Implementing Tile-based Chip Multiprocessors with GALS Clocking Styles

Zhiyi Yu and Bevan Baas

ECE Department, University of California, Davis

Abstract— **This paper investigates implementation techniques for tile-based chip multiprocessors with Globally Asynchronous Locally Synchronous (GALS) clocking styles. These architectures can simplify the physical design flow since they allow focusing on a single processor when designing an entire chip. However, they also introduce challenges to maintain system robustness and scalability. We propose a physical design flow for these architectures, investigate timing issues for robust implementations, and propose methods to take full advantage of their potential scalability. As a design example, we present data from a recently implemented single-chip 6×6 tile-based GALS processing array.**

I. Introduction

It has become increasingly difficult to continue scaling system performance by increasing clock frequencies using modern deep submicron fabrication technologies. However, the recent appearance of chips with multiple processors [1] shows the promise of combining greater integration with tile-based chip multiprocessor architectures due to their high performance, high scalability, and potentially high energy efficiency. Some recent tile-based chip multiprocessors include RAW [2], TRIPS [3], Smart Memories [4], and AsAP [5].

The globally synchronous clocking style is encountering challenges although it is still widely used in most IC systems. The clocking system is increasingly difficult to design with larger chip sizes, higher clock rates, larger relative wire delays, and larger parameter variations [6]. Additionally, high speed global clocks consume a significant portion of many power budgets.

Tile-based synchronous chip multiprocessor architectures, as shown in Fig. 1 (a), have difficulty in taking full advantage of scalability benefits since the clock tree must be redesigned when the number of processors in the chip changes—which can result in a large effort in high performance systems. Also, clock skew in globally synchronous chips is expected to increase as the number of processors increases.

The Globally Asynchronous Locally Synchronous (GALS) clocking style separates processing blocks such that each part is clocked by an independent clock domain. This approach is a promising strategy to address many clock design challenges and can also reduce power consumption by adaptive clock frequency scaling [7]. Furthermore, when using the GALS clocking style with a tile-based chip multiprocessor as shown in Fig. 1 (b), scalability is dramatically increased and the physical design flow is greatly simplified, since a single processor can be designed and the entire chip can be generated easily by duplicating a single processor design.

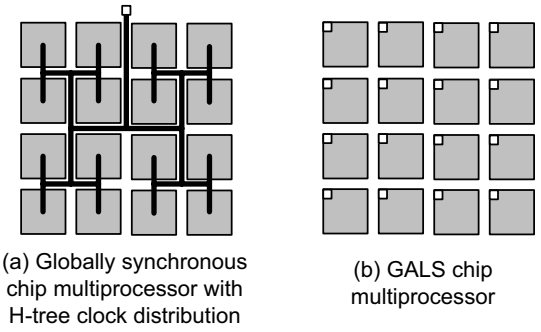

Fig. 1. Block diagrams of (a) Synchronous and (b) GALS tile-based chip multiprocessors; the small boxes in the right figure are the local oscillators for each processor

While tile-based chip multiprocessors with GALS clocking styles provide great benefits, they impose some design challenges regarding the robust handling of timing issues and limits in taking full advantage of system scalability.

This paper investigates key issues with the implementation of tile-based GALS chip multiprocessors. Most of the techniques investigated in the paper have been implemented and verified in a recent fabrication [5] that is the first tile-based GALS chip multiprocessor to the best of our knowledge.

A. Physical design flow for a tile-based GALS chip multiprocessor

Since each processor including its clock tree can be exactly the same in a GALS tile-based chip multiprocessor, it provides near-perfect scalability and greatly simplifies the physical design flow. One processor design can be easily duplicated to generate an array processor.

Fig. 2 shows the hierarchical physical design flow for a tile-based chip multiprocessor with a GALS clocking style. We assume a local oscillator is used to provide the clock for each processor, and it is designed separately for more robust operation. The right column of Fig. 2 shows the physical design flow for a single processor, and a similar flow also applies for the oscillator and the entire chip.

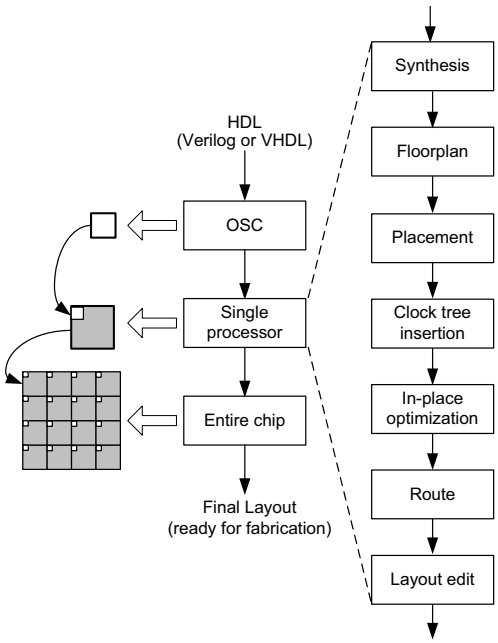

Fig. 2. Hierarchical physical design flow of a tile-based GALS chip multiprocessor, with details specific to a synthesized standard cell design. A single processor tile can be replicated across the chip.

II. Timing issues of GALS chip multiprocessors

Although the key features of GALS chip multiprocessors simplify the physical design flow, they introduce challenges in handling the timing of signals, including signals within a single processor, signals between processors, and signals between chips.

Signals that cross GALS clock domains clearly require special care. We can classify methods for guaranteeing safe domain crossing into one of two categories:

- *Single transaction handshaking* where each data word is acknowledged before a subsequent word can be transferred, and a corresponding latency exists for each data transfer.
- *Coarse grain flow control* where data words are transmitted without individual acknowledgments. This technique generally requires larger buffers but can normally sustain higher throughputs, especially for links with latency that is significant with respect to the clock cycle time.

In this work, we consider only the second approach due to its higher throughput and because the larger buffer does not present a significant penalty when compared to the area of a coarse block such as a processor.

Figure 3 contains an overview of important timing issues in GALS chip multiprocessors using coarse grain flow control. All signals in such interfaces can be classified into three categories for an example with processor A sending data to processor B:

- A → B clock: the clock synchronizes the *source synchronous* signals traveling from A to B. This scheme requires an extra clock tree to be inserted at processor B which brings additional timing concerns for inter-processor communication.
- A → B signals: includes the data to be sent, and can also include other signals such as a "valid" signal.
- B → A signals: processor B can send information back to the source; for example, a flow control signal such as "ready" or "hold" falls into this category.

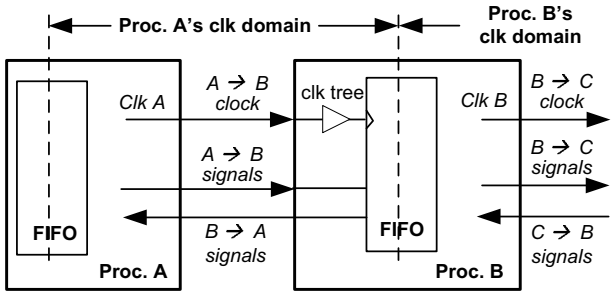

Fig. 3. An overview of timing issues in GALS chip multiprocessors; each clock domain covers multiple processors and each processor contains multiple clock domains

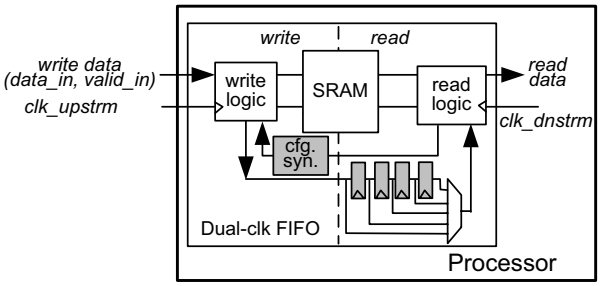

Fig. 4. Configurable logic at the asynchronous boundary in a FIFO; two clock domains exist within a single processor

Besides this inter-processor communication, there is also the issue of transferring data across clock domains. A dual clock FIFO [8] can reliably handle this asynchronous interface, and is briefly described in the following section.

A. Multiple clock domains within one processor

Figure 4 is a high-level diagram of one dual clock FIFO [8] with a processor. The FIFO read clock (*clk_dnstrm*) and FIFO write clock (*clk_upstrm*) are totally unrelated and synchronization logic is necessary at the clock domain boundary to handle rate matching and the asynchronous metastability problem. Other methods are possible, but a synchronizer using multiple flip-flops is most widely used. Although it can not drive the *mean time-to-failure* (MTTF) to infinity, it can make it arbitrarily high [9]. The number of synchronization registers used is a tradeoff between the synchronizer's robustness and the system performance overhead due to the synchronizer's latency. The latency to communicate across the asynchronous boundary is approximately 4 clock cycles in this example, which is made up of the write logic latency (1 cycle), synchronization latency (2 cycles if two registers

TABLE I

GALS SYSTEM PERFORMANCE REDUCTION COMPARED TO A SYNCHRONOUS SYSTEM USING DIFFERENT NUMBERS OF SYNCHRONIZATION REGISTERS FOR 4 APPLICATIONS

	1 reg.	2 reg.	3 reg.	4 reg.
8 × 8 DCT	0.7%	1.4%	2.1%	2.8%
Mergesort	0%	0%	0%	0%
JPEG encoder	0.15%	0.3%	0.45%	0.6%
802.11a/g tx	0.65%	1.3%	1.95%	2.6%
Average	0.37%	0.75%	1.1%	1.5%

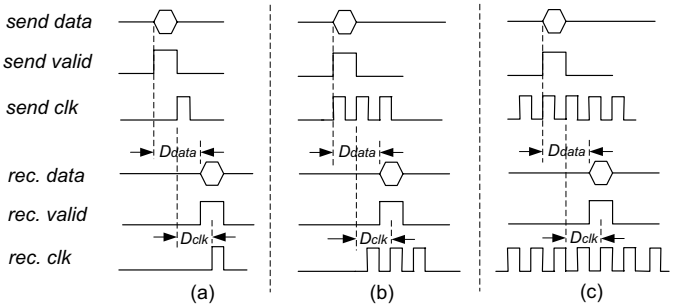

Fig. 5. Three methods for inter-processor communication: (a) sends clock only when there is valid data; (b) sends clock one cycle earlier and one cycle later than the valid data; (c) always sends clock

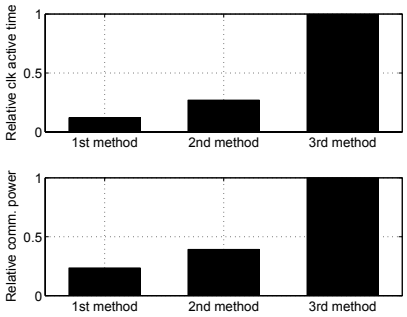

Fig. 6. Relative clock active time and communication power consumption for the three inter-processor communication methods described in Fig. 5 for a 2-D 8×8 DCT application

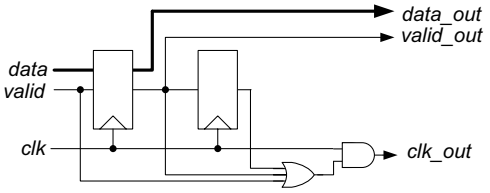

Fig. 7. Circuit for the Fig. 5 (b) inter-processor communication method

are used), and the read logic latency (1 cycle). Since the coarse grain source synchronization method is used instead of the single transaction handshaking method, the throughput of the asynchronous boundary communication can achieve 1 data word per cycle when the FIFO is not full or empty.

In a relevant example [9], it was estimated that the MTTF when using one register is measured in years and it will increase to millions of years for two registers, so a small number of synchronization registers is sufficient.

Table I shows that the system performance (throughput) overhead is always quite small for even the case of four synchronization registers. The applications are simulated on the AsAP GALS chip multiprocessor with 32-word FIFOs [5].

B. Inter-processor timing issues

Figure 5 shows three strategies to send signals from one processor to another. Subfigure 5 (a) is an aggressive method where the clock is sent *only* when there is valid data (here we assume the clock is sent out one cycle later than the data although they can also be in the same cycle). This method has high energy efficiency, but imposes a strict timing requirement between the delay of data (D_{data}), the delay of clock (D_{clk}) and the clock period (T), as shown in Eq. 1.

$$t_{hold} < D_{data} - D_{clk} < T - t_{setup} - t_{clk\_to\_Q} \quad (1)$$

Subfigure 5 (c) is a conservative method where the clock is always active. With this method, the delay of data does not have to satisfy Eq. 1, as long as *data* and *valid* reach processor B within the same clock period, which results in a system similar to wave pipelined systems [10] with a slightly modified set of timing equations and of course additional constraints on minimum path times. We propose a compromise method as shown in Subfigure 5 (b), where the clock starts one cycle before the data and ends one cycle later than the data. This scheme has high energy efficiency similar to the aggressive method since it keeps a relatively low clock active time. Figure 6 compares the communication power of these three methods for a 2-dimensional 8×8 DCT application using four processors, assuming the power is zero when there is no active clock, and the power is reduced by 50% when there is no inter-processor data transferring but the clock is running. The second scheme has a much more relaxed timing requirement compared to the first method, as shown in Eq. 2. Figure 7 shows a circuit to realize this second scheme.

$$-T < D_{data} - D_{clk} < 2T \quad (2)$$

Figure 8 shows a generic implementation for inter-processor communication where processor A sends signals (including *data, valid,* and *clock*) to processor B. Along the path of *data* there are delays in processor A (D_{data_A}), at the inter-processor wiring (D_{data_w}), and in processor B (D_{data_B}). The path of *clk* has a similar makeup. Not considering the clock tree buffer, the data path and clock path have roughly the same logic and similar delays. To compensate for the clock tree delay and to meet the timing requirements, configurable *DLY* logic is inserted in both processor A and B as shown in Fig. 8. Equation 1 can then be simplified as follows:

$$t_{hold} < D_{insert} - D_{clk\_tree} < T - t_{setup} - t_{clk\_to\_Q} \quad (3)$$

Here D_{insert} and D_{clk_tree} are the delays of inserted logic and clock tree respectively. Normally the t_{hold}, t_{setup}, and $t_{clk\_to\_Q}$

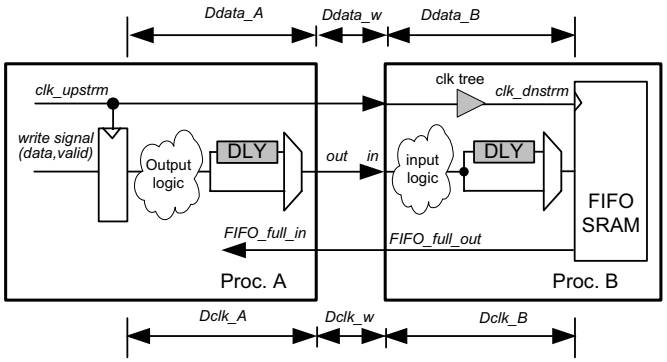

Fig. 8. Configurable logic at the inter-processor boundary where one clock domain covers two processors; GALS style brings an additional clock tree at the consumer processor

can be neglected since they are generally small compared to the clock period. Equation 4 lists three delay possibilities for the inserted delay logic, and the optimal delay for each inserted gate is shown in Eq. 5 and Eq. 6.

$$D_{insert} = 2D_{MUX} + \{0, D_{DLY}, 2D_{DLY}\} \quad (4)$$

$$D_{MUX} = D_{clk\_tree}/2 \quad (5)$$

$$D_{DLY} = T/2 \quad (6)$$

As a typical example, if the clock tree delay is 6 Fanout-of-4 (FO4) delays and the clock period is 20 FO4; then the optimal delays for MUX and DLY gates are: $D_{MUX} = 3$ FO4, and $D_{DLY} = 10$ FO4.

The third type of signal consists of signals that flow in a direction opposite to the clock signal. As illustrated in Fig. 8, a common example of such a signal is the *FIFO_full* signal. Such signals do not need to match delays with others so they are relatively easy to handle, but they can not be too slow so they arrive in the correct clock period. The timing constraints discussed below handle this requirement.

Previously discussed circuits match the delay according to logic delays, but the real circuit delay is also highly dependent on wiring and gate loads. Specific input delay and output delay constraints clearly quantify circuit timing requirements. The value of input delays and output delays should follow Eqs. 7 and 8 for the architecture shown in Fig. 8.

$$output\_delay = T - D_{data\_A} \quad (7)$$

$$input\_delay = T - D_{data\_B} \quad (8)$$

If $T = 20$ FO4, $D_{DLY} = 10$ FO4, $D_{MUX} = 3$ FO4, and output logic at processor A is 2 FO4, then $D_{data\_A} = 15$ FO4 and *output_delay* should be 5 FO4. *Input_delay* can be calculated similarly. Table II lists delays for one example case study.

C. Inter-chip timing issues

Similarly to the case of inter-processor communication, *inter-chip* communication presents timing challenges as illustrated in Fig. 9. Besides the delay at the producer processor

TABLE II
TYPICAL TIMING CONSTRAINT VALUES FOR PROCESSOR INPUT AND OUTPUT DELAYS

Constraint	Signals	Reference clk	Value
input delay	*data_in*	*clk_dnstrm*	5 FO4
input delay	*valid_in*	*clk_dnstrm*	5 FO4
input delay	*FIFO_full_in*	*clk_upstrm*	10 FO4
output delay	*data_out*	*clk_upstrm*	5 FO4
output delay	*valid_out*	*clk_upstrm*	5 FO4
output delay	*FIFO_full_out*	*clk_dnstrm*	10 FO4

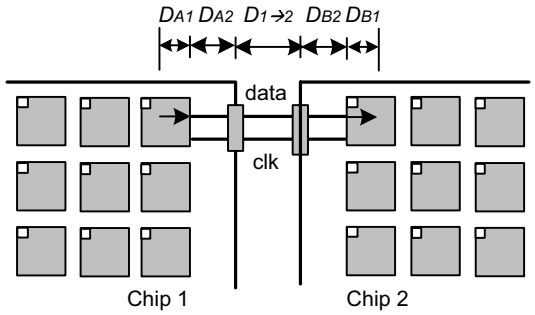

Fig. 9. Inter-chip communication

(D_{A1}) and consumer processor (D_{B1}), there is also delay at the chip A boundary (D_{A2}), chip B boundary (D_{B2}), and the inter-chip delays including pads, package, and printed circuit board delays ($D_{1\rightarrow 2}$). The three communication schemes shown in Fig. 5 also apply to inter-chip communication and the configurable delay logic embedded into each processor shown in Fig. 8 is still valuable to adjust the data delay for inter-chip communication. Due to the more complex environment with inter-chip communication, Fig. 5 (b) and (c) methods are preferred.

III. Scalability issues of GALS chip multiprocessors

Tile-based architectures and GALS clocking styles improve the scalability of systems and allow adding more processors easily into the chip. But some additional issues must still be considered to take full advantage of its potential scalability. The key idea is to try to avoid or isolate all global signals if possible, so that multiple processors can be directly tiled without further changes.

A. Clocking and buffering of global signals

The GALS style avoids making the most important signal a global one: the clock. Signals discussed in previous sections are all *local* signals which run within one processor or travel at the boundary of two processors, so they can be controlled by a full speed clock. But it is likely that there are some unavoidable global signals such as configuration and test signals. These global signals can be pipelined into multiple segments [11] and still run at full speed; or can use totally asynchronous communication to avoid clock constraints [12]—both of these methods significantly increase

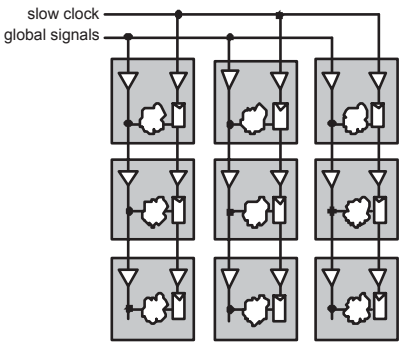

Fig. 10. Global signals controlled by a low-speed clock are buffered with inserted buffers and route through processors with internal wiring

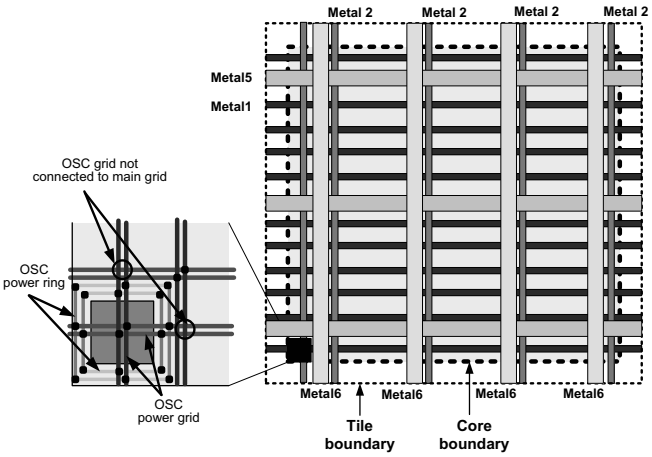

Fig. 11. An example power distribution scheme with power wires reaching beyond the processor core to allow it to directly abut other processors

design difficulty. We note that many *necessarily global* signals are related to functions that are either seldom used in normal operation, or are typically used at powerup time. Therefore, making them run at slow speed in many cases does not strongly affect system performance. A dedicated low-speed clock can then be used to control these less-critical global signals. These signals can be fed through each processor with internal wiring and buffering, to increase their driving strength and to enable them to be directly connected to an adjacent processor without any intermediary circuits at all. Figure 10 illustrates this scheme.

B. Power distribution

Power distribution can also be viewed as a global signal and deserves special consideration. In order to enable processors to directly abut each other without any further modifications, a complete power grid for each single processor should be designed. The width of metal power wires must be carefully considered to meet the voltage supply requirement, according to Eq. 9, where $V_2 - V_{orig}$ is the allowable voltage drop, I is the estimated current for the entire chip (not a single processor), L and w are the length and width of the metal wire, and ρ is

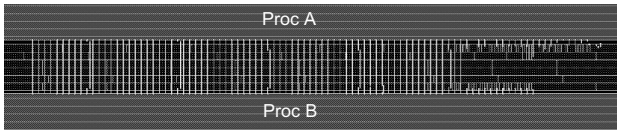

Fig. 12. Pins connections between two vertical processors; nearly directly abutting each other enables very short wires

the metal's resistivity per unit length and width.

$$I \times \frac{\rho L}{w} = V_2 - V_{orig} \tag{9}$$

Figure 11 shows a complete power distribution example for a single processor using 6 metal layers, which is common in $0.18\,\mu\mathrm{m}$ technology. Metal 5 and 6 are used for global power distribution with wide wires, and Metal 1 and 2 are used to distribute power to each gate with narrower wires. VIAs are placed between Metal 6 and 5, Metal 5 and 2, and Metal 2 and 1. Power grids reach out of the core to the tile boundary and enable processors to directly abut others at the chip level.

A related issue with processor power distribution is the power plan for the local oscillator, if one exists. To get clean clock power, the local oscillator should be placed and routed separately and then inserted into the processor as a hard macro. The left part of Fig. 11 shows an example implementation for the oscillator power grid. The oscillator should be placed away from the noisiest processor circuits to reduce clock jitter—this is likely at the corner of the processor. A placement blockage (block halo) should be added around the oscillator which blocks any logic from being placed at this location, to simplify the routing of oscillator signals and also to reduce effects from other logic. Finally, the oscillator in this example has a separated power ring and power grid which are not connected to the main processor power grid to get a clean power supply.

C. Position of IO pins

The position of IO pins for each processor is also important for scalability since they must connect with other processors. Figure 12 shows an example connection of two vertical processors, where IO pins directly abut each other with very short connecting wires.

IV. A DESIGN EXAMPLE

We have designed and implemented a single-chip tile-based 6×6 GALS multiprocessor in $0.18\,\mu\mathrm{m}$ CMOS technology [5]. The processor targets computationally intensive DSP applications as well as some scientific applications. The inherent features of those applications make it efficient to adopt several architectural features which distinguish it from other processors, such as a simple architecture, small memories for each single processor, and nearest neighbor communication between processors. No cache is used in this processor system so no cache related concerns such as data consistency are needed. The chip utilizes the Artisan standard cell library and was auto placed and routed. Figure 13 shows the die

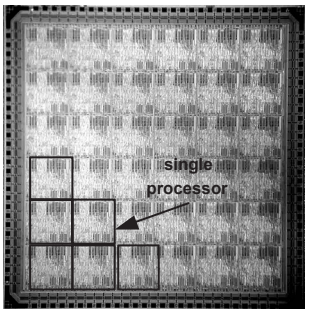

Fig. 13. Chip micrograph of a 6×6 GALS array processor [5]

micrograph. The chip is fully functional at a clock rate of 475 MHz under typical conditions at 1.8 V, and achieves a peak performance of 17 GOPS with a power consumption of approximately 3 W. The size of each processor is 0.66 mm^2. Each processor dedicates approximately 8% of its area to communication circuits, and less than 1% to each local clock oscillator. In order to provide a flexible and safe solution, from 0 to 4 synchronization registers are selectable by configuration at each asynchronous boundary.

The physical design flow shown in Fig. 2 was used. Verilog was used as the front end design language and was synthesized using Synopsys *Design Compiler*. The synthesized netlist was then imported into the automatic placement and routing tool, Cadence *Encounter* to do floorplanning, placement, clock tree insertion, routing, and in-place optimization to change the size of gates and optimize logic to alleviate wire delay effects. The result from the place and routing tool was imported into another custom layout tool (*icfb*) to do final layout editing such as pad bounding, IO pin labeling, and layout layer checking.

Intensive verification methods were used throughout the design process including: gate level dynamic simulation using *NC-Verilog*, static timing analysis using *Primetime*, DRC/LVS using *Calibre*, spice level simulation using *Nanosim*, and formal verification using *Tuxedo*. The entire back end design flow took approximately 4 person-months including setup of tools and standard cell libraries.

The final 6×6 chip design was extended from a 3×3 design a few days before tapeout from Verilog to GDS II in a total of 10 hours—clearly demonstrating the excellent scalability of this architecture and approach.

V. Summary

Implementation techniques for a tile-based GALS chip multiprocessor are discussed. This architecture improves system scalability and simplifies the physical design flow. At the same time, it imposes some design challenges. These include several timing issues related to inter-processor communication, inter-chip communication, and asynchronous boundaries within single processors. By carefully addressing these timing issues, it is possible to take full advantage of its scalability, and the processor architecture makes it possible to design a high performance system with a small design group within a short time period.

Acknowledgments

The authors thank E. Work, D. Truong, W. Cheng, T. Jacobson, T. Mohsenin, other VCL processor co-designers, R. Krishnamurthy, M. Anders, and S. Mathew; and support from Intel, UC MICRO, NSF Grant No. 0430090, MOSIS, Artisan, and a UCD Faculty Research Grant.

References

[1] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation cell processor," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2005, pp. 184–185.

[2] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, W. Lee, A. Saraf, N. Shnidman, V. Strumpen, S. Amarasinghe, and A. Agarwal, "A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2003, pp. 170–171.

[3] S. W. Keckler, D. Burger, C. R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. S. Hrishikesh, N. Ranganathan, and P. Shivakumar, "A wire-delay scalable microprocessor architecture for high performance systems," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2003, pp. 168–169.

[4] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," in *International Symposium on Computer Architecture (ISCA)*, June 2000, pp. 161–171.

[5] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas, "An asynchronous array of simple processors for DSP applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2006, pp. 428–429.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *IEEE International Conference on Design Automation (DAC)*, June 2003, pp. 338–342.

[7] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2002, pp. 29–40.

[8] Ryan W. Apperson, "A dual-clock FIFO for the reliable transfer of high-throughput data between unrelated clock domains," M.S. thesis, University of California, Davis, Sept. 2004.

[9] J.M.Rabaey, *Digital Integrated Circuits – A Design Perspective*, Prentice-Hall International, Inc, first edition, 1998.

[10] M. Fukase, T. Sato, R. Egawa, and T. Nakamura, "Scaling up of wave pipelines," in *International Conference on VLSI Design*, Jan. 2001, pp. 439–445.

[11] P. Cocchini, "Concurrent Flip-Flop and repeater insertion for high performance integrated circuits," in *IEEE International Conference on Computer Aided Design (ICCAD)*, Nov. 2002, pp. 268–273.

[12] B. R. Quinton, M. R. Greenstreet, and S. J.E. Wilton, "Asynchronous IC interconnect network design and implementation using a standard ASIC flow," in *IEEE International Conference on Computer Design (ICCD)*, Oct. 2005, pp. 267–274.