"Timing Closure by Design," A High Frequency Microprocessor Design Methodology

S. Posluszny, N. Aoki, D. Boerstler, P. Coulman[1], S. Dhong, B. Flachs[2], P. Hofstee, N. Kojima, O. Kwon, K. Lee[3], D. Meltzer[4], K. Nowka, J. Park[5], J. Peter, J. Silberman[4], O. Takahashi, P. Villarrubia[1]

IBM Austin Research Lab, Austin, TX
(512) 838-6508
poslus@us.ibm.com

ABSTRACT

This paper presents a design methodology emphasizing early and quick timing closure for high frequency microprocessor designs. This methodology was used to design a Gigahertz class PowerPC microprocessor with 19 million transistors. Characteristics of "Timing Closure by Design" are 1) logic partitioned on timing boundaries, 2) predictable control structures (PLAs), 3) static interfaces for dynamic circuits, 4) low skew clock distribution, 5) deterministic method of macro placement, 6) simplified timing analysis, and 7) refinement method of chip integration with early timing analysis.

Keywords

Timing closure, microprocessor, methodology, chip integration, CAD, timing analysis, PLA, dynamic circuits.

1. INTRODUCTION

Timing closure for large microprocessor designs is becoming more and more difficult as 1) chip complexity increases, 2) cross-chip wire delays become more significant, 3) dynamic circuits become more prevalent, and 4) cycle times shorten. Just as "Correct by Construction" techniques reduce introduction of layout errors in chip designs, "Timing Closure by Design" techniques reduce introduction of timing problems. These timing problems can not be typically found or fixed until late in the design process making it difficult to meet required frequency targets. The "Timing Closure by Design" methodology has the goals of 1) achieving the highest possible processor frequency, and 2) reducing the design time to achieve that desired frequency. The main themes of this methodology are early timing planning with an eye towards the physical implementation, and using components and design techniques with predictable timing characteristics. This methodology was used to design an experimental 19 million transistor PowerPC microprocessor that was designed to operate at 1.0 Gigahertz (1.62V, 85^{o}C) [1]. The chip was manufactured in IBM's 0.12 micron Leff, 3.5nm Tox, 6 layer copper interconnect process technology. As a testament to the efficiency of the overall design methodology and timing closure process, the chip was designed in only 18 months with approximately 15 designers. A previous 1.0 Gigahertz integer processor [2,3,7] was built using many of the same concepts described in this paper.

Many other approaches to the timing closure problem concentrate on enhancing and integrating DA tools, such as combining synthesis, placement and timing analysis [6]. Our approach is much more deterministic using predictable components and techniques, and relies less on optimization tools. This approach forces the microarchitect and logic designer to consider implementation timing from the start and avoids back-end tools that can mess up the designed timing relationships. The key points of "Timing Closure by Design" are as follows:

- Early delay planning and partitioning of the high level design on timing boundaries. Stable timing and drive requirements of component macros.
- Predictable implementation of dataflow and control circuits, for example dynamic PLAs.
- Quasi-static interfaces between dynamic macros by stretching pulses into the next cycle.

[1]IBM Server Division, Austin, TX
[2]Motorola, Austin, TX
[3]Sun Microsystems, CA
[4]IBM Watson Research Lab, Yorktown, NY
[5]Samsung, Korea

DAC 2000, Los Angeles, California

- Clock distribution with low skew and jitter.
- Macros placed in a deterministic and repeatable manner.
- Simplified static timing analysis techniques used early and often.
- Chip integration methodology, which refines the design and provides quick turnaround from logic to full chip timing results.

The rest of the paper will describe in detail the key points listed above. The combination of specific design concepts and techniques with tailored CAD tool flows is what makes our "Timing Closure by Design" methodology unique.

PLANNING AND LOGIC PARTITIONING

A complete and detailed timing plan at the beginning of the project is critical to achieve quick timing closure at the end of the project. The timing plan should have a delay budget for dataflow and control portions of the design and include buffering and wire delays to move signals across the chip. To have an efficient design, it is important to balance all the execution and control paths to have approximately the same delay. It is also important to recognize the contribution of clock skew and jitter and recognize how important it is to reduce their effect on the timing closure process. Figure 1 illustrates the timing budgets used on our 1.0 Gigahertz microprocessor [1]. Notice that there is only one major component in each of the control and dataflow paths, a dynamic PLA or functional block. This major component needs to be implemented as a single physical macro and can be designed independently from the other macros on the chip.

The partitioning of the high level logic must be on cycle boundaries. Even the control should be partitioned on cycle boundaries allowing the control to merge with the dataflow only at the end of the cycle. The microarchitecture designer needs to know up front what functions can be implemented in one cycle and the general placement of macros on the chip, accounting for wire delays between macros. Partitioning on timing boundaries achieves 1) early recognition of over or under specification of function for a particular cycle, 2) stable arrival times for macro inputs, 3) stable drive strength and delay requirements for macro outputs, and 4) eliminates the delay apportionment problem of timing paths that traverse multiple macros. This type of partitioning also insulates the macro designer from needing to adjust their design due to changes in other macros. Typically a single circuit designer owns a complete macro and therefore the major delay element of the complete cycle. The macro scope, ownership, and independence from other requirements significantly improve the timing closure process.

CONTROL

200ps	8-way MUX-Latch
470ps	Dynamic PLA, Comparator
140ps	Single Stage Logic, Repower, Wire Delay
140ps	Latch SELECT Setup Time
+50ps	Clock Skew and Jitter
1000ps	

DATAFLOW

200ps	8-way MUX-Latch
610ps	Functional Block
140ps	Repower, Wire Delay
0ps	Latch DATA Setup Time
+50ps	Clock Skew and Jitter
1000ps	

Figure 1 Timing Budget

2. PREDICTABLE CONTROL AND DATAFLOW CIRCUITS

Using structured circuit and layout approaches can eliminate some of the timing uncertainty in the macro design process. For example, we used dynamic PLAs and comparator structures for all of our control logic [3]. The dynamic PLA provides 1) high frequency operation, 2) quick logic personalization, 3) predictable area and delay, and 4) early recognition of excess logic for one cycle. As compared to a standard cell approach, no heuristic logic synthesis or auto placement is required with PLAs or comparators. Those tools may require many runs and adjustments to input parameters to achieve timing requirements, as well as inject uncertainty late in the design process when last minute logic changes are necessary. The PLAs have exclusive latch drivers (one fanout), which are placed adjacent to the PLAs, minimizing input wire delays and consequently input skews. Due to the high performance of the PLAs (300-470ps delay), a single level static or dynamic gate can be connected to the PLA outputs, increasing functionality without significantly adding delay uncertainty. Figure 2 illustrates a template for interfacing different configurations of control PLAs, compare macros and individual gates with the dataflow path.

Dataflow macros were built using delayed-reset domino circuits [5]. The delayed reset helps to spread the clock load throughout the cycle and reduces VDD degradation. Most macros were also designed to have an equal number of logic stages for each path through the macro. This helped to ensure proper timing of pulses within the macro. Given the short cycle requirements, macros were limited to 2-5 dynamic logic levels. This up front design limit of logic levels actually helped timing closure by identifying circuit implementation problems early in the process.

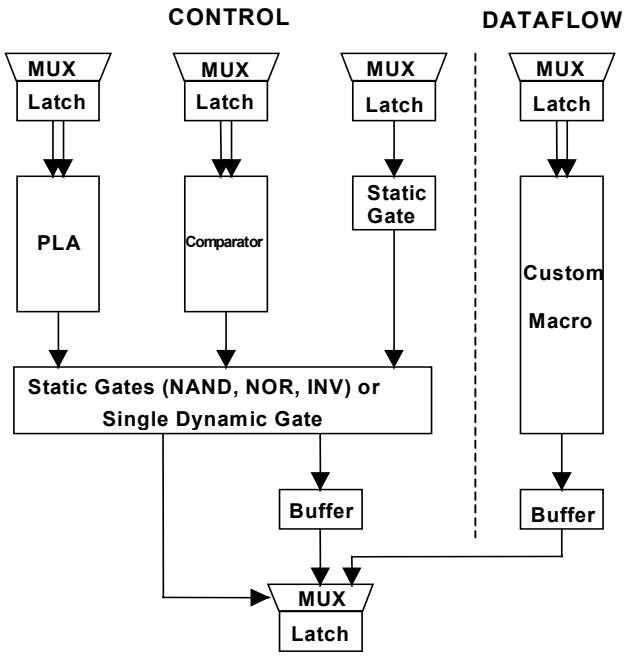

Figure 2 Control Template

Static inverters and 2-way NAND gates were automatically generated. The inputs to the generation process were NFET and PFET sizes and the input bus width. A naming convention was used to identify the gate characteristics. For example, INV64_30_78, referred to a 64-bit inverter with 30 micron NFETs and 78 micron PFETs. Being able to generate any size inverter or NAND was helpful to tune the critical paths. Having the device sizes built into the block name was also helpful in analyzing the VHDL and timing reports.

3. QUASI-STATIC INTERFACES

Dynamic circuits are typically faster than static circuits, but pose certain drawbacks: 1) they are susceptible to noise glitches, 2) there are strict pulse overlap requirements on inputs, 3) both true and complement signals are required to be generated throughout the logic tree, and 4) reset signals need to be provided at proper times. It can be very difficult to guarantee pulse overlaps once the signals are outside the confines of a single macro and are routed with an automated tool. We therefore contain dynamic pulses within a macro with the exception of pulse outputs of a dynamic latch feeding directly into a macro. The trailing-edge of a macro output pulse is triggered by the clock. This stretches the pulse into the following cycle. Pulse stretching essentially makes the output signal quasi-static and reduces the possible HOLD time problems that can be seen at the receiving macro or latch, which ensures operation independent of frequency. Figure 3 illustrates the timing relationships of the global clock, macro inputs, and stretched macro outputs. Noise on global signals is addressed by placing a redrive inverter near the receiving macro to filter out noise glitches. We have reduced the amount of dual-rail signaling within a dynamic macro by requiring only a single-rail pulse-stretched output to be generated from the macro. Using only a single-rail output also reduces the number of global wires that need to be routed across the chip. The cost of only distributing single-rail signals is the addition of a True/Complement generator within the latch.

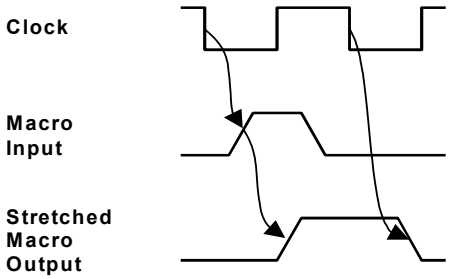

Figure 3 Pulse Stretching

4. LOW SKEW CLOCK DISTRIBUTION

By generating a clock with a very low skew and jitter (less than 20ps), we can discount the clock delay when performing our timing analysis. This isolates the clock generation and distribution problem from the timing closure problem of the rest of the chip. We used a grid-tree approach (similar to [4]) to distribute the clock. Specifically we used one central buffer driving two partition buffers, which then drove 16 sector buffers. The sector buffers drove a chip wide grid (approximately 0.5 micron pitch). A second and third clock grid were superimposed on the left and right side of the chip to provide a 200ps delayed clock signal to the memory management units. Three sector buffers drive each of these two delayed grids. The wire widths in the H-trees were adjusted based on sector loading to achieve the minimum skew. These H-trees were also shielded with power and ground lines to control capacitance and inductance effects. This provided predictable clock delay regardless of surrounding macros and global wires. “Twig” wiring was used to connect from the grid to the macro clock pins. These twig wires were automatically inserted as direct and shortest possible connections between the grid and macro pins. The clock tree, grid and twig wires were treated as blockages for the power grid generator and global routing. Figure 4 illustrates the clock distribution, showing the H-tree, grid and twig wires.

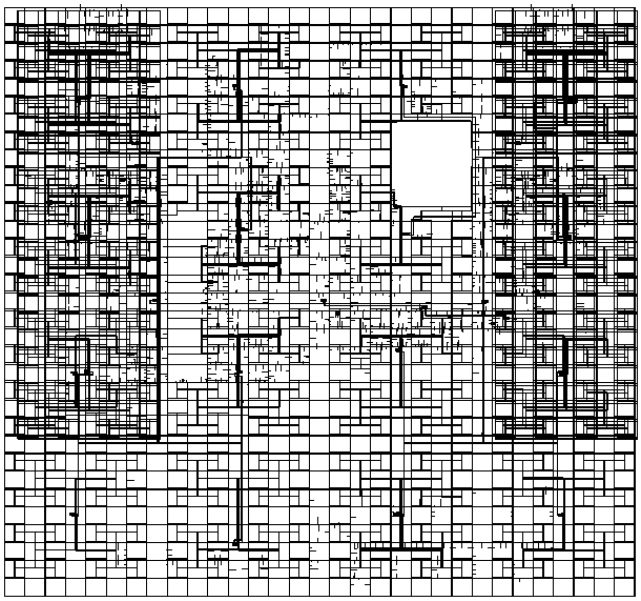

Figure 4 Clock Distribution

A large part of the clock distribution is included as part of the large custom macros. Each macro's clock distribution was designed to match the delay of the 64-bit latch clock distribution with three levels of inversion. This part of the clock distribution is accurately simulated with the rest of the macro. Another method to reduce the effect of clock skew is to distribute and use only one global clock. Using multiple clock phases in a design only exacerbates the clock skew problem and excess margin needs to be built into the design to ensure enough clock overlap time. Timing closure is improved when the clock is well designed (low skew and jitter), the affect of clock delay and skew is minimized (single global clock), and the clock distribution is largely automated.

5. DETERMINISTIC PLACEMENT

Since wire delays are more and more prominent in timing paths, correct macro placement is more critical in achieving chip frequency targets. Automated placement programs can simplify the task of placing macros on the chip and optimize area utilization and wire lengths, but they introduce uncertainty in wire delays between macros. Timing closure can be better achieved by manually placing macros relative to each other, locking in inter-macro wire delays. A feature in our internal floorplanning tool (ChipBench), allowed us to define a text file listing horizontal and vertical groups of macros. This placement file was used to place macros relative to each other. When macro images moved from early estimates to actual layouts, the placement file was reloaded and macros were automatically shifted based on the new macro sizes. Again, by using relatively large custom macros and dynamic PLAs instead of standard cells, we reduced the number of placeable objects on the chip, simplifying the task of manual placement. Figure 5 illustrates the chip floorplan, showing actual macro placements for the prototype chip.

There was only one level of design hierarchy exposed to the chip integration process, namely the chip level instantiating leaf level custom macros. This contained the problem of proper pin placement to the custom macros. We avoided the task of unit level pin assignments, and consequently, refinements due to timing and routing problems.

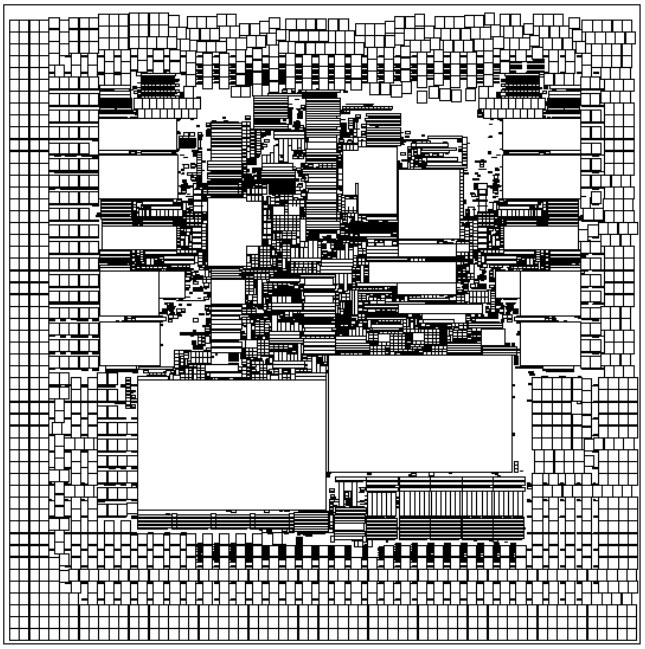

Figure 5 Chip Floorplan

6. TIMING ANALYSIS

Fast and efficient full chip timing analysis aids timing closure by speeding the refinement process and allows more design iterations to be evaluated. Simplified timing rules were hand written for each macro [3] using the Delay Calculator Language (DCL). These rules grouped input buses together and used the worst case delay for all the bits of the bus. This type of abstraction was possible because of the partitioning on timing boundaries discussed earlier. The abstraction drastically reduces the number of timing segments through a macro and the number of timing tests for the macro inputs. Figure 6 illustrates two rule abstractions, one with all paths specified and the other with just the worst-case delay specified. These simplified rules allowed us to analyze the full chip in less than ten minutes, from loading the netlist and parasitics to writing out a timing report. This reduction in analysis time also enabled us to interactively and incrementally run timing analysis while floorplanning the chip. The timing impacts of modifying macro placement could immediately be checked and verified.

A typical timing path as reported by the static timing tool (Einstimer), has a driving latch, dynamic macro, two redrive inverters, and the receiving latch. Since the majority of the chip path delay resides at the macro level, the macro delay will be the main factor determining the chip's final cycle time. Macro timing was analyzed using detailed simulation. Even before layout was

started, wire models were added to the transistor schematics to accurately calculate the macro delay. At the chip level, the main lever we have to affect timing closure is optimizing wire delays. The wire delays were controlled by modifying placement, manually adding redrive inverters, and defining a preference of wire levels, widths and spacing to the router. We modeled global wire delays in three ways: 1) Elmore delays from Steiner tree estimates, 2) 3-D estimation from initial chip tile routing, and 3) 3-D extracted final wires. The Steiner tree estimates were done during floorplanning, providing quick turnaround, but poor accuracy. The wire delay accuracy from the initial tile routes and estimated congestion proved to be very close to the final 3-D extracted delays, yet required much less CPU time to compute. Therefore we used the wire delays from the initial tile routes to hone in on the final timing.

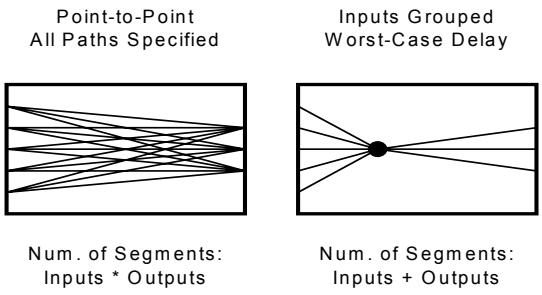

Figure 6 Timing Rule Abstraction Comparison

7. CHIP INTEGRATION FLOW

The chip integration process has been broken down into four stages. Each successive stage requires more detailed and complete information and takes more time to iterate through. Consequently the majority of the design schedule is within stage one and only at the very end, about a week before sending to manufacturing, does chip integration move to stage four. Below are the attributes of each stage. Gross timing problems are found in stage one and the majority of the detail timing closure happens in stage two. Depending on the number of modifications, stage one takes on the order of hours to cycle through, stage two takes about a day, stage three takes about two days and stage four only takes a few hours.

7.1 Stage One - Early Placement, No Routing

- Structural VHDL entry, instantiate latches and component macros
- Code delay rules (DCL) using predicted delays
- Code physical rules with planned pin locations and macro size
- Manually place macros using placement file
- Run wire delay estimation code only using Steiner trees
- Run full chip timing analysis

7.2 Stage Two - Estimated Routing

- Refine structural VHDL
- Refine delay rules with simulated results
- Generate PLA schematics and layouts from VHDL and Espresso files
- Generate physical rules (size, pin locations, blockages) from actual layout
- Refine placement using placement file
- Run clock distribution generation (grid-tree)
- Run power grid generation
- Run initial “tile” routing and use results to estimate wire delays
- Run full chip timing analysis

7.3 Stage Three - Detailed Routing, Full Wire Extraction

- Refine structural VHDL
- Generate any new or modified PLAs
- Generate physical rules for any new or modified macros
- Refine placement using placement file
- Run clock distribution generation and twig routing
- Run power grid generation
- Specify any wiring preferences (extra wide or thick wires)
- Run complete, full chip routing
- Run 3-D extraction for accurate wire delays
- Run full chip timing analysis

7.4 Stage Four - Incremental Updates

- Make only wiring changes to VHDL
- Extract the netlist changes, Engineering Change Order (ECO)
- Remove any deleted wires and reroute only new or modified wires
- Run 3-D extraction for accurate wire delays
- Run full chip timing analysis

Having the ability to do quick timing analysis of the full chip at the very earliest stages of design with incomplete data, aids the timing closure process. The refinement nature of this process helps to reduce the number of timing surprises found late in the project.

8. CONCLUSIONS

We have shown how efficient timing closure can be achieved for a high frequency microprocessor design by using a structured, timing oriented methodology. The "Timing Closure by Design" methodology requires 1) early planning and timing based logic partitioning, 2) delay predictable components such as PLAs, 3) static global interfaces for dynamic circuits, 4) a low skew, single global clock distribution, 5) deterministic macro placement, 6) simplified timing analysis, and 7) a quick and easy design refinement process. This "Timing Closure by Design" methodology was instrumental in designing a 19 million transistor microprocessor and achieving its 1.0 Gigahertz frequency target with a small team on a tight schedule.

9. ACKNOWLEDGMENTS

We would like to thank the complete prototype design team for their tireless effort and contributions to this design methodology. We would also like to thank the IBM internal CAD groups for developing excellent tools and helping to incorporate those tools into our methodology. Special thanks to P. T. Patel for his insights into the chip integration process and help with our routing program.

10. REFERENCES

[1] Hofstee, P., et al., "A 1GHz Single-Issue 64b PowerPC Processor", ISSCC Digest of Technical Papers, p. 92, Feb. 2000.

[2] Silberman, J., et al., "A 1.0GHz Single-Issue 64b PowerPC Integer Processor", ISSCC Digest of Technical Papers, p. 230, Feb. 1998.

[3] Posluszny, S., et al., "Design Methodology for a 1.0 Ghz Microprocessor," ICCD98, p.17.

[4] Northrop, G., et al., "600 MHz G5 S/390 Microprocessor", ISSCC Digest of Technical Papers, p. 88, Feb. 1999.

[5] Nowka, K and T. Galambos, "Circuit Design Techniques for a Gigagertz Integer Microprocessor", ICCD98, p.11.

[6] Hojat, S. And P. Villarrubia, "An Integrated Placement and Synthesis Approach for Timing Closure of PowerPC Microprocessors," ICCD97, p 206-210, 1997.

[7] Hofstee, P., et al., "Designing for a Gigahertz," IEEE Micro, May-June 1998.