Performance and Power Analysis of Globally Asynchronous Locally Synchronous Multi-Processor Systems

Zhiyi Yu, Bevan M. Baas
ECE department, University of California, Davis
{zhyyu, bbaas}@ece.ucdavis.edu

Abstract

This paper investigates the performance and power dissipation of Globally Asynchronous Locally Synchronous (GALS) multi-processor systems. We show that communication loops are a source of significant throughput degradation in communications links and that there is no degradation whatsoever under certain conditions for one-way links, and that it is possible to design GALS multi-processors without this performance penalty. Independent clock domains and unbalanced computation in the GALS multiprocessor allow scaling of the clock frequency and supply voltage to achieve high energy efficiency. The synchronization overhead between independent clock domains results in a less than 1% performance reduction compared to a globally synchronous system over a number of DSP and numerical applications. Clock and voltage scaling can achieve an approximately 40% power savings with no reduction of performance. These results compare favorably with the 25% power savings and more than 10% performance reduction reported for GALS uniprocessors.

1. Introduction

Clocking circuits have become increasingly difficult to design with larger chip sizes, higher clock rates, larger relative wire delays, and larger parameter variations [1]. Additionally, high speed global clocks consume a significant portion of power budgets. The Globally Asynchronous Locally Synchronous (GALS) clocking style separates processing blocks such that each part is clocked by an independent clock domain. The approach is a promising strategy to address these design challenges. Previous GALS work includes performance and power analysis in an ASIC system [2] and a uniprocessor system [3, 4, 5]; and the clock domain analysis for a clustered array processor [6].

Modern deep submicron fabrication technologies are not able to sustain historical increases in clock frequencies, but do enable very high levels of integration such as chips with more than a billion transistors [7]. Multiple-processor chips now show a promising future [7, 8]. In this context, array processors—which combine multiple processors in an array—are increasingly attractive. An array processor can also provide high energy efficiency since parallel computing improves performance and may allow the clock frequency and voltage to be reduced.

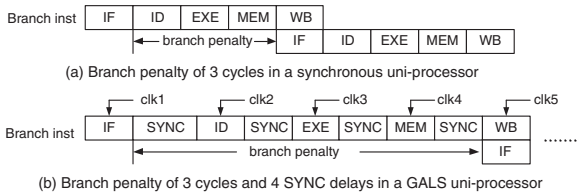

Figure 1. Pipeline control hazard penalties of a 5 stage synchronous uniprocessor and a 5 stage GALS uniprocessor

1.1. Performance reduction and energy efficiency of the GALS uniprocessor

In addition to multiple clock generators, GALS systems require synchronization circuits between clock domains to reliably transfer data. Small clock domains normally simplify clock trees, but unmatched clocks between different domains and synchronization circuitry introduce communication delays.

We define a *GALS uniprocessor* as an architecture where the processor itself is partitioned into multiple clock domains. The GALS overhead increases the delay between pipeline stages and reduces processor performance. Figure 1 shows the control hazard of a simple DLX RISC processor [9]. The lower subplot shows a GALS uniprocessor where each pipeline stage is in its own clock domain. During the cycle with the taken branch, the synchronous processor has a 3-cycle control hazard, while the GALS system has a $3 + 4 \times SYNC$ cycle penalty, significantly reducing system performance. Reported performance reductions of the GALS uniprocessor include 10% [3], 7%–11% [4] and 4% [5].

Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI'06)
0-7695-2533-4/06 $20.00 © 2006 IEEE

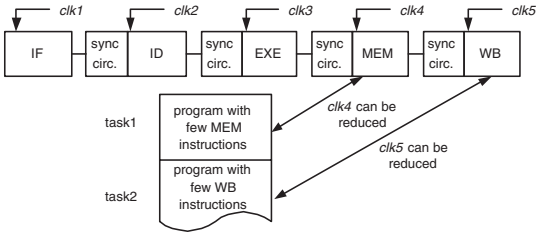

Figure 2. Clock scaling in a GALS uniprocessor

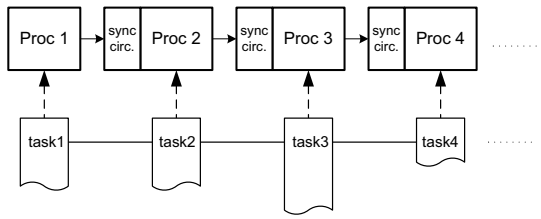

Figure 3. Mapping multiple tasks to a GALS array processor

GALS uniprocessors normally control their independent clock domains adaptively to achieve high energy efficiency, by reducing the clock frequency and voltage of modules that are less heavily used. Figure 2 illustrates this concept where the frequency of the MEM module clock *clk4* is reduced when executing task1 since it has few MEM instructions. Then in task2, the frequency of *clk5* is reduced. Unfortunately, reducing the clock of some modules reduces performance. The *static scaling* method sets the frequency before execution and reduces the energy by approximately 16% with an approximately 18% reduction in performance [3]. The *dynamic scaling* method changes the frequency at runtime and achieves 20%–25% energy savings along with a 10%–15% performance reduction [4, 5].

1.2. The GALS array processor

While the GALS uniprocessor puts synchronization logic between pipeline stages, the array processor puts synchronization circuits between different processors as shown in Fig. 3.

Clock frequency selection in GALS uniprocessors is based upon the utilization probability of each function module. Similarly, GALS array processors base their frequency selections on the activity of its computational tasks, as shown in Fig. 3.

2. A GALS Array Processor

Two array processor designs were implemented for comparison: one is fully synchronous and the other is a GALS array processor. Both contain multiple uniform simple processing units, as shown in Fig. 4a. Both processors contain

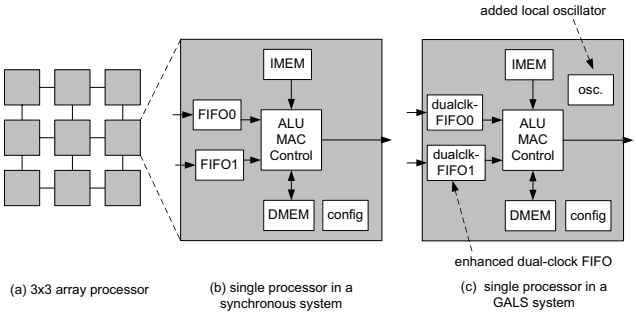

Figure 4. Two array processors: one using synchronous processors and the other using GALS array processors

small instruction and data memories, two 32-word FIFOs, a 16-bit datapath, and execute 32-bit instructions. Each processor communicates with its four neighboring processors.

Figure 4b shows a single synchronous array processor. It utilizes simple fully synchronous FIFOs. Figure 4c shows a single GALS array processor. In order to support the GALS methodology, a frequency configurable oscillator is added as the processor's local clock, and the synchronous FIFO is enhanced with features to allow it to perform as a dual-clock FIFO [10]. The dual-clock FIFO writes and reads data in independent clock domains and reliably transfers data across the domains. For increased characterization capability, a configurable number of synchronization registers are inserted at the clock domain interface to avoid metastability. The local oscillator occupies approximately 0.5% of the processor's area. The area overhead of the dual-clock FIFO is also around 0.5%. In addition, the GALS system has a simplified clock tree.

A 6×6 GALS array processor chip has been implemented [11]. The synchronous array processor is emulated by special configurations in the GALS array processing chip.

3. Performance Analysis of the GALS Array Processor

Several applications are mapped and simulated onto the RTL model of both synchronous and GALS array processors to investigate their performance. The synchronous system uses a global clock and has no synchronization registers. The GALS system uses a local oscillator and two synchronization registers.

The mapped applications we consider include: an 8-point DCT using 2 processors, an 8×8 DCT using 4 processors, a zig-zag transform using 2 processors, a merge sort using 8 processors, a bubble sort using 8 processors, a 5×5 matrix multiplier using 6 processors, a 64-point complex FFT using 8 processors, a JPEG encoder using 9 pro-

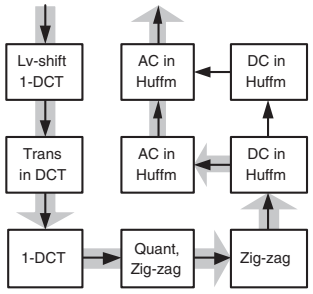

Figure 5. JPEG encoder core using 3x3 processors

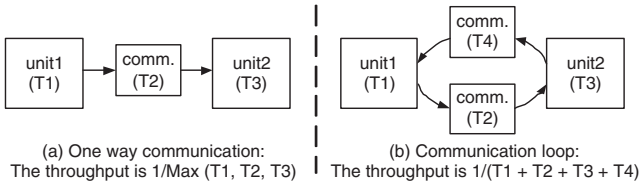

Figure 6. System throughput in a) one way communication path, and b) communication loop path

cessors, and an IEEE 802.11g/802.11a wireless LAN transmitter using 22 processors [12].

The JPEG encoder core composition as shown in Fig. 5 is one application example. The main functional blocks include a level shifter, an 8×8 DCT, quantization, zig-zag reordering, and a Huffman encoder. The 8×8 DCT is processed using two 1-dimensional DCTs. The second DCT data transpose is avoided by changing the quantization table order and zig-zag order. Four processors are used for the Huffman encoding.

3.1. Comparison of application performance

The first two lines of Table 1 show the computation time in clock cycles when mapping these applications onto the synchronous and GALS array processors. The third line lists the relative performance penalty of the GALS array processor. The performance of the GALS system is nearly the same as the synchronous system with an average of less than 1% performance reduction, which is much smaller than the 10% performance reduction of a GALS uniprocessor [3, 4].

3.2. Performance effects of GALS clocking

3.2.1 Importance of the communication loop delay

The performance penalty of a GALS system comes from its increased communication delay. More specifically, simple *one way communication* does not affect system performance, but *communication loops*—in which two units wait for information from each other—can degrade performance.

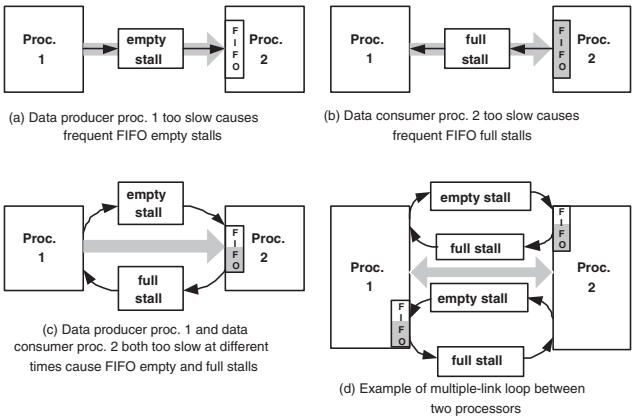

Figure 7. Examples of stalls and stall loops in a GALS array processor

In a *one way communication path* as shown in Fig. 6a, the system throughput is dependent on the slowest unit and is not related to the communication—assuming communication is not the slowest unit, which is true in our case. However, throughput is significantly impacted when the communication has feedback and generates a loop, as shown in Fig. 6b. If unit 1 and unit 2 both need to wait for information from each other, the throughout will be dependent on the sum of unit execution time and communication time. Then the communication time affects the performance of both synchronous and GALS systems, but the GALS system has a larger performance penalty due to its larger communication time.

A similar conclusion can be drawn from the GALS uniprocessor. In instructions without pipeline hazards, the GALS uniprocessor has the same performance as the synchronous uniprocessor since it has only *one way communication*. However, during instructions such as taken branches (where the new *PC* needs the feedback from the execution result), a *communication loop* is formed. Thus the GALS style brings a performance penalty in some cases. Other pipeline hazards also generate similar communication loops.

3.2.2 FIFO stall loops in GALS array processors generate communication loops

In both synchronous array processors and GALS array processors, *FIFO stalls* effect performance since one processor must wait for the information from another processor when they enter a stall status. GALS systems have a larger performance penalty than synchronous systems since they have larger latencies for FIFO stall information.

A *FIFO-empty stall* occurs when a processor reads an empty FIFO and must wait (stall) until data is available, as illustrated in Fig. 7a. A *FIFO-full stall* occurs when a processor writes a full FIFO and must wait until there is

Table 1. **Clock cycles (1/*throughput*) of several applications mapped onto a synchronous array processor and a GALS array processor, with 32-word FIFOs**

	8-pt DCT	8×8 DCT	zig-zag	mergesort	bubblesort	matrix	64 FFT	JPEG	802.11
Synchronous array	41	498	168	254	444	817.5	11439	1439	87857
GALS array	41	505	168	254	444	819	11710	1443	88989
GALS Perf. reduction	0%	1.4%	0%	0%	0%	0.1%	2.3%	0.3%	1.3%

writable space, as shown in Fig. 7b.

Pure FIFO-full stalls or FIFO-empty stalls alone as in Fig. 7a,b generate one way communication and have no effect on system throughput. The situation shown in Fig. 7c is one example of the FIFO stall communication loop. Processor 2 has FIFO-empty stalls and must wait for processor 1, and processor 1 has FIFO-full stalls and must wait for processor 2. When FIFO-full stalls and FIFO-empty stalls both exist (obviously at different times) in a link, they produce a communication loop and this reduces system performance—for both synchronous and GALS systems, albeit with less of a penalty for a synchronous system. Another situation which generates a FIFO stall communication loop is shown in Fig. 7d. In this case, processor 1 and processor 2 send data to each other, and each processor has both FIFO full stalls and FIFO empty stalls. An example of this case exists in our FFT application where some processors are used as data storing coprocessors and they send and receive data to computation processors.

Simulation results in Table 1 show that the GALS array processor has nearly the same performance as the synchronous array processor. This performance reduction is much less compared to the GALS uniprocessor's reduction. This implies that the chance of the FIFO stall loop in an array processor for our applications is much smaller than the probability of a pipeline hazard in a uniprocessor. These results match well with our model. In Table 1, the 8-pt DCT, zig-zag, mergesort and bubblesort have no GALS performance penalties since they have only one-way FIFO stalls. The 8×8 DCT and JPEG have situations like Fig. 7c and have an approximately 1% performance penalty. The 64-point FFT and 802.11g/a applications have situations like Fig. 7c and Fig. 7d and their performance penalty is slightly larger.

3.3. FIFO size affects synchronous and GALS system performance

FIFO stalls are highly dependent on the FIFO size. When the FIFO is large enough, there will be no FIFO-full stalls, and at the same time, the number of FIFO-empty stalls can be greatly reduced. With a sufficiently large FIFO, the communication loop in Fig. 7c will be broken due to the missing FIFO-full stalls. The likelihood of the communication loop in Fig. 7d will also be highly reduced, but is still possible since FIFO-empty stalls alone can still form a loop. Reduced FIFO stall loops increase the system performance and reduce the GALS performance penalty.

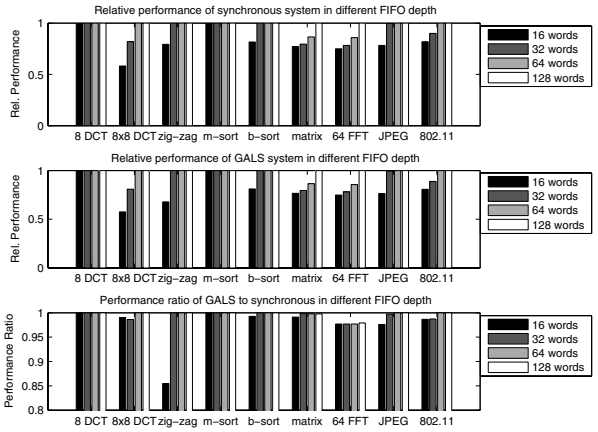

Figure 8. **Performance of synchronous and GALS array processors with different FIFO sizes**

The top and middle subplots of Fig. 8 show the performance of the synchronous and GALS systems with different FIFO sizes, respectively. Whether using a synchronous or GALS style, increasing the FIFO size will increase system performance because of reduced FIFO stall loops. Also, a threshold FIFO size exists above which the performance won't change. The threshold is the point when the FIFO-full stall becomes non-existent due to having a large enough FIFO size, and increasing the FIFO size further gives no benefit. The threshold is dependent on the application as well as the mapping method. In our case, the thresholds for the 8×8 DCT and 802.11g/a are 64 words; JPEG and bubble sort are 32 words; the 8-pt DCT and merge sort are less than or equal to 16 words.

The bottom subplot of Fig. 8 shows the performance ratio of the GALS system to the synchronous system. The ratio normally stays at a high level larger than 95%. When increasing the FIFO size, the ratio tends to increase due to fewer communication loops. The ratio normally reaches 1.0 at the threshold, which means the FIFO communication loops are all broken and the GALS system has the same performance as the synchronous system. The exception in the examples is the FFT in which the GALS system always

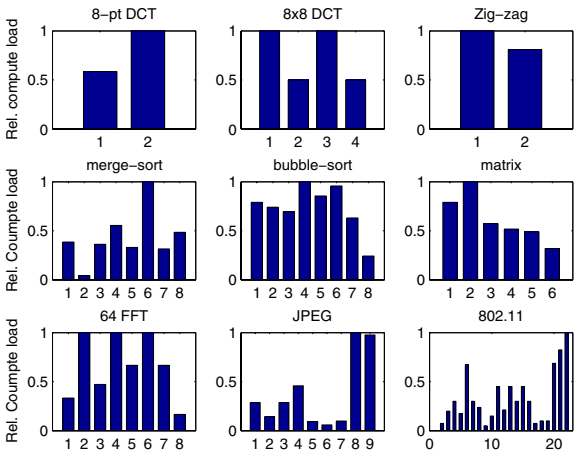

Figure 9. **Relative computational load of different processors in nine applications illustrating unbalanced loads**

has a noticeable performance penalty of approximately 2%. The reason can be seen from Fig. 7d where the FIFO-empty stall alone can generate the stall loop without a FIFO-full stall.

4. Energy Efficiency Analysis of Independent Clock Frequency Scaling

Several researchers have reported the high power efficiency of the GALS style due to its simplified clock tree [2, 6]. We focus on another power consumption benefit of the GALS system due to the flexibility of clock frequency and supply voltage scaling. The work presented here addresses static scaling methods.

The Synchroscalar [13] system utilizes processors with rationally-related clocks. While the approach avoids the extra hardware of asynchronous communication, its clocks are not as flexible as GALS clocks.

4.1. Unbalanced processor computation loads give power saving potential

Traditional parallel programming methods normally seek to balance computational loads in different processors. On the other hand, when using adaptive clock methods, unbalanced computational loads are no longer a problem, and in fact give an opportunity to reduce the clock frequency and supply voltage of some processors to achieve further power savings without degrading system performance [14]. Releasing the constraint of a balanced computational load enables the designer to explore wider variations in other parameters such as program size, local data memory size and communication methods. Figure 9 shows the unbalanced computational load among processors when mapping our applications onto an array processor.

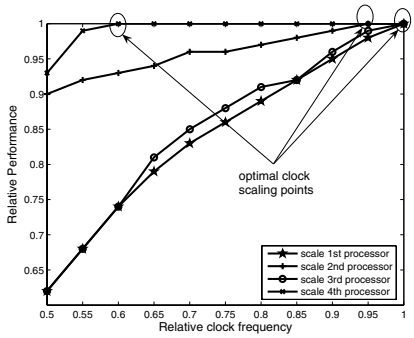

Figure 10. **Throughput changes with statically configured processor clocks for an 8×8 DCT**

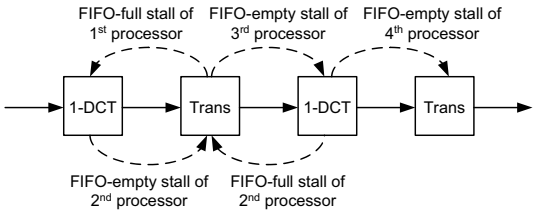

Figure 11. **Relationship of processors in an 8×8 DCT**

4.2. Computational load and position affect optimal clock frequency

The optimal processor clock frequency in a GALS array processor depends strongly on its computational load, and also depends on its position and relationship with respect to other processors.

Figure 10 shows the system throughput versus the clock frequencies of four processors in the 8×8 DCT. The computational load of the four processors is 408, 204, 408 and 204 clock cycles respectively. The throughput changes with the scaling of the 2^{nd} and 4^{th} processor much more slowly than the scaling of the 1^{st} and 3^{rd} processors. This illustrates the clear point that a processor with a light computational load is more likely to maintain its performance with a reduced clock frequency. Somewhat counterintuitively, however, the 2^{nd} and 4^{th} processors have the same light computational load, but the throughput changes with the 4^{th} processor scaling much more slowly than the 2^{nd} processor's scaling. Minimal power consumption is achieved with full throughput when the relative clock frequencies are 100%, 95%, 100%, and 57% of full speed respectively.

The reason for the different behavior of the 2^{nd} and 4^{th} processors comes from their different positions and FIFO stall styles as shown in Fig. 11. The 2^{nd} processor has both FIFO-full stalls and FIFO-empty stalls, while the 4^{th} processor has only FIFO-empty stalls.

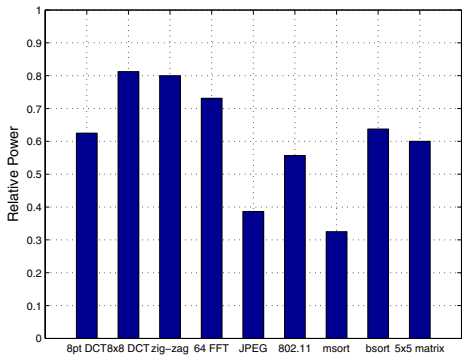

Figure 12. Estimated relative power of the GALS system with static clock/voltage scaling compared to a synchronous system

4.3. Estimating power reduction

Reducing the clock frequency allows for a reduction in voltage to obtain further power savings. The relationship between clock frequency and voltage, and power and voltage can be modeled by a simple linear relation and a square relation respectively. However, these relationships become much more complex in the deep submicron regime because of other parameters such as leakage power. For this analysis, we use a model derived from measured data from a 0.18 μm technology [15] to estimate power consumption.

Using the optimal clock frequency for each processor as described in Sec. 4.2, and the power-frequency-voltage model, we estimate the relative power consumption of the GALS array processor compared to the synchronous array processor after using static clock frequency and supply voltage scaling for several applications. The result is shown in Fig. 12. The GALS system achieves an average power savings of approximately 40% without affecting the performance. This power savings is much higher than the GALS uniprocessor which was reported to save approximately 25% power when operating with a performance reduction of more than 10% [3, 4, 5].

5. Summary and acknowledgments

It has been shown that communication loops are a source of significant throughput reduction in communication links and that there is no reduction under certain conditions for one-way links. A key advantage of the GALS array processor compared to the GALS uniprocessor is that communication loops occur far less frequently and therefore the performance penalty is significantly lower. The proposed GALS array processor has a throughput penalty of less than 1% with a power dissipation reduction of 40% over a variety of DSP and numerical workloads. These results compare well with a reported 25% power reduction and a 10% performance reduction with GALS uniprocessors.

Data presented in this paper are based on the fabricated GALS processor and its synchronous mode of operation [11]. While results will certainly vary over different applications and specific architectures, we expect the general conclusion that multi-processor GALS systems have smaller performance reductions and larger power reductions, should still hold.

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

References

[1] S. Borkar et al., "Parameter variations and impact on circuits and microarchitecture," in *DAC*, 2003, pp. 338–342.

[2] T. Meincke et al., "Globally asynchronous locally synchronous architecture for large high-performance asics," in *ISCAS*, May 1999, pp. 512–515.

[3] A. Iyer et al., "Power and performance evaluation of globally asynchronous locally synchronous processors," in *ISCA*.

[4] E. Talpes and D. Marculescu, "A critical analysis of application-adaptive multiple clock processor," in *ISLPED*.

[5] G. Semeraro et al., "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *HPCA*, 2002, pp. 29–40.

[6] A. Upadhyay et al., "Optimal partitioning of globally asynchronous locally synchronous processor arrays," in *GLSVLSI*, 2004, pp. 26–28.

[7] S. Naffziger et al., "The implementation of a 2-core multi-threaded Itanium family processor," in *ISSCC*, 2005.

[8] M. B. Taylor et al., "The raw microprocessor: A computational fabric for software circuits and general purpose programs," *IEEE Micro*, pp. 25–35, 2002.

[9] D. A. Patterson et al., *Computer Architecture – A Quantitative Approach*, Morgan Kaufmann, second edition, 1999.

[10] R. Apperson, "A dual-clock FIFO for the reliable transfer of high-throughput data between unrelated clock domains," M.S. thesis, UC Davis, 2004.

[11] Z. Yu et al., "An asynchronous array of simple processors for DSP applications," in *ISSCC*, Feb. 2006.

[12] M. Meeuwsen et al., "A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter," in *SiPS*, 2004, pp. 297–301.

[13] J. Oliver et al., "Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor," in *ISCA*.

[14] T. Njolstad et al., "A socket interface for gals using locally dynamic voltage scaling for rate-adaptive energy saving," in *ASIC/SOC*, Sept. 2001, pp. 110–116.

[15] K. Nowka et al., "A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *JSSC*, pp. 1441–1447, Nov. 2002.