A Semi-custom Design Flow in High-performance Microprocessor Design

Gregory A. Northrop
IBM Research
Yorktown Heights, NY 10598
gnorth@us.ibm.com

Pong-Fei Lu
IBM Research
Yorktown Heights, NY 10598
pflu@us.ibm.com

ABSTRACT
In this paper we present techniques shown to significantly enhance the custom circuit design process typical of high-performance microprocessors. This methodology combines flexible custom circuit design with automated tuning and physical design tools to provide new opportunities to optimized design throughout the development cycle.

Keywords
Standard cell, circuit tuning, custom design, methodology.

1. INTRODUCTION
The development of high performance microprocessors requires concurrent design at many levels (logical, circuit, physical) with large teams and tightly interlocked schedules. Often the best design flow is one that most effectively addresses the natural conflicts within this flow (e.g., logic stability vs. timing closure), in contrast to one that simply applies the most modern or aggressive approach in each domain. This paper describes such a case, in the development and use of a semi-custom design methodology which has significantly enhanced several generations of IBM zSeries (S/390) processors [1-3], as well as the IBM POWER4 processor[4]. The coordinated use of a common parameterized gate representation, standard cell generation capabilities, place and route merged with custom physical design, static transistor level timing + formal circuit tuning, and gain-based synthesis have all led to significant improvements in both quality-of-result and time-to-market in the conventional static CMOS design domain.

2. CUSTOM PROCESSOR DESIGN
The circuit design methodology described in this paper was developed and applied over 3 generations of the microprocessor family used in the IBM eServer zSeries (S/390 mainframe) [1-3]. These processors have achieved frequencies in excess of 1GHz in a 0.18u CMOS technology, combining high frequency with a relatively shallow pipeline (~7 stages) and extensive use of millicode [2] to implement the complex S/390 architecture with a RISC-like micro-architecture.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

The physical design of these processors makes extensive use of hierarchy, partitioning the chip into functional units (instruction, FXU, FPU…) and units into macros. There are typically ~6 units and ~200 distinct macros (about 600 total instances), and a macro can have anywhere from 1K to 100K or more transistors. The macro serves as the primary partitioning unit for logic entry (HDL), and a common macro connectivity description is used for both functional models for verification and for circuit design. Boolean verification is used to ensure functional equivalence between macro HDL and a schematic representation, which is verified against the physical design. All macros and units are fully floorplanned objects, and the global wiring (chip and unit) is done hierarchically, using a wiring contract methodology. Macros are all characterized for timing, noise, etc., and represented by models at the global level. Timing rules are generated for all macros using static transistor-level simulation. Global timing is run at both the unit and the chip level, using these macro rules and global wire extractions. The resulting timing is routinely used to generate assertions for all macros, which are used with the static transistor level timing to drive timing closure.

Concurrent design with carefully controlled feedback and iteration are the keys to bringing such a design to closure. Circuit and physical design start as soon as sufficient logic is defined, while the early emphasis is on simulation for functional verification. Early floorplanning and initial circuit design are used to check the cycle time feasibility, and as the design matures, the emphasis shifts from strictly functional verification to logic modification and repartitioning as the primary mechanism for achieving timing closure. This means that the efficiency, turn-around-time, and flexibility of the circuit design methods are as important to the ultimate chip cycle-time performance as is the intrinsic circuit performance.

Circuit implementations of macros fall into 3 classes, with each type occupying about 1/3 of the area: arrays, (cache, table logic) synthesized random logic macros (RLMs) for controls, and full custom dataflow. The dataflow is done predominantly in static, with dynamic circuitry reserved for only extremely critical function, in order to meet power requirements. Traditionally, full custom design is used for arrays, register files, and the dataflow stacks that are typical of the instruction and execution units. Custom design is very effective at optimizing performance and achieving a high area efficiency, particularly where elements are identical across the bit range of the data stack, as hierarchy and careful tiling lead to highly optimal designs. This clearly applies to register files, working registers, muxes, and the like.

However, many of the most labor intensive and critical functions, particularly those that implement more complex numerical functions (adders, incrementers, and comparators) do not make such a compelling case for full custom design. They are far less regular across the stack, more complex, and often do not tile very easily. They are often timing critical, and although the logical function usually has a stable definition early in the design process, the most appropriate circuit architecture may evolve. It is this class of function which is the primary application for the semi-custom design flow outlined below.

3. SEMI-CUSTOM DESIGN FLOW

3.1 Primitive parameterized bookset

The basic building block used in this methodology is a set of parameterized gates, called the primitive bookset, an example of which is shown in Fig 1. These gates are generally a single level of conventional (inverting) static CMOS, with complementary pull-up and pull-down nfet and pfet trees, wherein the parameterization simply scales the pfets with a parameter PW

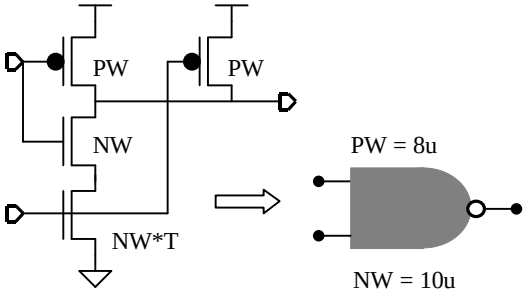

Figure 1. Parameterized gate and sample instance, showing parameters NW and PW. The expressions on the transistors are widths, and T represents an optional taper factor.

and the nfets with a parameter NW. In general, each fet can have an additional fixed multiplier, T, which is used to define multiple flavors of each logic type providing tradeoffs between the delays from each input pin. With the exception of the XOR and XNOR functions, all these primitive gates are a single level of inverting logic. A complete set of represented topologies can be found in Table 1.

Table 1. **List of parameterized logic types**

Logic type	Comments
INVERTER	
NAND2…NAND4	Multiple T
NOR2…NOR3	Multiple T
AOI21 & OAI21	Multiple T
AOI22 & OAI22	
XOR2 & XNOR2	Pass-gate style
MUX2	2-Way transmission gate mux
PGMERGE	G+P*C for adders (restricted AOI21)
DRXOR2	Dual rail xor (true & comp inputs)

Together, these parameterized gates form a basis set capable of covering most of the design space for combinational static circuitry found in a conventional ASIC library, since more complex functions, such as wide ANDs and Ors are typically composed of multiple levels of these gates. Note that this bookset is only a schematic representation; there is no directly associated layout.

A cell generation tool, described in more detail in section 3.5, is designed specifically to produce layout in a row-based standard cell image for arbitrary values of NW and PW for each primitive cell. In addition to its use in semi-custom design, this tool was also used to create a conventional library of discrete sizes for use with synthesis to build the RLMs. This standard cell library had non-parameterized cells with all the conventional views, including timing rules required for synthesis. The sizes were selected by generating a reasonably dense matrix of NW,PW values for each primitive cell and running it through the cell generator. The NW,PW values were cast as a power level (NW+PW) and a rise/fall ratio (PW/NW), also called the beta ratio. The ratio of adjacent power levels was around 1.25, and the range of beta ratios was adjusted to specific rise/fall times. Depending on the complexity of the function, there were from 10 to 25 power levels and from 1 to 4 beta ratios for each primitive type. Including a physically compatible set of cells using low Vt fets, there are ~1200 cells in this library. This type of library, commonly referred to as a "tall thin" library, provides the flexible sizing required for synthesis to achieve maximum cycle time performance in the RLM control logic.

3.2 Overall design flow

The design flow used to implement custom designs built from the primitive bookset is summarized in Fig 2. This flow is largely automated, with most of the work contained in the initial design and the place and route floorplan. After an initial pass, iteration of the design is a relatively rapid process.

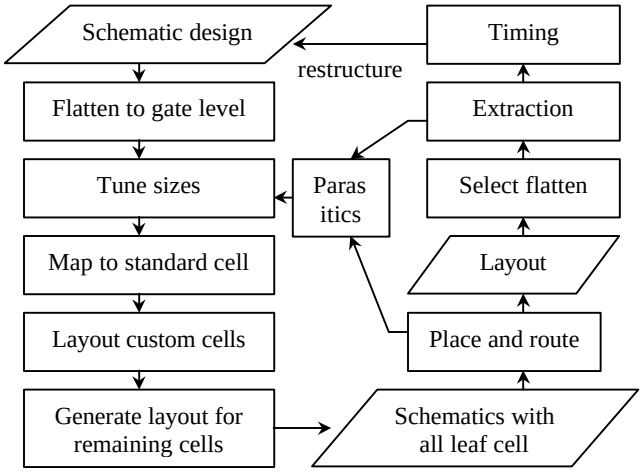

Figure 2. Complete design flow using the primitive bookset. The custom designed schematics can be a hierarchical combination of parameterized and custom designed cells.

3.3 Design flow details

This section catalogs some of the details of each step in the flow from Fig 2. Note that not all steps are needed, and a wide variety of combinations have been used, depending upon the details and needs of each design. The primary goal of this flow is to retain as much detailed control of the design at the

schematic level while using layout automation, particularly place and route, as much as possible.

3.3.1 Schematic design (contents)

The initial schematic design can be all or part of a macro, but generally should encompass function that can be floorplanned in a simple block, and whose components can be routed automatically. It can contain hierarchy, with the assumption that it will be flattened to a set of routable leaf cells going into final physical design. It can contain a mixture of parameterized primitive gates, cells from the standard cell library, and custom cells.

3.3.2 Gate level flattening

Even though the design will be flattened to the leaf cell going into physical design, there can be significant advantage to flattening a hierarchical design in schematic form, particularly when tuning a design with low symmetry in its structure or timing constraints.

3.3.3 Static circuit tuning

Automated tuning of fet widths to optimize the slack of a macro is one of the key elements of this methodology. It helps to take full advantage of the sizing flexibility of the primitive bookset to optimize timing and area. An expanded discussion of the tuning method can be found in section 3.4.

3.3.4 Mapping to the standard cell library

After sizing, often many parameterized gates can be mapped to cells from the standard cell library, allowing only a modest change in fet sizes and a minimal impact on timing. This helps to control data volume since all remaining primitive gates must be generated specifically for that macro.

3.3.5 Inclusion of custom leaf cell physical design

Any required custom leaf cells for the design must be abstracted for place and route. These can be either standard cell image, or other blocks that are pre-placed.

3.3.6 Automated cell generation

Any parameterized cells that remain after step 3.3.4 are generated. Details of the generation process can be found in section 3.5.

3.3.7 Floorplanning, place and route

An interactive floorplanning and place and route environment is used to complete the layout. This environment features:

- Hand constructed floorplan with detailed wire contracts.
- Customizable row configurations for placeable cells, capable of multiple row heights.
- Pre-placement of non-placeable and placeable cells.
- Placement constraints (regions, groups, net weights).
- Code-based or manual pre-routing.
- Grid based router.
- Incremental (EC) placement to retain stable timing when iterating a design.

3.3.8 Cell count and data volume reduction

After completion of place and route, designs that have made heavy use of tuning and cell generation will have a large number of unique cells, many used only once or a small number of times. Selective flattening of these cells in the layout, with a corresponding change back to the parameterized schematic representation, helps strike a balance between a high cell count and the large data volume of completely flat layout.

3.3.9 Parasitic feedback into tuning

The effects of wiring parasitics have become quite important, accounting for as much as 30% of the delay in even moderate size macros. Lumped capacitance, from either wire length (from place and route) or from a subsequent full extraction of the layout, can be merged into the schematic netlist and the circuit re-tuned to compensate for their effect.

3.3.10 Design restructuring & alternate circuits

The combination of automated sizing and timing-based real physical design allows the designer to try multiple restructurings and circuit architectures, and make a confident comparison of the relative quality of each approach. *This part of the flow, and the associated change in the approach to design, are the most important part of this methodology, and the place where the most benefit will be found when it is fully applied.*

3.4 Circuit tuning

A pivotal driver of this methodology is the use of circuit tuning to automate the sizing of transistors. While circuit tuning tools have been applied in CMOS design for a number of years, we believe that EinsTuner [5,6], the tool used here, delivers a quality of result that is very important to the overall improvements in design presented in this paper.

Tuning tools can be divided into 2 broad classes, static and dynamic. Dynamic tuning involves simulation with explicit waveforms and measures (delays and slews), while static tuning formulates the optimization problem through static timing, optimizing slack in the presence of timing assertions. The large, non-bitslice circuits for which semi-custom design is best suited present an impractical problem for dynamic tuning, but are an ideal candidate for static tuning, which can keep track of a large number of critical paths as tuning proceeds.

The EinsTuner static tuner is built on top of a static transistor-level timing tool (EinsTLT), which combines a fast event-driven simulator (SPECS) with a timing tool (Einstimer). The SPECS simulator provides timing information (delay and slew) along with first derivatives with respect to circuit parameters, specifically transistor width. EinsTuner uses this to formulate the optimization problem for solution by a large-scale general-purpose nonlinear optimization package LANCELOT [7], generally optimizing a linear combination of slack and area, nominally treating all fet widths as free parameters. Additional features of this tool that make it effective in a practical design environment include:

- Parasitics (lumped capacitance) from physical design.
- Area modelled as sum of fet widths.
- A fet-width ratioing mechanism, used to constrain fet widths to match hierarchy or gate parameterization.
- Input capacitance, node slew, effective pull-up and pull-down (beta ratio), and min/max fet-width constraints.

- Complete interactive environment (GUI), including size constraint generation and back annotation.

In its current state of development, EinsTuner is capable of tuning in excess of 3000 gates with run times normally < 24 hours, even for large circuits, such as a 64-bit adder (~2000 gates). Experience has shown that tuning results are largely independent of the starting point, meaning that a designer can have a high degree of confidence that the results from a run are optimal for the conditions and design. While heuristic tuning can be faster, such algorithms often need coaching for particular designs, and they offer little certainty as to how close a run is to the "true optimum". This is an important issue when using this methodology to compare and select circuit structures.

3.5 Cell generation

A second key component of this methodology is the use of a cell generator to create layout corresponding to the parameterized gates. This home-brewed tool, called C-cell, is not a general-purpose cell compiler, but rather a script-based system designed to produce optimal layout, but only for the defined parameterized bookset. The definition of the parameterized gate set is tightly integrated into this tool, delivering a framework that supports semi-custom design in a number of ways:

- Generate a set of layouts & associated views for use as a standard cell library, based upon a list of cell specifications: (primitive name, NW, PW).
- Parse a schematic, form a list of cells to generate to replace parameterized gates, minimizing the number of required cells for a maximum allowed deviation in size. Generate cells, and create a modified custom schematic referencing the generated cells.
- Includes a facility to convert between parameterized and standard (RLM library) cells.
- Has an integrated floorplanning aid with an interface to the place and route tool.
- Does layout post-processing, including selective layout flattening and shape trimming.

In the cell generation part of the tool, topology and technology specific code takes as input the gate type, size parameters NW and PW, and a global cell image (row height), and generates layout, after selecting the optimal configuration from a range of finger partitionings and topology options. In practice the measure of optimality is cell area, but factors such as wireability, manufacturability, etc., could also be weighted in the selection. While this system is not capable of implementing an arbitrary gate topology, it has been very successful in the domain of conventional static CMOS, where there are a small number of effective topologies, and optimization of the simplest (nand/nor) types is vital. To date, the effort required to migrate and modify this approach from technology to technology has been easily justified in its use in both the synthesis environment and in semi-custom design.

4. DESIGN EXAMPLE – 24 BIT ADDER

4.1 Adder function, timing, and floorplan

This 24 bit adder is used in the branch target address prediction in the POWER4 microprocessor [4]. The adder performs an addition between the Current Instruction Address (CIA) and the sign-extended immediate operand from the I-cache. The adder includes a 4-way mux to select among 4 possible target addresses: the link register, the sign-extended immediate field, the adder result, and the counter register. The mux controls are from an RLM outside the dataflow stack. The adder and mux, wrapped by latches manually designed separately, form a module. There are eight such 24-bit adder modules to handle 8 instructions delivered by the I-cache in parallel. A 10-way mux (in another macro) selects one of the 8 resulting branch addresses and 2 other sequential addresses, whose lower 12 bits are then sent back to I-cache to fetch the next 8 instructions. This loop path: I-cache->Adder->muxes->I-cache, takes 3 cycles to complete. It is one of the cycle-limiting paths in POWER4 design. The allotted timing budget for the adder is about half a cycle.

The floorplan of the macro containing 8 adder modules is shown

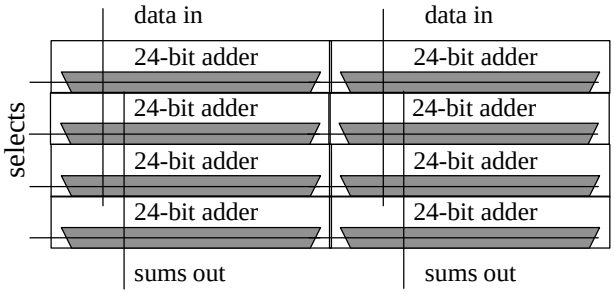

Figure 3. Macro floorplan containing group of 8 24 bit adders. The muxes discussed in section 4.2 are shown in gray.

in Fig. 3. The 8 adders are arranged as 2 stacks of 4 to meet constraints in the chip floorplan as well as wiring congestion. The height is dictated by the unit floorplan and needs to be minimized, while in the width dimension it is fixed at 14 tracks per bit as the rest of the dataflow. The address inputs and outputs flow vertically, while the control signals are from the sides. The mux inputs for the predicted values of the link register and the count register are common to all 8 adders; the CIA is formed by concatenating the word-offset field (0:2) with the I-cache fetch address (IFAR) <38:58> which is also common to all 8 adders.

4.2 Circuit architecture

The 24-bit adder is a static, carry-look-ahead Ling adder [8], which is a stage faster than a CLA adder. The 24 bits are grouped into six 4-bit groups for carry propagation. The local sum is implemented in parallel with the carry, and the result of the carry bit is used to select the final value.

One key circuit decision made early in the design was to hide the mux delay behind the adder. The assumption was that the I-cache data through the adder would be the critical path, while the mux selects and register file data will arrive early. Since the carry bit is the longest path, the partial sums can be pre-muxed with the other 3 inputs and the final result selected by the carry. This led to a skewed ‘late-mux’ design as shown in Fig. 4(a). Frequently it is assumed early in design that controls are non-critical, however that often turns out not to be true, as will be discussed later.

4.3 Circuit tuning and layout

In its final form, the design consisted of 485 gates of the following types and number: inv(164), nand2(63), nor2(31), aoi21(15), aoi22(176), oai21(25), oai22(11). Once the schematic design was complete and verified, the physical design followed much of the flow shown in Fig. 2. These steps were used: gate level flatten, tune, cell generation, place & route, full extraction, re-tune with actual parasitics, and incremental placement and re-route. Since the schematic was flattened before tuning, all gates could be sized independently by EinsTuner. Wiring use was limited to all of M1, and a portion of the M2 and M3 tracks, as some tracks are reserved for the macro level routing through the stack. Cell occupancy was maintained through EinsTuner's area constraint to about 70% of the floorplan area. No mapping to the standard cell library was done, and 297 cells were generated, since only a +/- 5% size variation was allowed. If a +/- 25% size variation had been allowed, this would have still required 143 cells. After place and route, the layout was flattened into shapes since the maximum reuse of any cell was 10, and most cells had only 1 or 2 instances. In addition flattening allowed the use of the trimming process to cut excess metal and poly shapes to reduce parasitics, which was found to improve the delay by an additional 2-3%. The turnaround time was less than a day; usually 2 to 3 iterations were sufficient to bring the design to convergence after a change in design and/or timing assertions.

4.4 Design iterations and timing convergence

The key issue of the semi-custom design is the timing assertions fed to the tuner. As the control signal timing is unknown in the beginning, the timing assertions are largely estimates that may not be substantiated as the chip timing stabilizes. In the 24-bit adder example, the assumption of the mux select timing turned out to be wrong after the first tape-out. The gating path was from the selects, thus the circuit construct in Fig. 4(a) was improper. The mux was re-designed using a balanced AOI-NAND scheme in Fig. 4(b). The new schematic was then re-tuned and iterated based on the correct timing assertions. The new design was completed in a week, and the negative slack was reduced by more than 80 ps, about 20% of the timing budget for the adder. Thanks to the contract based place and route, the adder re-design did not cause any global wiring change.

The other issue is that a new microprocessor design project like

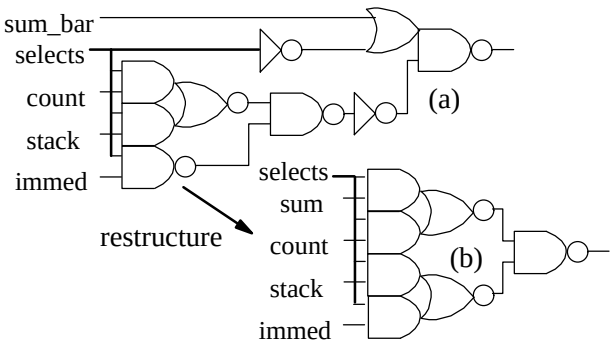

Figure 4. Mux implementations at output of adder. Initial implementation (a) assumed selects (from controls) were not critical, and final (b) with optimized control and sum paths.

POWER4 often straddles across several technology generations. Each technology migration induces device model changes (e.g. different p/n strength ratio) which full-custom designs are difficult to adjust to. With the semi-custom design approach, timing shift can be readily accommodated by the tuner, and with the flexible physical design, adjustments can be made in a timely fashion. We estimate that the semi-custom design at least reduces the total design time by 50%, even for designers unfamiliar with the place and route environment, who need to make an initial investment in learning the tools.

5. OVERALL IMPACT ON CHIP DESIGN

Application of this semi-custom methodology to custom macro design has shown that benefits come in two distinct waves. The first is an improvement in cycle time performance, primarily associated with the circuit tuning process. Generally designers apply it directly to their existing designs, looking for rapid turn-around, combined with some performance or area improvement. The second wave comes when a designer makes a more basic change in approach to design, concentrating more effort on circuit architecture, then using the rapid turnaround to quantify the performance of multiple designs. Each design approach can be tried in a real design context, optimally sized, including real parasitics and timing assertions. This leads to greater investment in optimizing the circuit architecture, rather than selecting one design early, and optimizing the sizing and physical design manually.

The difference this change in philosophy makes can be put in the context of the overall design closure shown in Fig 5. This shows the convergence of chip cycle time as a function of time for the latest S/390 processor, BlueFlame, [3] along with some of the typical design activities driving the improvement at each point, up through tape-out. The custom design phase extends up until the point where cycle time is ~1.25x the target, by which time full physical design is required for reliable extraction-based timing. With the conventional approach to custom design, the time required to implement physical design often requires

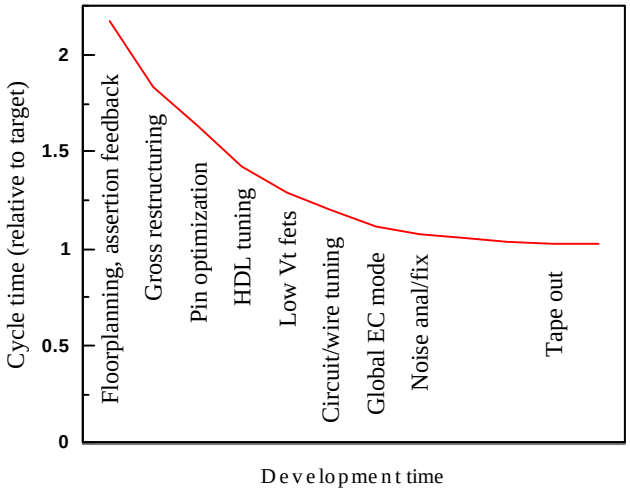

Figure 5. Chip timing closure progress as a function of time. The labels indicate the dominant activities used to improve timing at each stage.

commitment to a particular circuit architecture when the chip timing still exceedes the 1.8x-target range. As with the adder in section 4, this can lead to selecting a less than optimal circuit architecture. Being able to adapt quickly to changes associated with global timing convergence is a major advantage to semi-custom design. Experience suggests that these advantages are more valuable than any loss associated with the use of place and route in the physical design.

Net improvements in designer productivity, above and beyond achieving a superior design, are also of interest. Since this methodology was most heavily applied in the instruction unit of the BlueFlame processor, we have done some analysis there to try to understand its overall impact. This involved a code-based and manual survey of the design to gather specific statistics, followed by some interviews with designers to make work time estimates. Table 2 breaks down the circuits in this unit by their construction method. While the total semi-custom area was fairly small, the impact on the design work was significant, as it was applied to the most difficult functions. The survey indicated that most of the remaining full custom design was bit-slice in nature (working registers, muxes, and 3 register files), having relatively large areas implemented with a small number of cells. Table 3 lists the 23 semi-custom blocks by function type, giving the number of unique designs for each.

Table 2. Macro circuit area by construction method in BlueFlame instruction unit.

Circuit construction	**area %**
Custom	55.3
Semi-custom	15.2
RLM (synthesized)	29.5

Table 3. Number of semi-custom functions by type in BlueFlame instruction unit.

Function type	**Number of blocks**
Adder	3
Increment	4
Compare	8
Error check	2
Other	6

No truly valid quantification of the productivity improvements associated with semi-custom design could be made, due to a lack of both a detailed recording of design effort and any form of a control, since previous design points were different and the design teams and overall methodology have changed too. Instead, we made an effort to estimate the improvement, giving quantities that were derived from discussions with several designers. A proto-typical set of times are given in Table 4 for 7 stages of design, for the full custom approach and the equivalent design in semi-custom. Once a design has been through the complete semi-custom flow once, an iteration typically only requires a couple of days, including structural changes to the design. When one includes all stages of circuit and physical design and analysis, the semicustom approach requires roughly one half the time, yields as good or better results, and provides the ability to change the design quickly and reliably late in the design process.

Table 4. Estimated time (arbitrary units) by design phase for full- and semi-custom design flows.

Design steps	**Full-C**	**Semi-C**
Circuit arch, initial schematic	5	5
Floorplan, area estimate, wire util.	3	3
Parasitic estimation, circuit sizing	3	-
Post-initial timing circuit struct/sizing	6	-
Physical design – leaf cells	12	2
Automated tuning	-	2
Physical design – assembly	6	2
TOTAL	**35**	**14**
Time for each additional iteration	?	+2

6. ACKNOWLEDGMENTS

Our thanks to Chandu Visweswariah, Phil Strenski, and Ee Cho (circuit tuning), Joe Nocerra and Ching Zhou (cell generation), Keith Barkley (place and route), Brian Curran, Tom McPherson (timing convergence), and many designers for their patience, experiences, and suggestions.

7. REFERENCES

[1] Averill, R.M. et. al., Chip integration methodology for the IBM S/390 G5 and G6 custom microprocessors. IBM Journal of Research and Development, 43, 1999, 681-706.

[2] Check, M.A., Slegel, T.J., Custom S/390 G5 and G6 microprocessors. IBM Journal of Research and Development, 43, 1999, 671-680.

[3] Curran, Brian, et. al., A 1.1 GHz First 64b Generation Z900 Microprocessor, ISSCC Digest of Technical Papers, 238-239, Feb 2001.

[4] Anderson, Carl J., et. al., Physical Design of a Fourth-Generation POWER GHz Microprocessor, ISSCC Digest of Technical Papers, 232-233, Feb 2001.

[5] C. Visweswariah and A. R. Conn, Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation, IEEE International Conference on Computer-Aided Design, pages 244-251, November 1999.

[6] A. R. Conn, I. M. Elfadel, W. W. Molzen, Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, Gradient-based optimization of custom circuits using a static-timing formulation, Proc. Design Automation Conference, pages 452-459, June 1999.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A). Springer Verlag, 1992.

[8] Huey Ling, “High-Speed Binary Adder,” IBM J. Res. Develop., Vol. 25, No. 3, May 1981, pp. 156-166.