A Domain-Specific Cell Based ASIC Design Methodology for Digital Signal Processing Applications

Beibei Ren, Anru Wang, Joyopriya Bakshi, Kai Liu, Wei Li, Wayne Dai
CAD Lab, School of Engineering
University of California at Santa Cruz
1156 High Street, Santa Cruz, CA 95064
emails: {bbren, wanganru, jbakshi, lauka, ldragon, dai}@soe.ucsc.edu

Abstract

This paper proposes an innovative domain-specific cell based ASIC design flow to narrow the performance gap between the full custom ASIC design method and conventional standard-cell based ASIC design method. The flow can improve the design performance and still preserve the efficiency of the standard ASIC design flow. Targeting on digital signal processing applications, a domain-specific cell library is provided to augment of standard cell libraries. Experimental results of designing macros such as FFT, FIR etc. are shown in the paper. Based on this methodology a 64-tap FFT can achieve up to 24X performance improvement, with the Power $\times$ Delay $\times$ Area (PDA) criteria, over the conventional designed ASICs.

1. Introduction

The hand-crafted fully-custom application-specific integrated circuit (ASIC) design methodology usually deliveries chips with lower power consumption, 5 to 8 times higher speed [3] and more compacted chip layouts than conventional standard ASIC design method. On the other hand, automatic standard ASIC design flow enables more efficient design process. Based on pre-characterized and silicon verified standard cells, the emerging of design automation technology enables standard ASIC have many advantages over full custom designs, such as fast-turn-around time, less design resource requirement, especially human resources, easy for technology migration, high productivity, and thus low prices. How to close the performance gap between these two methodologies? What is the best way to design an ASIC that has performance competiable with full custom ASICs and still preserves the efficiency of standard ASIC designs? These are the issues gaining more and more concerns in nowadays design processes.

Chinnery et. al. analyzed the problems of the standard-cell based ASIC design in [4]. The authors summarized that the factors contribute to forming this performance gap were lying on architecture and logic design, floorplanning and placement, logic styles, clock timing overhead, and transistor sizing.

While researches are mainly focusing on improving the design tools and front-end architecture level optimizations of ASIC designs, developing a design flow significantly improves both the design quality and productivity is a very important issue.

Northrop et. al. proposed a semi-custom design flow in [8]. The flow combines fully-custom circuit design, automatic cell layout generation and post-layout cell transistor sizing to achieve high performance with high productivity. The approach adopted an automatic transistor sizing tool, EinsTuner, to change the cell driven strengths during the post-placement stage, according to the actual cell load and parasitic information. Along with a script-based cell layout generation tool, this approach can significantly reduce the design time, from circuit design to physical output, to up to 2.5X shorter without sacrificing the design quality. Unfortunately, the technology and tools presented in this paper are not available commercially.

[5] and [13] proposed automatic design flows focusing on post-layout cell resizing. Both approaches showed performance improvements on the ASIC designs. The drawback of these approaches lies in their capabilities of handling large scale designs. And the design time had been significantly sacrificed.

Besides modifying the conventional standard-cell based ASIC design flow, [10] and [13] proposed enhancing ASIC performance by improving the cell libraries. Studies showed that a library with proper set of cell types can dramatically affects the design qualities. Moreover, [2] proposed a design-specific, "flex-cell" based design methodology. The ideas of grouping basic logic cells together into a set of *flex-cells* and on-the-fly generating the cell layouts are very

1530-1591/04 $20.00 (c) 2004 IEEE

interesting. However, many issues such as flex-cell pattern recognization, design automation etc. need to be further addressed.

In this paper, we propose an automatic domain-specific cell based ASIC design methodology, focusing both on the front-end design algorithm and the back-end cell library. The flow provides a group of cells, specifically designed for digital signal processing (DSP) domain applications, as the compensation of the standard cell library for synthesis. DSP applications usually share many common features such as intensive datapath macro usage and highly regular bitslice structure. Sometimes, 30 to 60 percent of cells in whole design are datepath related [14]. Many macros such as adders, multipliers etc. are widely used in almost every applications. Our design flow was conducted focusing on how to improve the performance of these macros and the datapath units, while still preserve the automatic feature of cell-based ASIC design procedure.

Section 2 gives an overview of our DSP domain-specific cell based ASIC design flow. Three major steps, domain-specific cell library generation, domain-specific macro generation and useful skew-based clock tree insertion will be further explained and introduced in section 3, 4 and 5, respectively. Experimental results will be presented in section 6. Section 7 will draw the conclusion and describe the future works.

2. Domain-Specific Cell Based ASIC Design Flow

The development of domain-specific cell based design flow follows four major rules. First, the flow is developed to improve the performance of the designs in DSP domain, i. e. a domain-specific design flow. Second, the flow should employ as many commercialized design tools as possible, and should be able to seamlessly integrate with industry standard tool flow. Hence the flow has the capability to be easily transferred for any third party usage. Furthermore, the flow should maintain the automatic feature as in standard ASIC design flow. The last and yet important, the optimization techniques should be foundry and technology independent, which allows the proposed design flow to fit in different design technologies.

Figure 1 illustrates the major steps of our design flow. Three optimizations were included in the flow:

- Domain-specific macro generation using *DesignWare* from *Synopsys* during front-end synthesis;
- Domain-specific cell library design, including the layout generation, the timing/power library generation, the cell function description and the LEF generation. The libraries will be sent to each design step to work with standard cell library.

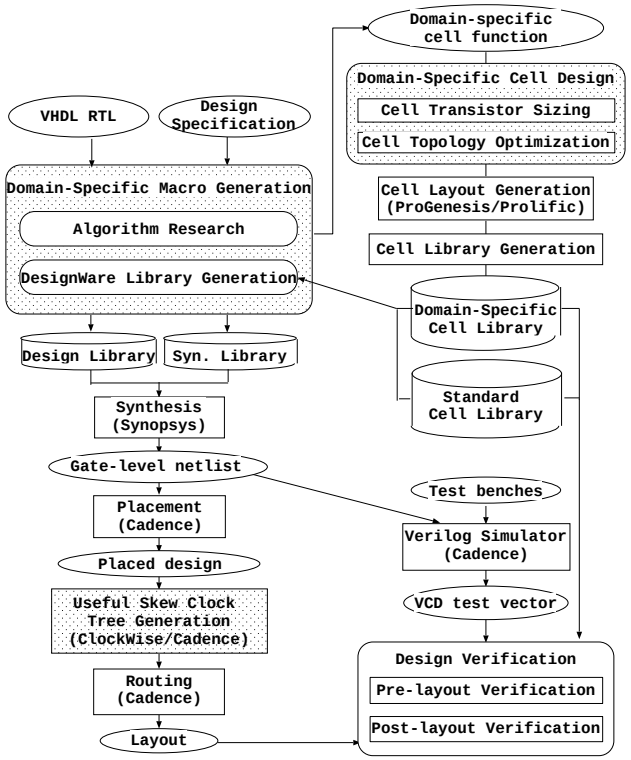

Figure 1. An innovative domain-specific cell based ASIC design flow.

- Useful skew based clock tree generation.

DSP applications usually have large varieties on design requirements depending on their different usages. Many designs such as communication devices or human embedded electronic assistant devices etc. treat low power consumption as their highest prior design goal, while other designs such as real-time DSP applications require high speed. Therefore, the front-end stage of the design flow is still based on logic synthesis, in order to meet different design requirements.

One important factor contributes to the performance gap between standard ASIC and full custom ASIC is the logic design. The fact that the synthesis tool and standard cell library are designed for general-purpose usage makes it difficult for designers to overcome this limitation. In our design flow, we create the design libraries in *DesignWare* for macros that are especially important in DSP applications, i. e. domain-specific macros. The synthesis tool can automatically choose these macros through synthesis libraries and optimize them with different design constraints. Along with the cells specifically designed for these macros, the design quality can be improved and the unnecessary intervention to the high-level design step can be eliminated.

After the circuit style and the transistor sizes are deter-

mined for each cell, a tool named *ProGenesis* from *Prolific Inc.* , is introduced for automatic cell layout generation. Special designed cells have design rule and cell template compatible with standard cells. They will work as the compensation of the standard cells during the entire design flow. Commercial tools such as *DynaCell* and *Abstract* from *Cadence* are employed after layout generation to create necessary libraries for each design and verification steps.

The back-end stage of the design flow includes placement, routing, post-layout RC parasitic extraction, timing and power verification, design rule checking (DRC), and layout versus schematics (LVS) checking etc.

After placement, a commercial tool named *ClockWise* from *Celestry*, now *Cadence*, is used for clock tree generation. In conventional ASIC designs, creating a clock tree with zero skew, i. e. all clock signals arriving at the flip-flops (FFs) at the same time, is a very critical and challenging job. Instead of generating a zero skew clock tree, *ClockWise* creates a clock tree that intentionally generates and utilizes the clock skews, i. e. useful skew. Our experimental results showed that inserting a clock tree with carefully created useful skew can not only improve the design timing by increasing the timing slack, the difference between required data arrival time and actual data arrival time, but also reduce the design power consumption.

The technology used in the flow development and the experiments in this paper is *TSMC0.18*. However, since all above design optimization techniques were foundry and technology independent, the proposed flow can be easily transferred for any third party usage and applied to different technologies or cell libraries.

3. Domain-Specific Cell Library Generation

In this section, an automatic domain-specific cell design flow is presented. As shown in figure 2, flow is divided into three steps, cell circuitry design, cell layout generation and cell characterization.

3.1. Cell circuitry design

Several rules are followed to design library cells. (1) Multiple driven strengths are provided for cells with special functionality or implementation; (2) When determined the cell logic style, no dynamic logic will be considered, due to their difficulties of characterization. (3) Input signals should always connect to a gate, or through an inverter.

We focus our attentions on designing three kinds of domain-specific cells. First are the cells with new logical functions specifically designed for DSP macros. The designs in this domain usually have many common features, such as intensive mathematic usage, especially addition and multiplication, and regular design style, such as bit-sliced

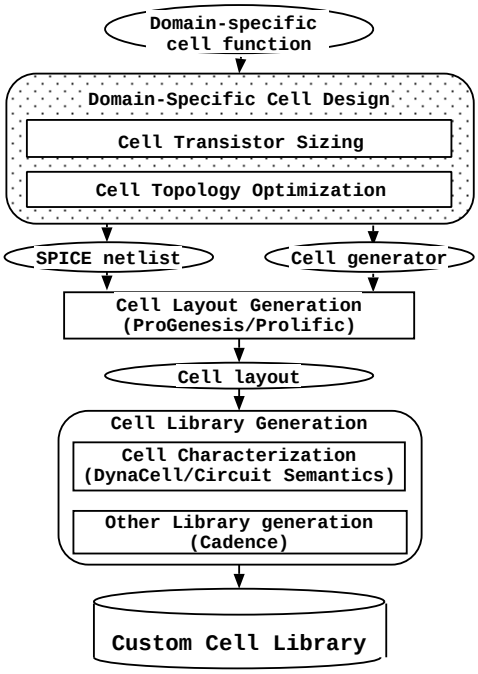

Figure 2. Cell library generation flow.

structure and parallelism etc. Using the cells with functions, transistor sizes and templates that can benefit these particular features will result in better performance than using general-purpose standard cells only. For example, in our experiment, a set of cells, including a 5:1 multiplexer and a 4:2 compressor cell were designed for a low power, high speed multiplier macro [1][15]. We provide at least five driven strengths for each new designed cell.

The second kind of cells are the cells having same functionality but different implementations as standard cells. Depending on different usages, there exist cases where for cells with same functionality, some implementations are preferred than others. Moreover, many logic styles [9] such as complementary passing transistor logic (CPL) etc., usually can reduce the circuit complexity, thus save the power and area without sacrificing delay. In our project, an 1-bit full adder was designed using a modified low power passing transistor logic[11][12], which showed better PDA performance.

Furthermore, to achieve certain driven strength or power consumption requirements, cells with same implementation should have special transistor sizes. Using general-purpose cells provided by standard libraries, these requirements cannot be satisfied. Therefore, the third kind of cells are the cells with same functionality and logical implementation as standard cells, but with different transistor sizes. In our experiments, we added 2 to 3 more driven strengths for 10 fundamental cells. One of them is using smallest sizes for all

transistors. Many DSP macros do not have high speed requirement. For those cases, using these cells will result in power saving and still meet the delay requirement.

3.2. Cell transistor sizing

A set of simulation-based program/scripts were provided for automatically sizing transistors in library. Using a general-purpose hypertext preprocessor language, PHP, a script was provided to create HSPICE circuit files of each cell, with parameterized technology setting, transistor sizes, cell loads, desired delay measurement and input stimulus. The running script contains PHP code for setting up environments, running HSPICE simulation, setting up parameters and generating final simulation result file. The simulation results include the delay of interested input/output signal pair, average power measurement, n/p transistor counts, as well as each transistor widths.

These simulation results will be sent to a sizing program written in MATLAB. The strategy for choosing transistor sizes are as follows:

1. Choose a cell size set S by:
   $S = \{ s_i \in S \mid d_i < D \}$, where d_i is average delay of each cell s_i, D is desiring delay requirement.
2. For each s_i, calculate the a_i as following:
   $a_i = max\{ \sum^{all\ trn_{ij}} floor(trn_{ij}/max\_Trn),$
   $\sum^{all\ trp_{ij}} floor(trp_{ij}/max\_Trp) \}$,
   where $trn(p)_{ij}$ is the size of each n (or p) transistor in s_i, $max\_Trn(p)$ is the maximum n(or p) transistor width allowed in layout without folding. Thus, a_i is an pretty good estimation of cell area.
3. Determine cell candidate set $S_{candidate}$ by:
   $S_{candiate} = \{ s_c \in S_{candidate} \mid s_c \in S \&\& \forall s_j \in S, a_c \leq a_j, c \neq j \}$
4. Final choice s_f is:
   $s_f = \{ s_f \in S_{cadidate} \&\& \forall s_i \in S_{candidate}, d_f < d_i \}$

3.3. Cell layout generation

To generate cell layouts, we employ an automatic compaction-based cell layout generation tool *ProGenesis* from *Prolific Inc*. Taking the cell SPICE netlist as input, *ProGenesis* first creates a .CEL file containing all cell topology information. By matching the SPICE netlist of the cell to the provided default generator list, the procedure is very fast and usually within seconds. When necessary, *ProGenesis* also allows designers to create a cell with their own topology styles, through a graphical topology editor, *ProSticks*. *ProSticks* translates the information of transistor sizes, orders, routing layers, relative positions among the routing trunks and pin positions etc. into stick diagram and generate a cell "generator" file in tcl code.

Next, the compaction engine of *ProGenesis* takes optimized generator, along with pre-setup design rule database file and cell template file to create final cell layout. Experiments showed that, the generated cells can achieve competiable cell area with hand-crafted cell layouts. The average area overhead is 10%.

Note that, although the automatically created cell layouts sometimes have larger area, the design time has been significantly reduced. Moreover, the separation of design rule and cell template with cell topology in *ProGenesis* makes the technology migration much easier. The most time consuming step is the cell topology optimization, which is a manual procedure. However the topology information will be stored in generators and can be reused in any other technologies. Moreover, cells with different transistor sizes share same generator as long as they have same topologies. Therefore, the layout creation time is saved. And it is a foundry and technology independent design process.

3.4. Cell library generation

Besides creating DRC and LVS clean cell layouts, additional libraries such as timing/power libraries, LEF file and cell function libraries are also needed to be generated for design and verification usage. In our flow, most of these steps are finished by using industry proved commercial tools. *DynaCell* from *Circuit Semantics* was used in the cell characterization and *Abstract* from *Cadence* was used in the LEF generation process. Although the cell function library needs to be manually generated, it only needs to be done once.

4. Domain-Specific Macro Generation

Synopsys provides a product family *DesignWare* to let designers create library macros from their own design requirements. With these macros in a design, *DesignWare* can provide designers with advantages of flexibility and controllability. Using our optimized domain-specific macros during the high-level optimization performed by the synthesis tool, such as arithmetic optimization and implementation selection, designers can increase both efficiency and the quality of results by reusing our optimized domain-specific macros when designing DSP applications.

Along with domain-specific cells, the fundamental macros for DSP domain are being developed. They are further linked to the *DesignWare* from *Synopsys* through synthesis library. This new domain-specific design library will work as a compensation of the default synthesis library provided by *Synopsys*. By doing this, the high-level behavior VHDL design can be easily mapped to our im-

proved macro implementations without manual intervention.

The proposed macro design process includes five subtasks: (1) Optimize the algorithms of DSP macros; (2) Realize each macro with proposed algorithm in *Synopsys DesignWare* format; (3) Apply optimization constraints to macros while compiling; (4) Create the synthesis library that make the new macro designs recognizable by synthesis tools; (5) Group all domain-specific macro designs into library package as a compensation of the default *Synopsys* library. Similarly as creating domain-specific cell generators, creating design and synthesis libraries for domain-specific macros only needs to be done once and is technology independent.

5. Useful Skew-based Clock Tree Generation

Clock tree generation is a key factor that affects the performance of ASIC designs. Controlling clock skew is crucial to meeting overall design timing. Ideally, the clock signals should arrive at each flip-flop at same time, i.e. zero clock skew. Due to many practical design issues such as voltage dropping along the wires, crosstalk noise tolerance, resource availabilities etc. , designing a clock tree with zero skew is difficult, especially when the design is complicated. A usual approach is to bound or minimize the global clock skew. Global clock skew is the largest skew value among all pairs of flip-flops within a clock domain, which is simply the difference in arrival times between the earliest and latest arriving clock signals [6].

However, [7] and [16] reported that this approach might produce sub-optimal system timing. In fact, the clock skew is only a local constraint, rather than a global constraint. In our design flow approach, a clock tree generation tool, *ClockWise* from *Celestry Design Technologies Inc.* , now *Cadence*, were adopted. It inserts a clock tree into the design, with intentionally created clock skews, called useful skew.

The advantages of useful skew clock scheduling can be summarized as follows on improving the design performance: (1) Improve overall design slack; (2) Reduce the burdens for logic designers on optimizing the critical path timing; (3) Using useful skew approach, the critical path in global minimum skew approach may not be critical path anymore, thus the power can be saved; (4) Increase the timing safety margin. Hence increase the design tolerance of clock timing uncertainties; (5) Reduce peak current consumption by distributing the flip-flop switching point in the range of permissible skew. Overall, the useful skew approach can surprisingly increase the chip performance. And shorten the timing-closure period.

6. Experiments

Five designs were experimented using the methodology proposed in this paper, 16-bit adder (Adder16), 16x16-bit multiplier (Mult16x16), 64-tap FFT(FFT64), 32-tap FIR(FIR32), and covariance matrix estimation (CME). The transistor counts varies from 0.5K to 1.3M.

The domain-specific cell library includes 44 cells in 18 types, created in two weeks by two personnel with four CPUs, including generating the timing/power library, LEF file, verilog function file etc. The cells were designed with *TSMC0.18* technology cell library. When generating layouts, the usage of metal two layer was allowed in complicated cells and in vertical channel routing only. The technology migration time for these cells, including generating necessary library files is estimated in five days, with two personnel, four CPUs and one license token of DynaCell.

To show the improvement, a set of baseline design was provided for these macros by using default, conventional standard-cell based design flow. The post-layout comparison is summarized in Table 6. The timing were measured by using *Synopsys, Path Mill* and the power were measured using *NVT* from *Celestry*, now *Cadence*. Besides best delay, best area and best power, a PHP script was given for automatically generating design points using synthesis tool with different delay and power targets. Ideally, the best PDA should be chosen from post-layout measurement results. However in this experiment, we found that choosing design points from pre-layout results could still give the correct choices. Figure 3 shows an example delay vs. PDA curve for FFT64. The example layouts of FFT64 were shown in Figure 4.

Optimization Improvement	Post-layout Measurement Comparison			
	Baseline	Best P	Best D	Best PDA
Adder16	1.00X	1.96X	3.07X	2.13X
Mult16x16	1.00X	2.30X	1.84X	1.80X
FFT64	1.00X	9.76X	2.55X	24.69X
FIR32	1.00X	3.47X	2.22X	5.98X
CME	1.00X	5.21X	5.69X	17.56X

Table 1. Optimization improvement summary. Baseline are designs using conventional standard-cell based ASIC design flow.

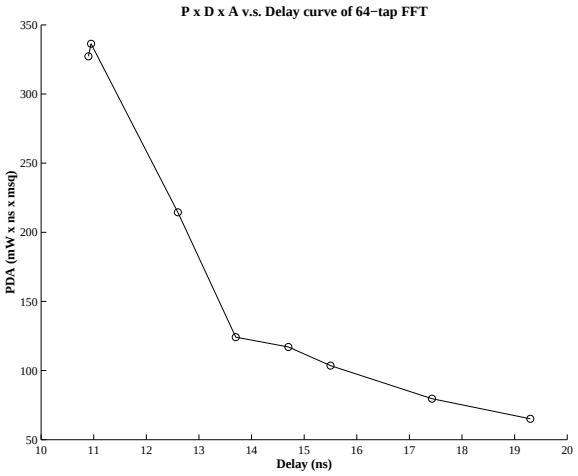

Figure 3. PDA vs. Delay curve of 64 tap FFT design.

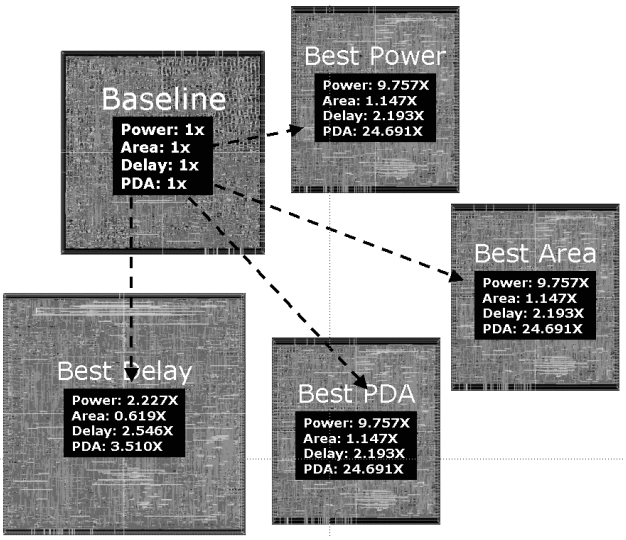

Figure 4. FFT layouts in layout editor, best delay, best power, best area and best PDA results were shown.

7. Conclusion and Future Works

This paper proposed a domain-specific cell based ASIC design flow for DSP domain applications. Three major optimization steps were introduced: domain-specific cell design, domain-specific macro generation and useful skew clock tree generation. Experiments had shown that using this flow the design performance can be significantly improved. The flow is developed with consideration of preserving the automatic feature, adopting as many commercial tool as possible and independent to foundry and technology.

We will continue to improve and complete our domain-specific cell library as well as *DesignWare* macro library. Datapath-driven placement, post-layout cell re-mapping are also important means to further improve the performance.

8. Acknowledgement

This work was supported by DARPA Mission Specific Processing program. RTL description files were provided by Boeing. Some of the baseline data were provided by the research group of Professor Don Bouldin from University of Tennessee.

References

[1] I. S. Abu-Khater and A. B. etc. Circuit techniques for cmos low-power high-performance multipliers. In *IEEE Journal of Solid-State Circuits*, 1996.

[2] D. Bhattacharya and V. Boppana. Automated flex-cell based design. In *Closing gap between full-custom and ASIC*. 2002.

[3] D. Chinnery and K. Keutzer. Introduction and overview of the book. In *Closing gap between full-custom and ASIC*. 2002.

[4] D. G. Chinnery and B. Nikolic. Achieving 550 mhz in an asic methodology. In *Design Automation Conference*, 2001.

[5] M. Cote and P. Hurat. Better cells for better designs. In *Closing gap between full-custom and ASIC*. 2002.

[6] W. Dai and D. Staepelaere. Useful-skew clock synthesis boosts asic performance. In *Closing gap between full-custom and ASIC*. 2002.

[7] J. Fishburn. Clock shew optimisation. In *IEEE Transactions on Computers*, pages 945–951, 1990.

[8] G. A. .Northrop and P.-F. Lu. A semi-custom design flow in high-performance microprocessor design. In *Design Automation Conference*, 2001.

[9] D. Radhakrishnan. Low-voltage low-power cmos full adder. In *IEEE Proc.-Circuits Devices Syst.*, 2001.

[10] K. Scott and K. Keutzer. Improving cell libraries for synthesis. In *IEEE Custom Integrated Circuit Conference*, pages 128–135, 1994.

[11] A. M. Shams and T. D. etc. Performance analysis of low-power 1-bit cmos full adder cells. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2002.

[12] M. Vesterbacka. A 14-transistor cmos full adder with full voltage-swing nodes. In *Signal Processing Systems*, 1999.

[13] M. Vujkovic and C. Sechen. Optimized power-delay curve generation for standard cell ics. In *Design Automation Conference*, 2002.

[14] M. Wang and W. Dai. Proposal for mission-specific processing project, 2001.

[15] A. Weinberger. 4-2 carry-save adder module. In *IBM technical Disclosure Bulletin*, 1981.

[16] J. Xi and D. Staepelaere. Using clock skew as a tool to achieve optimal timing. In *Integrated System Design*, 1999.