedings of WESCON*.

[12] M. Mozek, D. Vrtacnik, D. Resnik, U. Aljancic, M. Cvar, and S. Amon. Calibration and error correction algorithms for smart pressure sensors. In *IEEE MELECON*, Cairo, Egypt, May 7-9 2002.

[13] G. C.M. Meijer P.C. de Jong. A high-temperature electronic system for pressure-transducers. *IEEE Transactions on Instrumentation and Measurement*, 49(2):365 – 370, April 2000.

[14] R.L. Schwartz and T. Christiansen. *Learning Perl,*. O'Reilly and Associates, November 1993.

[15] V. Stojanovic and V. Oklobdzija. Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems. *IEEE Journal Solid-State Circuits*, 34(4):536–548, April 1999.

[16] E. Swartzlander T. Callaway. Power-delay characteristics of cmos multipliers. In *13th IEEE Symposium on Computer Arithmetic*, Asilomar, California,USA, July 6-9 1997.

[17] G. Thomas and R. Finney. *Calculus and Analytic Geometry*. Addison-Wesley, 9th edition, 1996.

[18] H.K. Trieu, M. Knier, O. Köster, H. Kappert, M. Schmidt, and W. Mokwa. Monolithic integrated surface micromachined pressure sensors with analog on-chip linearization and temperature compensation. In *The Thirteenth Annual International Conference on Micro Electro Mechanical Systems*, volume 13, pages 547–550, Piscataway,NJ, 2000.

[19] J. Zhang, J. Zhou, P. Balasundaram, and A. Mason. A highly programmable sensor network interface with multiple sensor reaout circuits. In *Proceedings of IEEE Sensors 2003*, Toronto, Canada, Oct 22-24 2003.