High Performance and Energy Efficient Multi-core Systems for DSP Applications

By

ZHIYI YU
B.S. (Fudan University) June, 2000; M.S. (Fudan University) June, 2003

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Chair, Dr. Bevan M. Baas

Member, Dr. Vojin G. Oklobdzija

Member, Dr. Rajeevan Amirtharajah

Member, Dr. Soheil Ghiasi

Committee in charge
2007

© Copyright by Zhiyi Yu 2007
All Rights Reserved

Abstract

This dissertation investigates the architecture design, physical implementation, result evaluation, and feature analysis of a multi-core processor for DSP applications. The system is composed of a 2-D array of simple single-issue programmable processors interconnected by a reconfigurable mesh network, and processors operate completely asynchronously with respect to each other in a Globally Asynchronous Locally Synchronous fashion. The processor is called Asynchronous Array of simple Processors (AsAP). A 6×6 array has been fabricated in a 0.18 μm CMOS technology. The physical design concerns timing issues for robust implementations, and takes full advantages of their potential scalability. Each processor occupies 0.66 mm^2, is fully functional at a clock rate of 520–540 MHz under 1.8 V, and dissipates 94 mW while the clock is 100% active. Compared to the high performance TI C62x DSP processor, AsAP achieves performance 0.8–9.6 times greater, energy efficiency 10–75 times greater, with an area 7–19 times smaller. The system is also easily scalable, and is well-suited to future fabrication technologies.

An asymmetric interprocessor communication architecture is proposed. It assigns different buffer resources to the nearest neighbor interconnect and the long distance interconnect, can reduce the communication circuitry area by approximately 2 to 4 times compared to the traditional Network on Chip (NoC), with similar routing capability. A wide design exploration space is investigated, including supporting long distance communication in GALS systems, static/dynamic routing, varying numbers of ports (buffers) for the processing core, and varying numbers of links at each edge.

The use of GALS style typically introduces performance penalties due to additional communication latency between clock domains. GALS chip multiprocessors with large inter-processor FIFOs as AsAP can inherently hide much of the GALS performance penalty, and the penalty can even be driven to *zero*. Furthermore, adaptive clock and voltage scaling for each processor provides an approximately 40% power savings without any performance reduction.

Acknowledgments

I would like to thank all of the individuals who made this work possible.

I want to thank professor Bevan Baas, for his academic guidance and financial support. His devotion and enthusiasm on research will effect me strongly in my future career. I want to thank my dissertation reading committee members including professor Vojin G. Oklobdzija, professor Rajeevan Amirtharajah, and professor Soheil Ghiasi, for their useful comments on my research. I want to thank previous VCL group members including Michael Lai, Omar Sattari, Ryan Apperson and Michael Meeuwsen. It was with them that I had a happy time when I first came to Davis, and their efforts and contributions helped to make the AsAP project successful, which becomes the strong basis of my dissertation. I want to thank current VCL group members including Eric Work, Tinoosh Mohsenin, Jeremy Webb, Wayne Cheng, Toney Jocobson, Zhibin Xiao, Paul Mejia, and Anh Tran; I always found I can learn something from them and they keep VCL group active.

I want to thank Intel Corporation, Intellasys Corporation, Xilinx, National Science Foundation (grant No. 0430090 and CAREER Award 0546907), UC MICRO, ST Microelectronics, SEM, SRC, and UC Davis Faculty Research Grant, for their generous financial donations to our research. I also want to thank Artisan, MOSIS, and TSMC; it was with their help that we had a successful fabrication of the AsAP chip.

I want to thank my family, my relatives and all of my friends. You might not know the details of my research area, but the support and help I get from you all might be more important than the pure academic help. It is because of you that I am a happy person and can keep pursuing my dreams.

Contents

Abstract **iii**

Acknowledgments **iv**

List of Figures **viii**

List of Tables **xi**

1 Introduction **1**

1.1 Challenges 2

1.1.1 High performance and energy efficiency 2

1.1.2 Future fabrication technologies 5

1.2 Solution — multi-core systems 6

1.3 Contributions 8

1.4 Dissertation organization 9

2 Architecture of the Multi-core System **11**

2.1 Key features of the multi-core processor 11

2.1.1 Chip multiprocessor and task level parallelism 12

2.1.2 Memory requirements of the targeted tasks 13

2.1.3 Simple single issue datapath 14

2.1.4 GALS clocking style 15

2.1.5 Wires and on chip communication 17

2.2 The AsAP processor system 18

2.2.1 Single AsAP processor design 19

2.2.2 Inter-processor communication — Reconfigurable 2-D mesh network . . . 26

2.3 Application implementations and software 29

2.3.1 Application implementations 29

2.3.2 Software 31

2.4 Related work 32

2.4.1 Traditional DSP processors 32

2.4.2 Pioneering multiprocessor systems and multi-core processors 33

2.4.3 Modern multi-core systems 36

2.4.4 Distinguishing multi-core processors 40

2.5 Summary 44

3 An Low-area Multi-link Interconnect Architecture **45**
3.0.1 Background: traditional dynamic routing architecture . . . 46
3.0.2 Background: static nearest neighbor interconnect . . . 47
3.1 Low-area interconnect architecture . . . 48
3.1.1 Asymmetric architecture . . . 48
3.1.2 Theoretical analysis . . . 50
3.1.3 Static routing vs. dynamic routing . . . 53
3.2 Design space exploration . . . 56
3.2.1 Single port vs. multiple ports for the processing core . . . 56
3.2.2 Single link vs. multiple links . . . 59
3.3 Supporting GALS long distance communication . . . 67
3.3.1 Source synchronization for long distance communication . . . 69
3.3.2 Care more about the clock delay, less about skew or jitter . . . 70
3.4 Implementation and results . . . 70
3.4.1 Area . . . 71
3.4.2 Performance comparison . . . 72
3.5 Summary . . . 75

4 Physical Implementation of the GALS Multi-core Systems **77**
4.1 Timing issues of GALS multi-core systems . . . 77
4.1.1 Inter-processor timing issues . . . 78
4.1.2 Inter-chip timing issues . . . 81
4.2 Scalability issues of GALS chip multiprocessors . . . 82
4.2.1 Clocking and buffering of global signals . . . 83
4.2.2 Power distribution . . . 84
4.2.3 Position of IO pins . . . 85
4.3 A design example — implementation of AsAP . . . 85
4.3.1 Physical design flow . . . 86
4.3.2 Implementation for high speed . . . 88
4.3.3 Testing . . . 89
4.4 Summary . . . 90

5 Results and Evaluation of the Multi-core System **93**
5.1 Area, speed, and power . . . 93
5.1.1 Small area and high area efficiency . . . 94
5.1.2 High speed . . . 96
5.1.3 High peak performance and low average power consumption . . . 96
5.1.4 Result of communication circuitry . . . 98
5.2 High performance and low power consumption for DSP applications . . . 98
5.2.1 Performance scaling with the processor number . . . 102
5.3 Summary . . . 103

6 System Feature Analysis: GALS vs. Synchronous **105**
6.1 Exploring the key GALS chip multiprocessor design options . . . 106
6.1.1 Clock domain partition of GALS chip multiprocessors . . . 106
6.1.2 Inter-processor network . . . 107
6.2 Simulation platform — the GALS and non-GALS chip multiprocessors . . . 108
6.3 Reducing and eliminating GALS performance penalties . . . 109

6.3.1 Related work 109
6.3.2 Comparison of application performance: GALS versus synchronous 110
6.3.3 Analysis of the performance effects of GALS 111
6.3.4 Eliminating performance penalties 117
6.4 Scalability analysis of GALS chip multiprocessors 119
6.4.1 Auto generated clock trees for different sizes of chip multiprocessors . . . 119
6.4.2 The effect of clock tree on system performance 121
6.5 Power analysis of adaptive clock frequency scaling 123
6.5.1 Related work—adaptive clock scaling of the GALS uniprocessor 124
6.5.2 Unbalanced processor computation loads increases power savings potential 125
6.5.3 Finding the optimal clock frequency—computational load and position . . 126
6.5.4 Power reduction of clock/voltage scaling 127
6.6 Summary 128

7 Conclusion **131**
7.1 Future work 132

Glossary **135**

Bibliography **139**

List of Figures

1.1 Power consumption of Intel microprocessors from 1970 to 2006 4
1.2 Wire delays (in FO4s) for fixed-length (1 cm) wires 5

2.1 Key features of AsAP and resulting benefits 12
2.2 Multi-task application executing models 12
2.3 IEEE 802.11a/g wireless LAN (54 Mbps, 5/2.4 GHz) baseband transmit path . . . 13
2.4 Area breakdown of four modern processors; memory occupying most of the area . 13
2.5 Block diagrams of synchronous and GALS tile-based chip multiprocessors 16
2.6 Some *linear pipelined* algorithm models for DSP tasks and applications 18
2.7 Block diagram of an AsAP processor 19
2.8 AsAP 9-stage pipeline 20
2.9 Programmable clock oscillator 21
2.10 Physical distribution of oscillator frequencies across different processors 22
2.11 Example waveform of clock halting and restarting 22
2.12 Relative instruction memory cost by using *embedded NOP* instruction 24
2.13 Comparison of three different addressing modes 26
2.14 Nearest neighbor inter-processor communication diagram 27
2.15 Two strategies for communication in GALS systems 28
2.16 Block diagram of the dual-clock FIFO 28
2.17 8 x 8 DCT implementation using 4 processors 30
2.18 JPEG encoder core using 9 processors 31

3.1 Interprocessor communication in NoC systems and a generalized NoC architecture 46
3.2 The concept and circuitry of the nearest neighbor interconnect architecture 47
3.3 The concept and circuitry diagram of the proposed communication architecture . . 50
3.4 Buffer stall probability (p) along with buffer size (N) 51
3.5 The overall system buffer stall probability along the buffer size ratio 53
3.6 Diagrams of architectures with various numbers of input ports for the processing core 56
3.7 The latency of architectures with different numbers of ports for one-to-one comm . 57
3.8 The latency of architectures with different numbers of ports for all-to-one comm. . 58
3.9 The latency of architectures with different numbers of ports for all-to-all comm. . . 59
3.10 Diagram of inter-processor connection with one link at each edge 60
3.11 Inter-processor connections with double links 61
3.12 Setup the interconnect path from point A to B 63
3.13 Inter-processor interconnect with three and four links 64
3.14 layouts of seven processors with different communication circuitry 65

3.15 Comparing the processors with different communication circuitry 66
3.16 The latency of interconnect architectures for one-to-one comm. 67
3.17 The latency of interconnect architectures for all-to-all comm. 68
3.18 The latency of interconnect architectures for all-to-one comm. 68
3.19 Synchronization strategies for GALS long distance communication 69
3.20 The relative communication circuit area of several interconnection architectures . . 71
3.21 The latency of different interconnect architectures for one-to-one comm. 72
3.22 The latency of different interconnect architectures for all-to-one comm. 73
3.23 Comparing the communication latency of three architectures 73
3.24 The latency of application models uniformly combined by the four basic patterns . 74

4.1 An overview of timing issues in GALS chip multiprocessors 78
4.2 Three methods for inter-processor communication 79
4.3 Relative clock active time and communication power consumption 79
4.4 Circuit for the Fig. 4.2 (b) inter-processor communication method 80
4.5 Configurable logic at the inter-processor boundary 80
4.6 Inter-chip communication 82
4.7 Global signals controlled by a low-speed clock 83
4.8 An example power distribution scheme 84
4.9 Pins connections between two processors vertically 85
4.10 Pins connections between two processors horizontally 86
4.11 Chip micrograph of a 6×6 GALS array processor 87
4.12 Hierarchical physical design flow of a tile-based GALS chip multiprocessor 88
4.13 A standard cell based back end design flow emphasizing the verification 89
4.14 Speed-up methods during synthesis and place and routing 90
4.15 AsAP board and supporting FPGA-based test board 91

5.1 Area evaluation of AsAP processor and several other processors 94
5.2 Size of a single processing element in several chip multi-processor systems 95
5.3 Processor shmoo: voltage vs. speed 96
5.4 Power and performance evaluation of AsAP processor and several other processors 97
5.5 Comparison of communication circuit of four chip multiprocessors 98
5.6 Relative area for various implementations of several DSP kernels and apps. 100
5.7 Relative execution time for various implementations of several DSP apps 101
5.8 Relative energy for various implementations of several DSP apps 101
5.9 Increase in system throughput with increasing number of processors 102

6.1 Three example clock domain partitions 107
6.2 Two chip multiprocessors 108
6.3 A GALS system boundary and timing of the synchronization delay 109
6.4 Pipeline control hazard penalties 110
6.5 Illustration of three key latencies and application throughput in GALS systesms . . 112
6.6 FIFO operation during normal situation, and FIFO empty, and FIFO full 114
6.7 System throughput in one way communication path, and communication loop path 115
6.8 Data producer proc.1 and data consumer proc.2 both too slow 116
6.9 Performance of synchronous and GALS array processors with different FIFO sizes 117
6.10 Examples of multiple-loop links between two processors 118
6.11 64-pt compelx FFT implementation 119

6.12 An example clock tree for a single processor 120
6.13 The peak performance of GALS and synchronous array processor 122
6.14 Increased clock tree delay at 1.7 V supply voltage for different clock trees 123
6.15 The peak performanc with the number of processors 124
6.16 Clock scaling in a GALS uniprocessor 125
6.17 Relative computational load of different processors in nine applications 126
6.18 Throughput changes with statically configured processor clocks 127
6.19 Relationship of processors in the 4-processor 8×8 DCT application 127
6.20 The relationship between clock frequency and its power consumption 128
6.21 Relative power of the GALS array processor 129

List of Tables

1.1 Example multi-core processors presented in ISSCC from 2003 to 2007 7

2.1 Memory requirements for common DSP tasks 14
2.2 AsAP 32-bit instruction types and fields 23
2.3 Classes of the 54 supported instructions 23
2.4 Data fetch addressing modes 25
2.5 Computation load of the nine processors in JPEG encoder 31
2.6 Major features of some commercial DSP processors 33
2.7 Comparison of the distinguished key features of selected parallel processors 41
2.8 Comparison of the selected parallel processors 42
2.9 Comparison of inter-element communication of selected parallel processors 43

3.1 Data traffic of router's input ports and output ports for each processor 49
3.2 The required buffer sizes for defined stall probabilities 52
3.3 Comparison of different interconnect approaches 55
3.4 Interconnect architecture options for double links 62

4.1 Typical timing constraint values for processor input and output delays 82

5.1 Area breakdown in a single processor 94
5.2 Estimates for a 13 mm $\times$ 13 mm AsAP array implemented in various technologies 97
5.3 Area, performance and power comparison of various processors for several apps . . 99

6.1 Clock cycles (1/*throughput*) of several applications 111
6.2 The fraction of the time the inter-processor communication is active 113
6.3 Effective latency (clock cycles) of several applications 115
6.4 Data for globally synchronous clock trees for different array processors 121

Chapter 1

Introduction

Applications that require the computation of complex digital signal processing (DSP) workloads are becoming increasingly commonplace. These applications often comprise multiple DSP tasks and are frequently key components in many systems such as: wired and wireless communications, multimedia, large-scale multi-dimensional signal processing (e.g., medical imaging, synthetic aperture radar), some large-scale scientific computing, remote sensing and processing, and medical/biological processing. Many of these applications are embedded and are strongly energy-constrained (e.g., portable or remotely-located) and cost-constrained. In addition, many of them require very high throughputs, often dissipate a significant portion of the system power budget, and are therefore of considerable interest.

There are several design approaches for DSP applications such as ASICs, programmable DSPs and FPGAs. ASICs can provide very high performance and very high energy efficiency, but they have little programming flexibility. On the other hand, programmable DSPs are easy to program but their performance and energy efficiency is normally 10–100 times lower than ASIC implementations [1]. FPGAs fall somewhere in between. One-time fabrication costs for state of the art designs (e.g., 90 nm CMOS technology) are roughly one million dollars, and total design costs of modern chips can easily run into the tens of millions of dollars [2]. Programmable processors are the platforms be considered in this dissertation to allow high one-time fabrication costs and design costs to be shared among a variety of applications; but higher performance and energy efficiency are expected to be achieved using the architecture proposed in this dissertation compared to the

traditional programmable DSPs.

1.1 Challenges

Previous IC designers have been mainly concerned with fabrication cost and performance; minimizing the number of transistors to reduce the area is the main approach to reduce the cost; and increasing the clock frequency is the main approach to increase the performance. Currently, how to achieve energy efficiency and how to adapt to the advanced fabrication technologies become important challenges.

1.1.1 High performance and energy efficiency

High performance innovations are challenging

Increasing the processor clock frequencies and using wide issue processor architectures have worked well to improve performance but recently have become significantly more challenging.

Deeper pipelining is one of the key techniques to increase the clock frequency and performance, but the benefit of the deeper pipeline is eventually diminished when the inserted Flip-Flop's delay is comparable to the combinational logic delay. Moreover, deeper pipeline stage increases cycles-per-instruction (CPI) and impacts negatively to the system performance. Researchers found that the depth per pipeline is approximately 8 Fanout-4 (FO4) inverter delays to obtain highest performance [3], which corresponds to 20–30 pipeline stages. The pipeline delay of some modern processor is already close to 10 FO4 [4] so that the deeper pipelining technique for high performance is reaching its limit. Also, increasing pipeline stages necessitates more registers and control logic, thereby further increasing design difficulty as well as power consumption. As reported by A. Hartstein [5], the optimum pipeline depth for maximum energy efficiency is about 22.5 FO4 delay (about 7 stage pipeline), using BIPS^3/Watt as the metric—BIPS are billions of instructions per second.

The other key technology—shrinking the size of transistors to increase the clock frequency and integration capability—has amazingly followed Moore's Law [6] for about 40 years. Although the pace of this innovation is still going on, the limitations are right ahead in another couple of generations: either because of the physical limit when the size of transistors approaches the

size of atoms, or because of the fabrication cost prohibiting further progress.

Wide issue processor architectures such as VLIW and superscalar are another efficient approaches for high performance computation while their benefit is also quickly diminished when the issue width is more than 10; since most applications don't have so many independently parallel executable instructions in per fetch. For example, an 8-way VLIW TI high performance C6000 DSP [7] and an 8-wide superscalar RISC microprocessor [8] were reported in 2002, and there is no examples beyond this range till now.

Power dissipation becomes the key constraint

The high performance design of modern chips is also highly constrained by the power dissipation as well as the circuit constraints. Power consumption is generally dominated by dynamic power with the trend that leakage power is playing another key role. The dynamic power is formed by Equation 1.1

$$P = \alpha C V^2 f \tag{1.1}$$

where α is the circuit state transition probability, C is the capacitance, V is the supply voltage, and f is the clock frequency. The leakage power mainly results from the reduction of the transistor threshold voltage [9] and is also highly dependent on the supply voltage. Most high performance technologies, such as increasing clock frequencies and increasing processor issues (which means increasing number of circuits and increasing capacitance) result in higher power consumption; all these imply a new era of high-performance design that must now focus on energy-efficient implementations [10].

Portable devices powered by battery certainly concern the power consumption since it determines their operational life time between each battery charging. Traditional non-portable systems such as PC also concern power consumption, since it highly determines the packaging costs, cooling system costs, and even limits the operation speeds and integration capacities of the systems. Figure 1.1 shows the power consumption of main Intel microprocessors from 1970 to 2006. The data between 1970 to 2000 is from S. Borkar [11] where he found that the changes of the power consumption follow the Moore's law increasing from 0.3 W to 100 W, and estimated that the processor power consumption will go up to 1000 W in a couple of years if this trend contin-

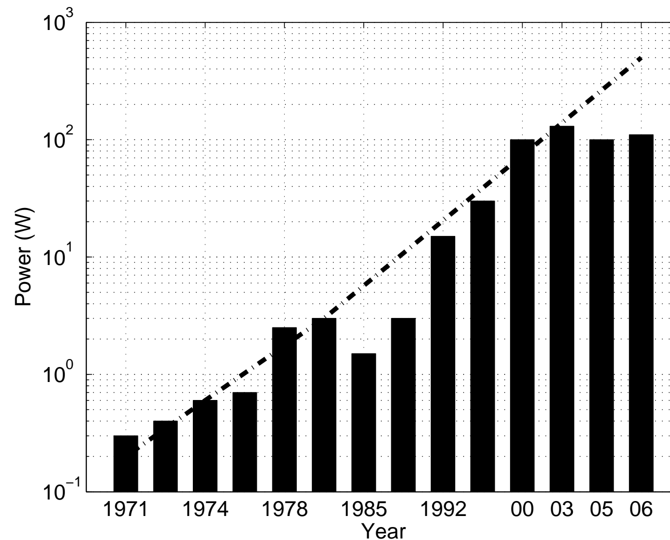

Figure 1.1: Power consumption of Intel microprocessors from 1970 to 2006; data between 1970 to 2000 are from [11]; data in 03, 05, 06 are from [12, 13, 14] respectively. Bars are the real values and the dashed line shows the trend

ues. Similarly to the power consumption, the power density increased significantly from a couple of Watts per mm^2 to about 100 Watts per mm^2 and becomes another key issue. This trend is halted in these years thanks to low power techniques such as voltage and frequency control technologies. The power consumption of recently reported microprocessors is still around 100 W. It also implies that 100 W is the power limit the current packaging technology and cooling technology can tolerate. Power consumption has become the highest constraint for designers and limits the achievable clock frequency and processor performance [10].

The gap between dream and reality

H. D. Man shows a future ambient intelligent computing example which illustrates the gap between the future requirement and the current reality in both performance and power [15]. In his opinion, there will be three major devices in the future intelligent computing system. One is the main powerful computation components, like today's PC; its target performance is 1 TOPS, with power consumption less than 5 Watts, corresponding to the energy efficiency 100 to 200 GOPS/W. This requirement is about 1000 times higher than today's PC which has about 10 GOPS while consuming 100 W, corresponding to 0.1 GOPS/W. The second device is the handable devices powered by

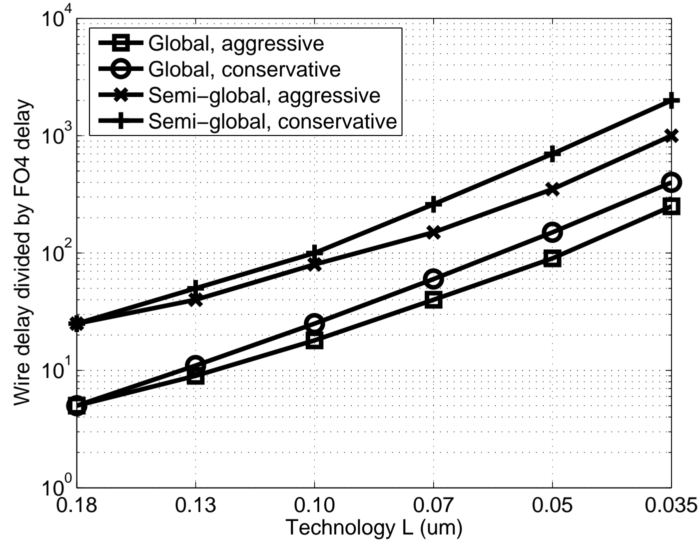

Figure 1.2: Wire delays (in FO4s) for fixed-length (1 cm) wires [16]; global wires have larger width, height, and spacing which result in smaller resistance than semi-globle wires; aggressive and conservative are two extreme scaling projections to bound scaling parameters.

battery, like current mobile phone, targets to 10 to 100 GOPS with less than 1 W, corresponding to 10 to 100 GOPS/W. This requirement is about 10 times higher than current solutions which use RISC processors and/or DSP processors. The third component is the sensor network to receive and transfer information, powered by energy harvesting methods such as mechanical vibration, with power consumption less than 100 uW; developing such low power components is another challenging topic.

1.1.2 Future fabrication technologies

Future fabrication technologies are also imposing new challenges such as wiring and parameter variations.

In the early days of CMOS technology, wires could be treated as ideal. They transmit signals with infinite speed, without power consumption, and without coupling effect. This assumption is no longer true. For global wires, their length is nearly constant along with the technology scaling if the chip size stays the same; which makes their delay nearly constant. Compared to gate delay which scales down with the technology, the delay of the *global* and *long* wires scales up with the technology. As shown in Fig 1.2 [16], a 1 cm long global wire delay in modern 65 nm technol-

ogy is around 100 FO4 delay; which corresponds to more than one clock cycle period in modern processors and the global wires have to be pipelined or be eliminated through architecture level innovations. Similar with the delay, the power consumption of the long wires also scales up along with the technology compared to the gates. Besides the effect on delay and power consumption, the inductive and capacitive coupling between wires adds signal noise and impacts system reliability.

Furthermore, future fabrication technologies are expected to have tremendous variability compared to current technologies in both gates and wires. Fundamental issues of statistical fluctuations for submicron MOSFETs are not completely understood, but the variation increases leakage of transistor and causes a variation of the speed of individual transistors, which in turn leads to IC timing issues [17]. Borkar et al. reported chips fabricated in advanced nanometer technologies can easily have 30% variation in chip frequencies [18]. This variance will be present at time of fabrication and also have a time-varying component.

1.2 Solution — multi-core systems

In order to address the challenges in performance, power and future technologies, innovations on computer architecture and design are needed; and multi-core systems are one of the most, or the most promising technology. As was also pointed out by a computer architecture group at EECS department of UC Berkeley [19]: the *"shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures"*.

Deep submicron fabrication technologies enable very high levels of integration such as a recent dual-core chip with 1.7 billion transistors [13], thus reaching a key milestone in the level of circuit complexity possible on a single chip. A highly promising approach to efficiently use these circuit resources is the integration of multiple processors onto a single chip to achieve higher performance through parallel processing, which is called a multi-core system or a chip multiprocessor (CMP) system.

Multi-core systems can provide high energy efficiency since they can allow the clock frequency and supply voltage to be reduced together to dramatically reduce power dissipation during

Table 1.1: Major commercial general-purpose multi-core processors presented in ISSCC before 2007

Year	Dual-core	Quad-core	8-core
2003	Mitsubishi's dual-core [20]		
2004	UltraSPARC [21]		
2005	Itanium [13] SPARC [24] BlueGene/L [25]	Fujitsu's 4-core [22]	CELL [23]
2006	Xeon [14] x86 [27] Power^{TM} [28]		Niagara1 [26]
2007	POWER6 [29] Power^{TM} [32] Merom [34]	Renesas' 4-core SoC [30] Opteron [33]	Niagara2 [31]

periods when full rate computation is not needed. Giving a simple example, assuming one uniprocessor is capable of computing one application with clock rate f and voltage v, and consuming power p; now if using a dual core system and assuming the application can be partitioned into two cores without any overhead, then each core only needs a clock rate $f/2$ and the voltage can be reduced accordingly; assuming a linear relation between voltage and clock frequency and the voltage is reduced to $V/2$, then the power dissipation of the dual core system will only be about $p/4$ calculated by Equation 1.1. Multi-core systems also potentially provide the opportunity to independently control each processor's clock and voltage to achieve higher energy efficiency, if different processors are in separate clock and voltage domains.

Furthermore, multi-core systems are suitable for future technologies. The distributed feature can potentially constrain the wires into one core and eliminate the global (long) wires. The multi-core systems also provide flexible approaches to treat each core differently by adjusting the mapped application, supply voltage, and clock rate; to utilize each core's specific features due to variations. For example, when one processor in the chip is much slower than the others, a low workload can be mapped on it without effecting system performance.

The multi-core systems have high scalability since a single processor can be designed and the system can be obtained by combining multiple processors. Thus the systems can be easily adjusted according to the required performance and cost constraints by changing the number of cores; which is much easier than changing the issue-width or the pipelining of uni-processors.

Table 1.1 shows the major commercial general-purpose multi-core processors presented in ISSCC before 2007. Multi-core processors started to emerge in 2003 and had a jump in 2005 and currently quad-core processors have become the main style. The trend to migrate from uniprocessors to multi-core processors is clear. The research community presented some more aggressive multi-core systems such as 16-core RAW [35] in 2003, 36-core AsAP [36] in 2006, and 80-core Intel chip [37] in 2007.

Although the trend is clear, there are a lot of issues that remain unclear to design an efficient multi-core processor. For example, what kind of processing element should be used in the multi-core systems; how to efficiently connect and communicate these multiple processing elements; and what is the clocking style to be used, etc. These are the questions this dissertation will try to investigate. Another big issue is related to the software such as how to describe applications and kernels to expose their parallel features; and how to program the multi-core processor using high level language; but those are beyond the main topic of this thesis.

1.3 Contributions

This dissertation makes a couple of contributions.

- It presents a successful multi-core processor, which is called Asynchronous Array of simple Processors (AsAP) [36, 38, 39], including its architecture design, physical implementation, application programming and results evaluation. The chip is organized by multiple simple single issue processors, interconnected by a reconfigurable mesh network, and operating in a globally asynchronous locally synchronous (GALS) [40] clock style. The system can efficiently compute many DSP applications, to achieve high performance, high energy efficiency, and is well suited for implementation in future fabrication technologies. The works not responsible by me are not included in this dissertation. Please refer to the publications of our group for the additional information, such as the details of the dual-clock FIFO design [41], the design of the large shared memory [42], the programming of an 802.11a/g wireless transmitter [43, 36], and the design of the MAC unit [44], etc.

- It generalizes the physical implementation techniques for multi-core processors with GALS

clock styles [45]. It presents the methodologies to handle the timing issues in GALS chip multi-processors including the inter-processor and inter-chip timing issues as well as the timing within a single processor. It takes full advantage of system scalability by taking care of the unavoidable global signals, the power distribution, and the processor IO pins.

- It investigates the performance, scalability, and power consumption effect when adopt GALS style to multi-core processors, by comparing to the corresponding totally synchronous systems [46]. It finds that this GALS array processor has a throughput penalty of less than 1% over a variety of DSP workloads, and this small penalty can be further avoided by large enough FIFOs and programming without multiple-loop communication links. It shows that unbalanced computational loads in chip multiprocessors increases the opportunity for independent clock frequency and voltage scaling to achieve significant power consumption savings.

- It proposes an asymmetric inter-processor communication architecture which uses more buffer resources for the nearest neighbor connections and fewer buffer resources for the long distance interconnect, and can save the area 2 to 4 times compared to the traditional NoC system while maintaining similar communication performance. It investigates the methodologies to support GALS clocking in long distance communication. It finds that using two or three links between each neighboring processors can achieve good area/performance trade offs for chip multiprocessors organized by simple single issue processors.

1.4 Dissertation organization

The organization of this dissertation is as follows. After the introduction, Chapter 2 introduces the architecture of the multi-core processor; Chapter 3 investigates the inter-processor communication system; Chapter 4 discusses and generalizes the physical design of tile-based GALS multi-core systems; Chapter 5 evaluates the measurement results of the fabricated multi-core chip; Chapter 6 analyzes the GALS effect on system features; and finally is the conclusion.

Chapter 2

Architecture of the Multi-core System

A multi-core processor system is proposed and designed, mainly for computationally-intensive DSP applications. The system comprises a 2-D array of simple programmable processors interconnected by a reconfigurable mesh network. Processors are each clocked by fully independent haltable oscillators in a GALS fashion.

The multi-core architecture efficiently makes use of task level parallelism in many complex DSP applications, and also efficiently computes many large DSP tasks through fine-grain parallelism to achieve high performance. The system uses processors with simple architecture and small memories to dramatically increase energy efficiency. The flexible programmable processor architecture broadens the target application domain and allows high one-time fabrication costs to be shared among a variety of applications. The GALS clocking style and the simple mesh interconnect greatly enhance scalability, and provide opportunities to mitigate effects of device variations, global wire limitations, and processor failures.

This chapter discusses the key features and architecture level design of the proposed multi-core system which is called AsAP [36, 39] for an *Asynchronous Array of simple Processors*.

2.1 Key features of the multi-core processor

Several key features distinguish the AsAP processor. These features and the resulting benefits are illustrated in Fig. 2.1 and are discussed in greater detail in the following subsections.

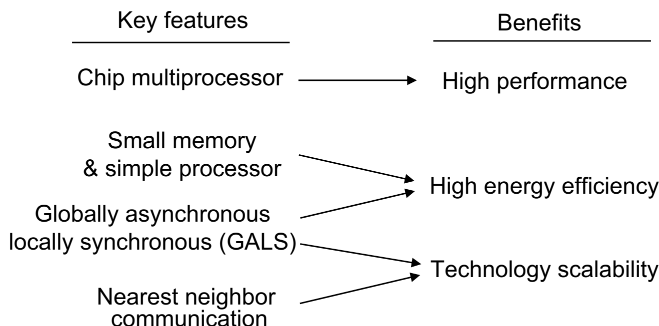

Figure 2.1: Key features of AsAP and resulting benefits

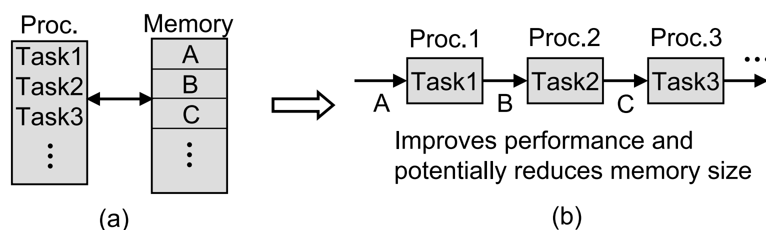

Figure 2.2: Multi-task application executing on (a) a traditional architecture, and (b) a stream-oriented multi-processor well suited for task level parallelism

2.1.1 Chip multiprocessor and task level parallelism

As discussed in Section 1.2, parallelization through multi-core systems are the future for high performance computation. Multi-core systems can use all types of parallelism techniques such as instruction-level parallelism, data-level parallelism and task-level parallelism.

Task-level parallelism can not be easily used on traditional sequentially-executing processors, but it is especially well suited for many DSP applications. As shown in Fig. 2.2 (a), the traditional system normally contains a powerful processor with a large memory, and executes the tasks of the application in sequence and stores temporary results into memory. The same application may be able to run on multiple processors using task level parallelism more efficiently as shown in Fig. 2.2 (b), where different processors handle different tasks of the application. Normally the data input of DSP applications is considered infinite length, so these processors can execute in parallel and achieve high performance. Also, the temporary results from each processor can be sent to the following processor directly and do not need to be stored in a large global memory, so less memory

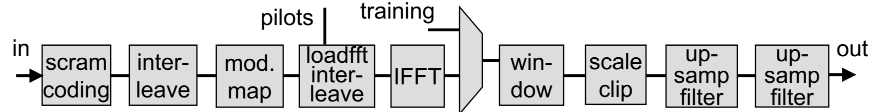

Figure 2.3: IEEE 802.11a/g wireless LAN (54 Mbps, 5/2.4 GHz) baseband transmit path

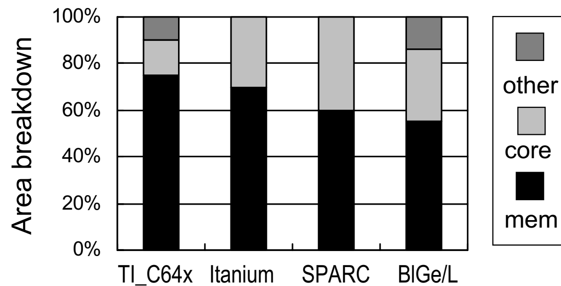

Figure 2.4: Area breakdown of four modern processors; memory occupying most of the area

is necessary compared to the traditional method.

Task level parallelism is widely available in many DSP applications. Figure 2.3 shows an example of a modern complex application that exhibits abundant task-level parallelism—the transmit chain of an IEEE 802.11a/g wireless LAN transmitter. It contains more than 10 tasks, and each of them can be directly mapped to separate processors to take advantage of the available task level parallelism.

2.1.2 Memory requirements of the targeted tasks

With an ever increasing number of transistors possible per die, modern programmable processors typically use not only an increasing amount of on-chip memory, but also an increasing percentage of die area for memory. Figure 2.4 shows the area breakdown of four modern processors [7, 13, 24, 25] with memories that occupy 55% to 75% of the processor's area. Large memories reduce the area available for execution units, consume significant power, and require larger delays per memory transaction. Therefore, architectures that minimize the need for memory and keep data near or within processing elements can increase area efficiency, performance, and energy efficiency.

A notable characteristic of the targeted DSP and embedded tasks is that many have very

Table 2.1: Memory requirements for common DSP tasks assuming a simple single-issue processor

Task	Instruction Mem requirement (words)	Data Mem requirement (words)
N-pt FIR	6	$2N$
8-pt DCT	40	16
8×8 2-D DCT	154	72
Conv. coding ($k = 7$)	29	14
Huffman encoder	200	350
N-point convolution	29	$2N$
64-point complex FFT	97	192
Bubble sort	20	1
N merge sort	50	N
Square root	62	15
Exponential	108	32

limited memory requirements compared to general-purpose tasks. The level of *required* memory must be differentiated from the amount of memory that *can* be used or *is* typically used to calculate these kernels. For example, an N-tap filter may be programmed using a vast amount of memory though the base kernel requires only $2N$ data words. Table 2.1 lists the actual amounts of instruction and data memory required for several DSP tasks commonly found in DSP applications. These numbers assume a simple single-issue fixed-point processor. The data show that several hundred words of memory are enough for many DSP and embedded tasks—far smaller than the 10 KBytes to 10 MBytes per processing element typically found in modern DSP processors. Reducing memory sizes can result in significant area and power savings.

In addition to the inherent small memory requirement of DSP applications, some architectural features such as address generators and embedded NOP instructions can also contribute to reduce the instruction memory requirement, which will be discussed further in Section 2.2.1.

2.1.3 Simple single issue datapath

The datapath, or execution unit, plays a key role in processor computation, and also occupies a considerable amount of chip area. Uniprocessor systems are shifting from single issue architectures to wide issue architectures in which multiple execution units are available to enhance system performance. For chip multiprocessor systems, there remains a question about the trade-off between using many small single-issue processors, versus larger but fewer wide-issue processors.

Wide-issue processors work well when instructions fetched during the same cycle are highly independent and can take full advantage of functional unit parallelism, but this is not always the case. Furthermore, a large wide-issue processor has a centralized controller, contains more complex wiring and control logic, and its area and power consumption increases faster than linearly along with the number of execution units. One model of area and power for processors with different issues derived by J. Oliver et al. [47] shows that a single 32-issue processor occupies more than 2 times the area and dissipates approximately 3 times the power of 32 single-issue processors.

Using multiple single-issue processors, such as AsAP, can usually achieve higher area efficiency and energy efficiency, and they perform particularly well on many DSP applications since those applications are often made up of complex components exhibiting task level parallelism so that tasks are easily spread across multiple processors.

Using relatively simple (and hence small) processors in many-core chips also has some other advantages. For example, it is easier to apply dynamic clock and voltage scaling on smaller processors to achieve higher energy efficiency; it allows fitting more processors into a single chip and can introduce more redundant processors which can provide flexible fault tolerance computation techniques; and smaller processors are easier to design and verify, etc.

Some disadvantages of using large numbers of simple processors include extra inter-processor communication overhead when the applications have complex communication requirement, which is discussed further in Sec. 5.1.4; and the system is less efficient if the application is not possible to partition into independent tasks.

2.1.4 GALS clocking style

A globally synchronous clock style is normally used in modern integrated circuits. But with the larger relative wire delays and larger parameter variations of deep submicron technologies, it has become increasingly difficult to design both large chips, and chips with high clock rates. Additionally, high speed global clocks consume a significant portion of power budgets in modern processors. For example, 1/4 of the total power dissipation in the recent 2-core Itanium [13] is consumed by clock distribution circuits and the final clock buffers. Also, the synchronous style lacks the flexibility to independently control the clock frequency among system sub-components to

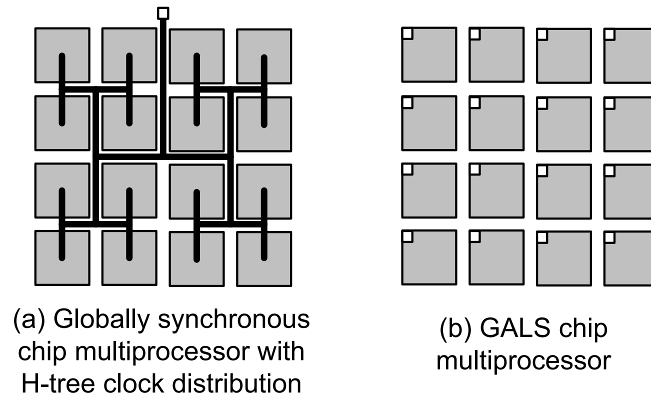

Figure 2.5: Block diagrams of (a) synchronous and (b) GALS tile-based chip multiprocessors; the small boxes in the right figure are the local oscillators for each processor

achieve increased energy efficiency. Furthermore, if using a synchronous clock style in multi-core systems, as shown in Fig. 2.5 (a), it has 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, which decreases performance.

The opposite clock style of globally synchronous—fully asynchronous in which there is no any timing reference exists—has the potential for speed and power improvements, but currently lacks EDA tool support, is difficult to design, and has large circuit overhead which reduces its efficiency.

The GALS [40] clocking style separates processing blocks such that each part is clocked by an independent clock domain. Its use enables the possibility of eliminating global clock distribution completely which brings power and design complexity benefits. Another significant benefit of GALS is the opportunity to easily and completely shut off a circuit block's clock (not just portions of the clock tree as with clock gating) when there is no work to do. Additionally, independent clock oscillators permit independent clock frequency scaling, which can dramatically reduce power dissipation in combination with supply voltage scaling [48]. Furthermore, when using the GALS clocking style in multi-core systems as shown in Fig. 2.5 (b), scalability is dramatically increased and the physical design flow is greatly simplified, since only a single processor must be designed and the entire chip can be generated easily by duplicating a single processor design.

2.1.5 Wires and on chip communication

As discussed in Section 1.1.2, wires are introducing greater delay, power consumption, and other impacts to chips, and the traditional communication methods within chips such as global bus has met considerable challenges. Global chip wires will dramatically limit performance in future fabrication technologies if not properly addressed since their delay is roughly constant with technology scaling—which leads to an increasing percentage of clock cycle time. A number of architectures such as systolic [49], RAW [35] and TRIPS [50] have specifically addressed this concern and have recommended using tile-based architectures. Therefore, architectures that enable the elimination of long high-speed wires will likely be easier to design and may operate at higher clock rates [16].

There are several methods to avoid global wires. Networks on chip (NoC) [51] treat different modules in a chip as different nodes in a network and use routing techniques to transfer data. Currently, most chip multiprocessors target broad general purpose applications and use complex inter-processor communication strategies. For example, RAW [35] uses a separate complete processor to provide powerful static routing and dynamic routing functions which is similar with NoC, BlueGene/L [25] uses a torus network and a collective network to handle inter-processor communication, and Niagara [26] uses a crossbar to connect 8 cores and memories. These methods provide flexible communication abilities, but consume a significant portion of the area and power in communication circuits.

Another method is local communication, where each processor communicates only to processors within a local domain. One of the simplest examples is nearest neighbor communication, where each processor directly connects and communicates only to immediately adjacent processors and the long distance communication is accomplished by software in intermediate processors. Most DSP applications, especially those that are stream-like [52], have specific regular features and make it possible to use a simple nearest neighbor communication scheme to achieve high area and energy efficiency, without a large performance loss. As can be seen from several popular industry-standard DSP benchmarks such as TI [53], BDTI [54], and EMBC [55], the most common tasks include FIR and IIR filtering, vector operations, the Fast Fourier Transform (FFT), and various control and data manipulation functions. These tasks can normally be *linearly pipelined*, as shown in Fig. 2.6 (a) and

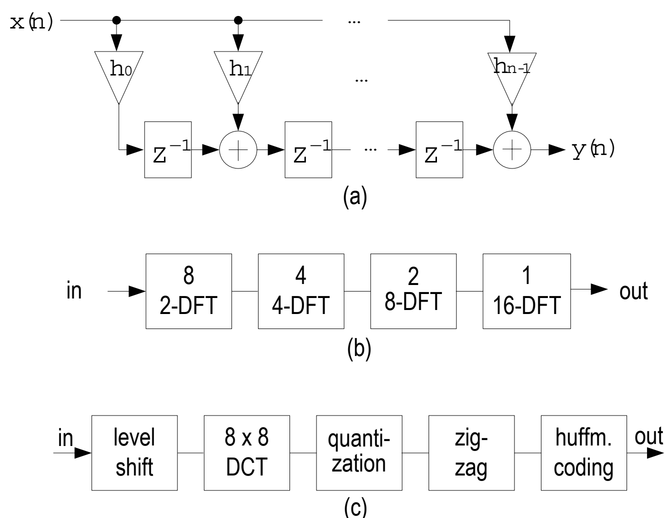

Figure 2.6: Some *linear pipelined* algorithm models for DSP tasks and applications; (a) Transposed direct FIR filter, (b) 16-pt FFT, (c) JPEG encoder

(b), and the result from one stage can be pumped directly to the next stage without complex global communication. Complete applications containing multiple DSP tasks also have similar features, as examples shown in Fig. 2.6 (c) and Fig. 2.3 for the JPEG encoder and the 802.11a/g baseband transmitter. All these examples can be handled efficiently by nearest neighbor inter-processor communication.

The greatest challenge when using nearest-neighbor interconnects is efficiently mapping applications that exhibit significant long-distance communication. The communication architecture will be investigated further in Chapter 3, to achieve communication ability close to NoC with cost close to the nearest neighbor interconnect.

2.2 The AsAP processor system

Figure 2.7 shows a block diagram of an AsAP processor and the fabricated processing array. Each processor is a simple single-issue processor, and contains: a local clock oscillator; two dual-clock FIFOs to provide communication with other processor cores; and a simple 16-bit CPU including ALU, MAC, and control logic. Each processor contains a 64-word 32-bit instruction memory and a 128-word 16-bit data memory. They also contain static and dynamic configuration logic to provide configurable functions such as addressing modes and interconnections with other

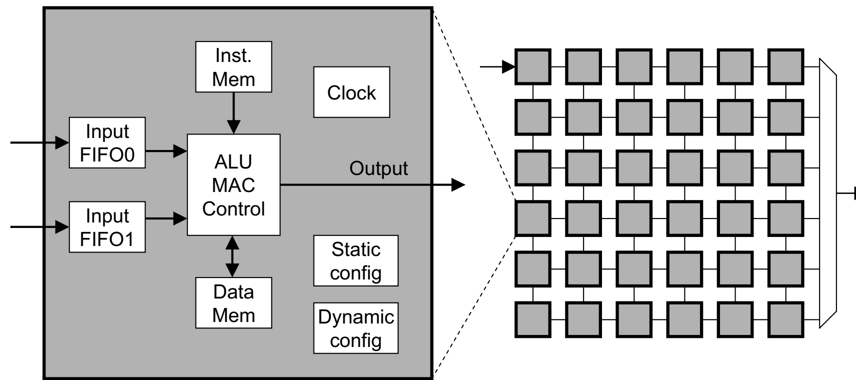

Figure 2.7: Block diagram of an AsAP processor

processors. Each processor can receive data from any two neighbors and can send data to any combination of its four neighbors. Each processor contains two input ports because it meshes well with the data flow graphs of the applications have studied. Clearly, two or more input ports are required to support graph fan-in and a third input port was found not frequently used. AsAP supports 54 RISC style instructions. It utilizes a memory-to-memory architecture with no register file. During the design phase, hardware was added only when it significantly increased performance and/or energy-efficiency for our benchmarks.

2.2.1 Single AsAP processor design

Pipelining and datapath

Each AsAP processor has a nine stage pipeline as shown in Fig. 2.8. The *IFetch* stage fetches instructions according to the program counter (PC). No branch prediction circuits are implemented. All control signals are generated in the *Decode* stage and pipelined appropriately. The *Mem Read* and *Src Select* stages fetch data from the data memory (Dmem), immediate field, the asynchronous FIFO interface from other processors, dynamic configuration memory (DCMem), the ACC accumulator register, or ALU/MAC forwarding logic. The execution stages occupy three cycles, and bypass logic is used for the ALU and MAC to alleviate data hazard pipeline penalties. The *Result Select* and *Write Back* stages select results from the ALU or MAC unit, and write the result to data memory, DC memory, or neighboring processors. To simplify pre-tapeout verification,

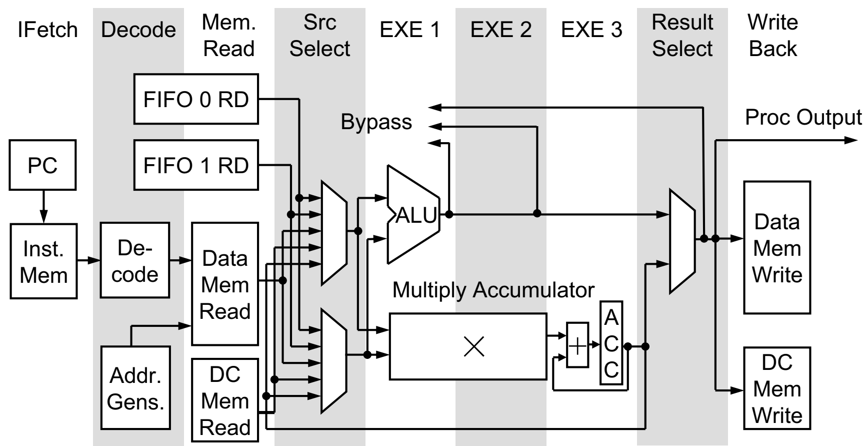

Figure 2.8: AsAP 9-stage pipeline

pipeline interlocks are not implemented in hardware, and all code is scheduled prior to execution by the programmer or compiler.

The MAC unit is divided into three stages to enable a high clock rate as well as the capability of issuing MAC and multiply instructions every cycle. The first stage generates the partial products of the 16×16 multiplier. The second stage uses carry-save adders to compress the partial products into a single 32-bit carry-save output. The final stage contains a 40-bit adder to add the results from the second stage to the 40-bit accumulator register (ACC). Because the ACC is normally read infrequently, only the least-significant 16 bits of the ACC are readable. More significant ACC bits are read by shifting those bits into the 16 LSBs. This simplification reduces hardware and relaxes timing in the final MAC unit stage which is the block's critical pipeline stage.

Local oscillator

Each processor has its own digitally programmable clock oscillator which provides the clock to each processor. There are no PLLs (phase lock loop), DLLs (delay lock loop), or global frequency or phase-related signals, and the system is fully GALS. While impressive low clock skew designs have been achieved at multi-GHz clock rates, the effort expended in clock tree management and layout is considerable [56]. Placing a clock oscillator inside each processor reduces the size of the clock tree circuit to a fraction of a mm^2—the size of a processing element. Large systems can be

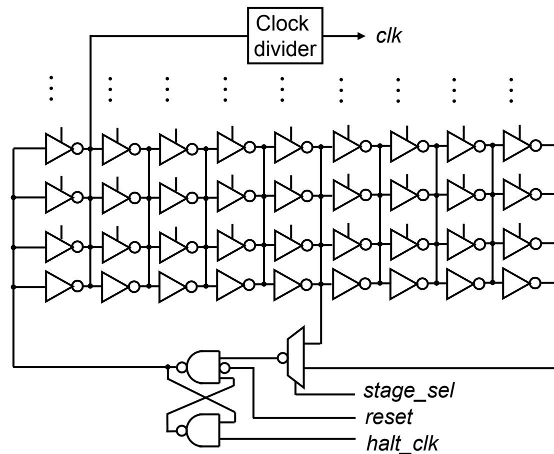

Figure 2.9: Programmable clock oscillator: an inverter ring with configurable tri-state inverters, ring size and frequency divider

made with arrays of processing elements with no change whatsoever to clock trees (that are wholly contained within processing elements), simplifying overall design complexity and scalability.

The oscillator is an enhanced ring oscillator as shown in Fig. 2.9. It is built entirely with standard cells and occupies only about 0.5% of the processor's area. Three methods are used to configure the frequency of the oscillator. First, the ring size can be configured to 5 or 9 stages using the configuration signal *stage_sel*. Second, seven tri-state inverters are connected in parallel with each inverter. When a tri-state inverter is turned on, that stage's current drive increases, and the ring's frequency increases [57]. Third, a clock divider at the output divides the clock from 1 to 128 times. The *halt_clk* signal and SR latch allow the oscillator to cleanly halt when the processor stalls without any partial clock pulses.

The oscillator has 2 ring-size-settings $\times$ 32768 tri-state-buffer-settings $\times$ 8 clock-divider-settings = 524,288 frequency settings, and measured results from the fabricated chip show that its frequency range is 1.66 MHz to 702 MHz over all possible configurations. In the range of 1.66 to 500 MHz, approximately 99% of the frequency gaps are smaller than 0.01 MHz, and the largest gap is 0.06 MHz. Despite the fact layout for all processors is exactly the same, variations largely due to the fabrication process cause different processors on the same die to perform differently than others. As Fig. 2.10 shows (data was measured by group member D. Truong), at the same configuration setting on the same chip the oscillator located in the bottom right processor has a frequency greater

506.7	498.5	497.4	505.9	510.1	520.9
507.6	506.2	498.9	506.8	510.6	520.8
508.4	507.2	503.7	507.3	511.2	517.1
514.7	509.9	511.6	512.1	513.9	515.5
519.3	521.0	515.8	519.3	518.2	531.5
536.2	535.2	538.6	532.4	537.1	541.2

Figure 2.10: Physical distribution of measured oscillator frequencies across different processors with the same configuration. Data are given in MHz with contour lines from 500 MHz to 540 MHz in 5 MHz steps.

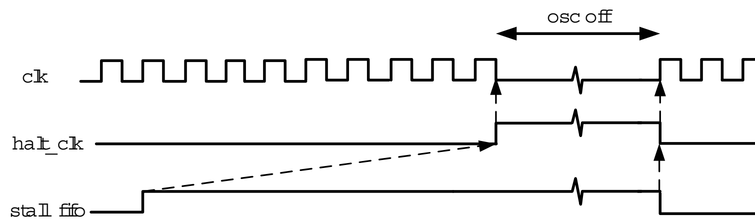

Figure 2.11: Example waveform of clock halting and restarting

than 540 MHz, while several oscillators in the top row have frequencies less than 500 MHz, the frequency difference is about 10%.

Processors stall when they try to read data from an empty FIFO or write data to a full FIFO. During these situations, the clock oscillator can be halted so that the processor consumes no power whatsoever except leakage. Figure 2.11 shows an example waveform for clock halting and restarting. Signal *stall_fifo* is asserted when FIFO access is stalled due to either an empty input or full output condition. After a nine clock cycle period, during which the processor's pipeline is flushed, the signal *halt_clk* goes high which halts the clock oscillator. The signal *stall_fifo* returns low when the cause of the stall has been resolved; then *halt_clk* restarts the oscillator in less than

Table 2.2: AsAP 32-bit instruction types and fields

Instruction type	6 bits	8 bits	8 bits	8 bits	2 bits
General	opcode	dest	src1	src2	NOP
Immediate	opcode	dest	immediate		NOP
Branch	opcode	–	–	target	NOP

Table 2.3: Classes of the 54 supported instructions

Instruction class	Number of instructions
Addition	7
Subtraction	7
Logic	11
Shift	4
Multiply	2
Multiply-accumulate	6
Branch	13
Miscellaneous	4

one clock period. Using this method, power is reduced by 53% and 65% for a JPEG encoder and a 802.11a/g transmitter application respectively.

Instruction set

AsAP supports 54 32-bit instructions with saturation modes, and supports three broad instruction formats. A summary of the 54 instructions is given in Tables 2.2 and 2.3. *General* instructions select two operands from memories, accumulator, FIFOs, and three ALU bypass routes; execute two operands by addition, subtraction, logic execution, multiply, or multiply-accumulate, and then select one destination from memories, accumulator and output ports. *Immediate* instructions receive input from a 16-bit immediate field. *Branch* instructions include a number of conditional and unconditional branch functions.

Seven addition instructions include: add and save low 16 bits; add and save high 16 bits; add with saturation; add with carry; add with carry and save high 16 bits; add with carry and with saturation; add by 1. Similar instructions exist for subtraction. The eleven logic instructions include: NOT, AND, NAND, OR, NOR, XOR, XNOR, MOVE, ANDWORD, ORWORD, and XORWORD. Besides the common conditional branch instructions which determine the destinations according to the conditional flags (such as negative, zero, carry and overflow), the instruction set also supports

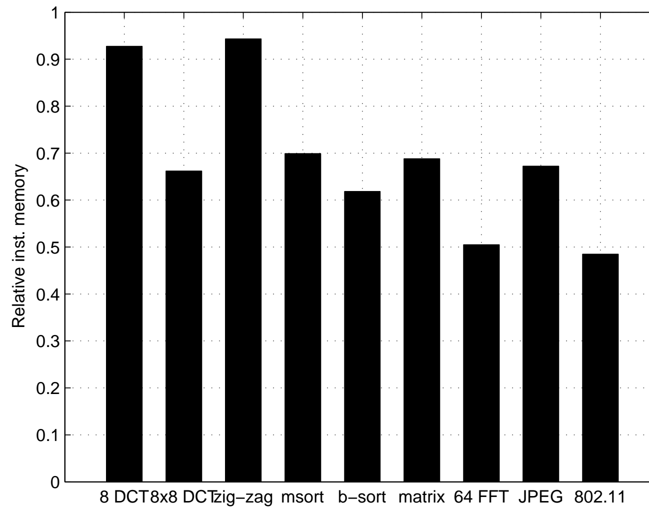

Figure 2.12: Relative instruction memory cost by using *embedded NOP* instruction

branching when the FIFO is empty or full. The four miscellaneous instructions include a 16-bit bit-reverse instruction (for FFT addressing), an accumulator shift, a 16-bit move intermediate, and halt.

Two bits in each instruction define how many NOP operations (from 0 to 3) should follow after instruction processing, which allows inserting NOPs to avoid pipeline hazards without requiring additional NOP instructions and helps reduce instruction memory requirements dramatically. Figure 2.12 shows that instruction memory requirements can be reduced by approximately 30% for 9 applications. Below is one example to show how to use this instruction format where the data memory with address 1 is incremented by 1 and the result is sent to data memory address 2 and output port.

```
Add Dmem 2 Dmem 1 #1 NOP3
Add Obuf       Dmem 2
```

Other than a bit-reverse instruction and a bit-reverse mode in the address generators which is useful for the calculation of the Fast Fourier Transform (FFT), no algorithm-specific instructions or hardware are used. While single-purpose hardware can greatly speed computation for specific algorithms, it can prove detrimental to the performance of a complex multi-algorithmic system and

Table 2.4: Data fetch addressing modes

Addressing mode	Example	Meaning
Direct	`Move Obuf Dmem 0`	`Obuf` ← `Dmem[0]`
Address pointer	`Move Obuf aptr0`	`Obuf` ← `Dmem[DCMem]`
Address generator	`Move Obuf ag0`	`Obuf` ← `Dmem[generator]`
Short immediate	`Add Obuf #3 #3`	`Obuf` ← `3+3`
Long immediate	`Move Obuf #256`	`Obuf` ← `256`
DCMem	`Move Obuf DCMem 0`	`Obuf` ← `DCMem[0]`
Bypassing	`Move Obuf regbp1`	`Obuf` ← `first bypass`
FIFOs	`Move Obuf Ibuf0`	`Obuf` ← `FIFO 0`
ACC	`Move Obuf Acc`	`Obuf` ← `ACC[15:0]`

limits performance for future presently-unknown algorithms—which is one of the key domains for programmable processors.

Data addressing

AsAP processors fetch data at pipeline stage *Mem Read*, using the addressing modes listed in Table 2.4. Three methods are supported to address data memory. *Direct* memory addressing uses immediate data as the address to access static memory locations; four *address pointers* access memory according to the value in special registers located in DCMem; and four *address generators* provide more flexible addressing such as automatic increment and decrement, with special-purpose hardware to accelerate many tasks. In addition to the data memory, AsAP processors can also fetch data from another 6 locations: 1) short immediate data (6 bits) can be used in dual-source instructions, 2) long immediate data (16 bits) can be used in the move immediate instruction, 3) the DCMem's configuration information, 4) three bypass paths from the ALU and MAC units to accelerate execution, 5) the two processor input FIFOs, and 6) the low 16 bits of the accumulator register.

Address generators help reduce the required instruction memory for applications since they can handle many complex addressing functions without any additional instructions. The upper figure of Fig. 2.13 shows the estimated relative instruction cost for a system using three addressing modes to fulfill the same functions. Compared to systems primarily using direct memory addressing and address pointers, the system containing address generators reduces the number of required instructions by 60% and 13% respectively. Also, using address generators can increase system

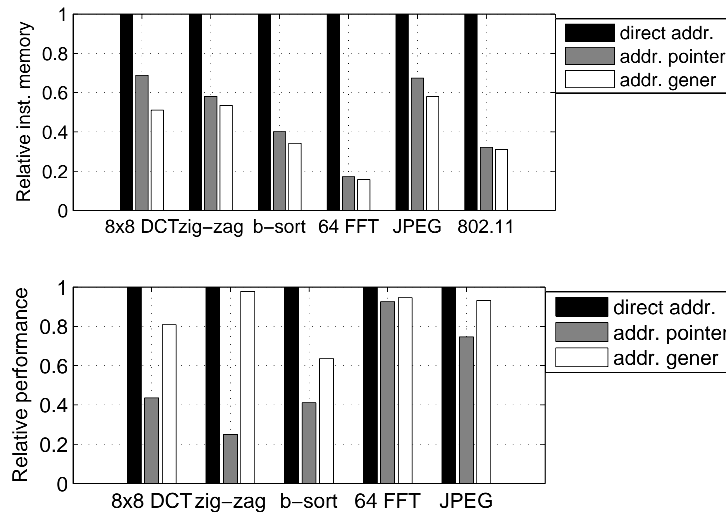

Figure 2.13: Comparison of relative instruction memory cost and system performance for three different addressing modes. Comparisons are made against the *direct address* case which uses straight line code with pre-calculated addresses only.

performance. As shown in the lower figure of Fig. 2.13, it comes within 85% of the performance of a system using direct addressing with pre-calculated addresses, and approximately 2 times higher performance compared to a system using address pointers alone.

2.2.2 Inter-processor communication — Reconfigurable 2-D mesh network

The AsAP architecture connects processors via a configurable 2-dimensional mesh as shown in Fig. 2.14. To maintain link communication at full clock rates, inter-processor connections are made to nearest-neighbor processors only. Each processor has two asynchronous input data ports and can connect each port to any of its four nearest neighboring processors. The input connections of each processor are normally defined during the configuration sequence after powerup. The output port connections can be changed among any combination of the four neighboring processors at any time through software. Input ports are read and output ports written through reserved program variables and inter-processor timing is in fact invisible to programs without explicit software synchronization.

A number of architectures including wavefront [49], RAW [35], and TRIPS [50], have specifically addressed this concern and have demonstrated the advantages of a tile-based architec-

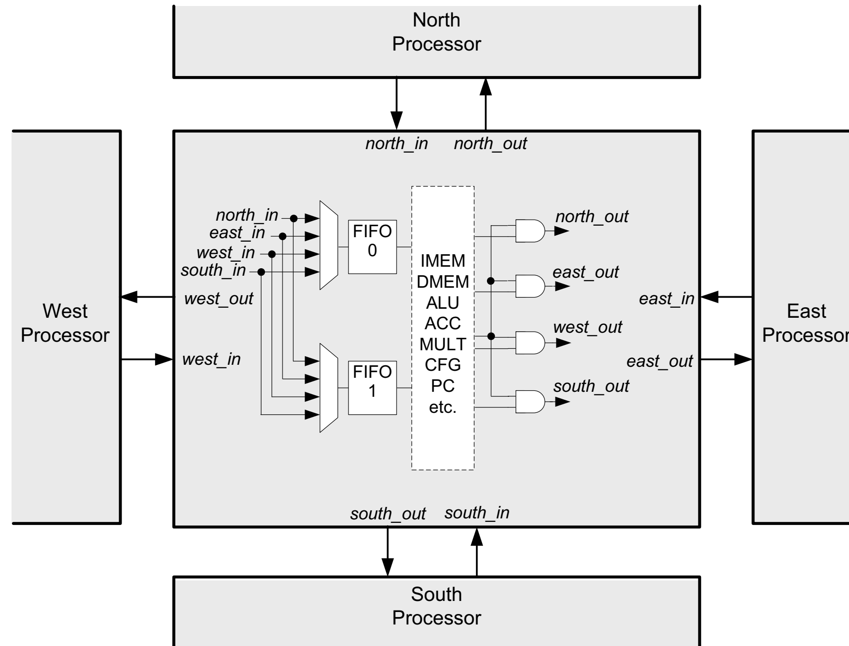

Figure 2.14: Nearest neighbor inter-processor communication diagram

ture. AsAP's nearest neighbor connections result in no high-speed wires with a length greater than the linear dimension of a processing element. The inter-processor delay decreases with advancing fabrication technologies and allows clock rates to easily scale upward. Longer distance data transfers in AsAP are handled by routing through intermediary processors or by "folding" the application's data flow graph such that communicating processing elements are placed adjacent or near each other.

Source synchronization for GALS systems

GALS systems introduce modules with different clock domains and the communication between those modules requires special concerns. The methods can be classified into two categories. The first method is asynchronous handshake [58] as shown in Fig. 2.15 (a). The source sends a *request* signal and one single data at each transaction and can start a new data transfer till it receives the *acknowledge* signal from the destination. A corresponding latency exists for each data transfer in this method. In order to sustain higher throughput, coarse grain flow control or *source synchronous* method can be used as shown in Fig. 2.15 (b), where the clock of the source processor

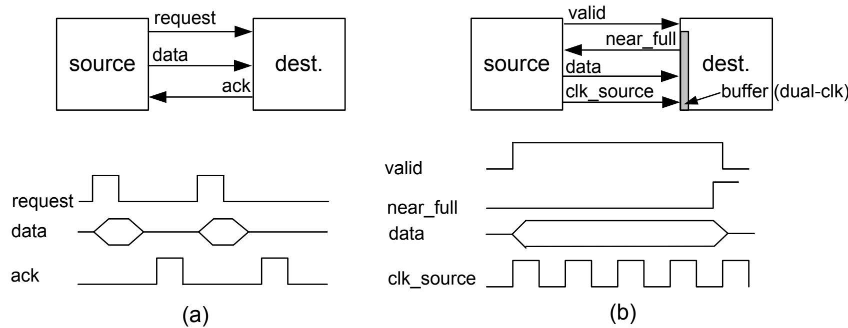

Figure 2.15: Two strategies for communication in GALS systems. (a) asynchronous handshake which requires more than one clock cycle for each transaction, and (b) source synchronous flow control which can sustain one transaction per clock cycle.

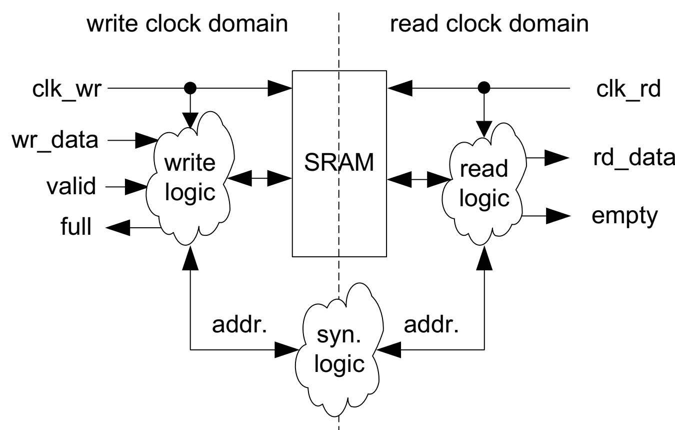

Figure 2.16: Block diagram of the dual-clock FIFO used for asynchronous boundary communication

travels along with the signals to control the writing into a buffer. Here the data can be transmitted in each clock cycle as long as the buffer is not full. This technique generally requires larger buffers in the destination and it also needs to support reads and writes in different clock domains since its writing is controlled by the source processor while its reading is controlled by the destination processor. Only the second approach is considered 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. The throughput of the asynchronous boundary communication can achieve 1 data word per cycle when the FIFO is not full or empty.

The reliable transfer of data across unrelated asynchronous clock domains is accom-

plished by mixed-clock-domain FIFOs. The dual-clock FIFOs read and write in fully independent clock domains. A block diagram of the FIFO's major components is shown in Fig. 2.16. The FIFO's write clock and write data are supplied in a source-synchronous fashion by the upstream processor and the FIFO's read clock is supplied by the downstream processor—which is the host for the dual-clock FIFO in AsAP. The read and write addresses are transferred across the asynchronous interface, which is used to decide whether the FIFO is full or empty. In order to avoid changing multiple values at the same time across the asynchronous interface, the addresses are gray coded when transferred across the clock domain boundary [41].

Configurable synchronization registers are inserted in the asynchronous interface to alleviate metastability. Although it can not drive the *mean time-to-failure* (MTTF) to infinity, it can make it arbitrarily high [59]. The number of synchronization registers used is a trade-off 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 are used), and the read logic latency (1 cycle). In a relevant example [59], 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. The system performance (throughput) overhead due to the synchronization latency is always quite small, as will be discussed in Section 6.3.

2.3 Application implementations and software

2.3.1 Application implementations

Dividing applications into several tasks, coding each task independently, and mapping each task onto a different processor(s) is the method to program the described GALS chip multiprocessor. Partly due to the natural partitioning of applications by task-level parallelism, the programming is less challenging than first expected. This is borne out somewhat by data in Table 2.1 showing the sizes of common DSP tasks. The mapped applications include: an 8-point Discrete Cosine Transform (DCT) using 2 processors, an 8$\times$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 matrix mul-

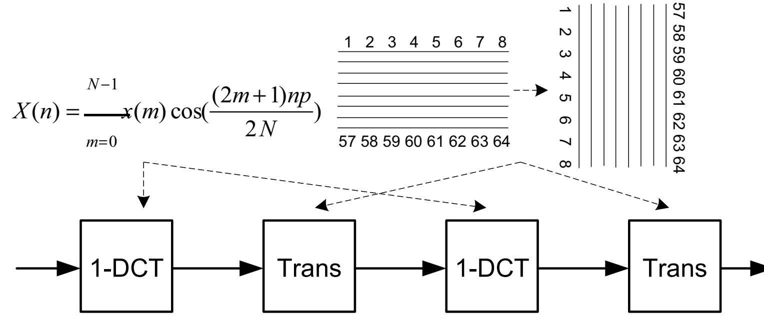

Figure 2.17: 8 x 8 DCT implementation using 4 processors

tiplier using 6 processors, a 64-point complex FFT using 8 processors, a JPEG encoder core using 9 processors, and an IEEE 802.11a/g wireless LAN transmitter using 22 processors [43]. 2-D DCT and JPEG are described in detail as two examples.

2-D DCT

An 8×8 DCT application is shown in Fig. 2.17. Equation 2.1 shows the algorithm for the 8×8 DCT and it can be further expressed and processed using two 1-dimensional DCTs as shown in equation 2.2; where $a(0) = 1/\sqrt{8}$ and $a(m) = \sqrt{2/8}$ for $1 \leq m \leq 7$. The first and third processors compute 1-dimensional 8-pt DCTs using an efficient algorithm [60]. The second and fourth processors perform row and column transposes of the data.

$$G_c(m,n) = a(m)a(n)\sum_{i=0}^{7}\sum_{k=0}^{7} g(i,k)cos\left[\frac{\pi(2k+1)n}{16}\right] cos\left[\frac{\pi(2i+1)m}{16}\right] \tag{2.1}$$

$$G_c(m,n) = a(m)\sum_{i=0}^{7}\left[a(n)\sum_{k=0}^{7} g(i,k)cos\frac{\pi(2k+1)n}{16}\right] cos\left[\frac{\pi(2i+1)m}{16}\right] \tag{2.2}$$

JPEG encoder

JPEG (Joint Photographic Experts Group) is a popular algorithm for still image compression [61] and Fig. 2.6 (c) shows its encoder block diagram. Level shift requires subtracting a fixed value from every sample value. Quantization divides the DCT result by specific value and rounding is required to get accurate result. Zig-zag reorders the 8 × 8 data block which groups the low frequency coefficients, bringing more zero-value data together and making the Huffman encoding

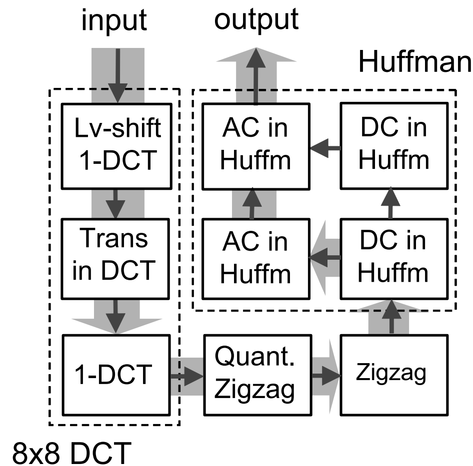

Figure 2.18: JPEG encoder core using 9 processors; thin arrows show all paths and wide arrows show the primary data flow.

Table 2.5: Computation load of the nine processors in JPEG encoder to execute an 8×8 data block

Processor No.	Function	Computation load (clock cycles)
1	Lv-shift, 1-DCT	408
2	Trans in DCT	204
3	1-DCT	408
4	Quant., Zigzag	652
5	Zigzag	134
6	DC in Huffm (below)	85
7	DC in Huffm (up)	140
8	AC in Huffm (below)	1423
9	AC in Huffm (up)	1390

more efficient. Huffman coding is an efficient variable length coding technique to represent information using fewer bits. Figure 2.18 shows a JPEG encoder core using nine processors. Three processors compute the level shift and an 8×8 DCT, and four processors implement a Huffman encoder. Processing each 8×8 block requires approximately 1400 clock cycles. Table 2.5 shown the computation load of the nine processors to execute an 8×8 data block.

2.3.2 Software

Besides using assembly language to program the processor, a high level language (which is called AsAP-C) and its corresponding compiler were developed to generate code for each indi-

vidual AsAP processor. AsAP-C contains most standard C language functions such as arithmetic calculations (addition, subtraction, multiplication, etc.), logic calculations (and, or, not, etc.), and control functions (while loops, for loops, etc.). A saturating integer type is defined to support DSP integer calculations which are commonly used in high level DSP languages [62]. Additionally, the language contains several functions specific for AsAP such as FIFO reads and direct inter-processor communication. Both inputs and outputs are mapped into the language through the reserved variable names: `Ibuf0`, `Ibuf1`, and `Obuf`. The software task also includes the mapping of processor graphs to the 2-D planar network. An area of interesting future work is tools for the automatic mapping of graphs to accommodate rapid programming and to recover from hardware faults and extreme variations in circuits, environment, and workload [63].

2.4 Related work

The related work is discussed in this section. Starting from the traditional DSP processors, then some pioneering multiprocessor systems and multi-core systems proposed/developed mainly in 1970's and 1980's are discussed; then some major modern multi-core processors (include those for general purpose and specific domain applications) are discussed and compared.

2.4.1 Traditional DSP processors

Programmable DSPs appeared in early 1980s for the high performance computation for DSP applications. In the early days, DSP processors were distinguished from general purpose processors by a couple of features [64]: such as higher arithmetic processing ability by integration of a hardware multiplier/accumulator and saturation circuity; and higher memory access bandwidth by multiple memory banks and sophisticated addressing mode generators.

When some new applications such as multimedia processing [65] emerged, both DSP processors and general purpose processors tried to enhance their features to make themselves suitable for those new applications. General purpose processors adopted some new technologies such as instruction level parallelism (superscalar); and data level parallelism (SIMD or MMX technology [66]). DSP processors also included some new architectures such as VLIW parallelism and hierarchical cache memory organization. As a result, the architectures of state-of-the-art DSPs and

Table 2.6: Major features of some commercial DSP processors

Company	Processor	Architecture type	Cache /Memory (KB)	Clock frequency (MHz)	Word width (bits)
Texas Instruments	C2000	single issue	13–295	40–150	32
Texas Instruments	C54x	single issue	24–280	50–300	16
Texas Instruments	C64x	8-way VLIW	1024–2048	500–1000	32
Analog Device	ADSP	single issue	8–48	75–160	16
Analog Device	SHARC	single issue	500–3000	150–400	32/40 floating pt.
Analog Device	TigerSHARC	4-way VLIW	500–3000	250–600	32 floating pt.
Freescale	DSP563xx	single issue	24–576	100–275	24
Freescale	MSC8144	quad core	10500	upto 1000	16
Freescale	MSC711x	6-way VLIW	80–448	upto 1000	16

general purpose processors become broadly similar with each other. A recent 600-MHz, 64 M transistor, eight-way VLIW DSP processor with 32 KB of level-1 caches and 1 MB of level-2 caches illustrates this point [7]. DSP processors of this type are well-suited for executing both numerically-intensive codes and larger more control-oriented general-purpose codes. Table 2.6 lists some features for a couple of major commercial DSPs [53, 67, 68] from Texas Instruments, Analog Device, and Freescale. There are some startup companies keep entering this area. For example, Stretch announced their product for video surveillance systems [69] recently.

Texas Instruments corporation reported their new C64x+ DSP in 2007 [70] which contains three DSP cores and was fabricated in 65 nm technology. Although the architecture is just simply putting three DSPs together and can categorize it in the traditional DSP processor domain; it illustrates the trend of DSP processors shifting from uni-core to multi-core architectures. The chip operates at 1.1 GHz, each core has 8 functional units and can perform up to 8000 MIPS or 8000 16b MMACs per second. The chip consumes 6 W and occupies 130 mm^2.

2.4.2 Pioneering multiprocessor systems and multi-core processors

Communication model: shared-memory and message passing

The idea of multiprocessor systems can be traced back to early 1970's, when a *shared-memory* multiprocessor architecture was proposed [71] in which processors exchange data through a global memory. In order to reduce the traffic between processors and global memory, each processor

normally has some cache embedded; which raises the issue of multiple copies of a single memory word being manipulated by multiple processors. This so called *cache coherence* issue significantly increases hardware complexity. Another major issue of the shared memory system is its limited scalability due to its centralized structure. Some work such as DASH [72] tried to improve the scalability of shared-memory systems by distributing the global memory to each processor node. Later another multiprocessor system platform using *message passing* communication was proposed [73] where processors exchange information by sending/receiving data between processors in a point-to-point fashion. Message passing architectures simplify hardware and also increase system scalability, but increase programming complexity. Some work such as FLASH [74] built systems to support both communication protocols in one hardware platform efficiently.

Interconnect topology

Under the two communication models, there are many interconnect topologies. The most basic topology is probably the *global bus* where all elements communicate through the global wires. The bus topology is simple, but it doesn't scale well since increasing the number of nodes on the bus can quickly introduce congestion and reduce communication performance; furthermore, the increasing delay/power of long wires in modern technology also makes the bus unattractive.

Crossbar is a network which is organized by a grid of switches to provide the connection from one group processors (memories) to another group of processors (memories), which makes it suitable for the shared memory architecture. It can provide more powerful communication capability than the global bus; but its complexity grows exponentially along with the number of nodes. A hybrid topology between bus and crossbar called multistage topology have been proposed where the network is organized by multiple stages and each stage provides part of the crossbar function; some examples includes *Omega* [75] and *perfect shuffle* [76].

As for the message passing communication, the most straightforward topology is probably the *completely-connected* network where each node has a direct link to every other node. It has the obvious disadvantage of the too large number of links. Other topologies suitable for the message passing communication include: 1-D linear array where each node is connected to its two neighbors; 2-D mesh array where each node is connected to its four neighbors; and 3-D mesh (cube) where each node is connected to its 6 neighbors. *2-D mesh* is attractive since it naturally fits into the 2-D

chip.

Some design cases

The transputer [77] is a popular parallel processor originally developed in the 1980s. It pioneers the philosophy of using multiple relatively simple processors to achieve high performance. The transputer is designed for a multiple processor board, where each transputer processor is a complete standalone system. It uses a bit serial channel for inter-processor communication which can support communication of different word lengths to save hardware, but with dramatically reduced communication speeds.

Systolic processor [78, 79] is another parallel computing architecture proposed in the 1980s. It pioneers in showing that some applications can be partitioned and connected by a couple of subtasks, and then these sub-tasks can be mapped to an array processor to achieve higher performance. Systolic processors contain synchronously-operating elements which send and receive data in a highly regular manner through a processor array. Due to its strict timing requirement for the data flow, the suitable applications for Systolic processor are quite limited. Some projects such as ChiP [80] (Configurable Highly Parallel Computer) can be categorized into Systolic systems, but it provides more flexible programmable interconnect structure so that the communication is not limited to the neighboring elements.

Wavefront processors [81, 49] were proposed right after Systolic processors, mainly by S.Y. Kung. In order to release the strict data flow timing requirement of Systolic processors, the Wavefront processor permits a *data-driven, self-timed* approach to array processing, and substitutes the requirement of correct *timing* by correct *sequencing* and thus significantly broadens the number of suitable applications. Furthermore, S.Y. Kung studied the VLSI device scaling trend and predicted the importance of the global (long) wiring [49], and suggested each processing element could execute asynchronous with each other, which pioneered the concept of GALS although he did not implement it. The work finished by U. Schmidt et. al. [82] was an exciting experiment of using wavefront array processor for image processing, unfortunately there has been no following up works. Some of the concepts of Wavefront array processors, such as GALS and data-driven communication, are similar with the techniques used in AsAP. But AsAP is distinguished from Wavefront processors broadly, such as its investigations on the granularity of each processing unit, the real

implementation of the chip and system, and more broad application mappings.

2.4.3 Modern multi-core systems

Although multi-core research in the 1980's showed great potential, they unfortunately did not dominate the market. The main reason is that simply increasing the processor clock frequency is much easier and cheaper than designing a brand new multi-core architecture, so there was not enough motivation for the shifting from uni-core processors to multi-core processors.

There is a clear revival in the multi-core processor research starting from the middle 1990's, in both academic and industry, because of the great challenges faced by the uni-core processors. An alternative solution is urgently required. Compared to the previous multiprocessors which were normally distributed in multiple chips, the modern single chip multi-core systems have some unique concerns such as the power consumption (which motivates AsAP's simple processing element decision), wiring issues (which results the distributed instead of centralized architecture, also motivates AsAP's GALS and mesh connection); and the faster and higher intra-chip communication bandwidth than inter-chip communication which affect the communication technology. Also, some researchers focus on the programming method to make the multiprocessor programming easier. Some modern multi-core processors will be discussed in this subsection, mainly from academic since they have more published information; and then compare the difference between AsAP to others.

PADDI-2 [83] was developed at UC Berkeley in the middle 1990's for DSP applications. It is a MIMD multiprocessor architecture consisting of arrays of simple elements (PEs) connected by a reconfigurable 2-level hierarchical network. The level-1 communication network consists of 6 data buses and is responsible for communication within a cluster of 4 PEs; and the level-2 communication network consists of 16 buses and handles traffic among the 12 clusters. PEs communicate with each other exclusively using data streams and control streams in a data-driven manner. The chip was fabricated in a 1 μm technology; it occupies 12 mm $\times$ 12 mm, operates at 50 MHz and provides 2.4 GOPS peak performance.

RAW [84] was developed at MIT starting from the late 1990's. Their initial target applications were stream-based multimedia computations although later they believe RAW can be a

universal solution for both general- and special-purpose applications. Motivated by the increasingly important wiring issues and high economic constraints of verifying new designs, they choose a tile-based architecture and use software to control inter-processor communication, wiring as well as instructions. One of the key features of their design is the static and dynamic switching circuity which allows an extremely low communication latency about 3 clock cycles for inter-processor communication. To achieve this goal is not free though, about half of the processor area is devoted to communication circuity; whether this cost is worthwhile is not easy to conclude and depends on specific applications. A 4 $\times$ 4 RAW array was implemented and reported in 2003 [35, 85] using IBM 0.18 μm CMOS technology. Each core contains 32 KB instruction memory, 32 KB Dcache and 64 KB SMem, and occupies 4 mm $\times$ 4 mm, and the entire chip is 18.2 mm $\times$ 18.2 mm including PADs. The chip core averages 18.2 Watts at 425 MHz.

Imagine [86] was developed at Stanford starting from the late 1990's, targeting stream style media processing. Motivated by the high parallelism and high computation locality of media applications, a streaming programming model and a corresponding Imagine architecture were developed. One of the main features of Imagine processor is that it provides a bandwidth hierarchy tailored to the demands of media applications which is organized by local register file, global register file and memory; and most of the computations can be finished in the registers thus improving performance as well as energy efficiency. The difference between global register file and cache is its high bandwidth which allows tens of words be fetched per cycle. A prototype Imagine processor was fabricated in a 0.15 μm CMOS technology [52]. It is an array containing 48 floating-point arithmetic units organized as eight SIMD clusters with a 6-wide VLIW processor per cluster, with 9.7 KBytes of local register file and 128 KBytes of global register file. Each cluster occupies 5.1 mm $\times$ 0.8 mm and the die size is 16 mm $\times$ 16 mm [87]. Imagine sustains 4.8 GFLOPS on QR decomposition while dissipating 7.42 Watts operating at 200 MHz. The Imagine processor was commercialized in 2007 in Stream Processors [88] in a 0.13 μm technology. The chip contains a 16-lane data-parallel unit with 5 ALUs per lane, two MIPS CPU cores, and I/Os. Each lane runs at 800 MHz at 1.0 V and occupies 3.2 mm^2. The peak performance is 128 GMACs with power consumption about 10 W.

Hydra [89] was proposed in Stanford in the late 1990's. From the high level view of its architecture, Hydra is similar to the traditional shared-memory multiprocessor systems: the chip

simply connects a couple of RISC processors (such as MIPS) and a L2 Cache together by a global bus; this centralized feature might limit its scalability potential. The novel contribution of Hydra is mainly on the programming: it simplifies the parallel programming as well as improves the performance by hardware supported thread-level speculation and memory renaming. Some of the technologies of Hydra were successfully used in the Sun Niagara SPARC processor [26]. The chip contains 8 symmetrical 4-way multithreaded cores sharing 3 MB of L2 cache. The die occupies 378 mm^2 in a 90 nm CMOS technology, operates at 1.2 GHz and consumes 63 Watts under 1.2 V. The second version of Niagara was reported in 2007 [31].

Pleiades Maia [90] was developed in UC Berkeley in the late 1990's; a reconfigurable DSP for wireless baseband digital signal processing. It combines an embedded microprocessor with an array of computational units of different granularities to achieve high energy efficiency; processing units are connected by a hierarchical reconfigurable interconnect network to provide flexibility; and handshake style GALS signaling was adopted to allow various modules to operate at different and dynamically varying rates. A prototype chip was implemented in a 0.25 μm CMOS process. The die size is 5.2 mm $\times$ 6.7 mm. The chip operates at 40 MHz on average and consumes 1.8 mW at a supply voltage 1 V; with energy efficiency between 10 and 100 MOPS/mW. The heterogeneous architecture is the main reason for Pleiades Maia' high energy efficiency; while it also limits its application domain.

Smart Memories [91] was proposed in Stanford in the early 2000's. It is a tile-based design made up of many processors, and has an ambitious goal to be a universal general design. In order to make it widely applicable, Smart Memories provides configurability to memories, wires and computational elements. This flexibility comes at a cost though. Compared to those programmable processors with specific domains such as Imagine processor, Smart Memories has about 50% performance degradation. The reconfigurable memory block of Smart Memories was fabricated in TSMC 0.18 μm technology [92] but the computation engine and the whole system has not been built. One reconfigurable memory with 512 $\times$ 36b SRAM occupies 0.598 mm^2 (in which 61% of the area is SRAM and the others are the logic providing the configuration), operates up to 1.1 GHz and consumes 141.2 mW.

TRIPS [93] was developed at UT Austin in the early 2000's, with the ambitious goal to build a system which provides high performance across a wide range of application types. In order

to provide configurability for both small and large-grain parallelism, TRIPS uses ultra-large cores—each core is a 16-wide-issue processor; the TRIPS can be configured to use ILP, TLP and DLP to be adapted to different application features and achieve high performance. It is a little questionable whether it is an achievable goal to build a universally energy efficient processor since the large core and additional configurability introduces non-negligible overhead. A prototype TRIPS chip was fabricated in a 0.13 μm technology [94]. The chip contains two cores and a separate on-chip network and occupies 336 mm^2. Its clock rate is 366 MHz and its power consumption is 36 W.

PipeRench [95] was developed at CMU in the early 2000's. The motivation of PipeRench is to efficiently execute numerically intensive applications, and its key feature is the dynamically reconfigurable datapath to match the features of applications. PipeRench is organized by sixteen stripes and each stripe consists of sixteen processing elements. A prototype PipeRench chip was fabricated in a 0.18 μm technology. Each processing element occupies 325 μm $\times$ 225 μm whose area is dominated by interconnect resources, and the whole die area is 7.3 mm $\times$ 7.6 mm. The chip can run at 120 MHz and executes a 40 tap 16-bit FIR filter at 41.8 million samples per second (MSPS). Operating at 33 MHz, the FIR filter consumes 518 mW without virtualization and 675 mW with virtualization.

WaveScalar [96] was proposed in University of Washington in 2003 targeting general purpose applications. A proposed architecture is organized as 16 clusters and each cluster contains tens of processing elements. Each processing element contains 8 instruction buffers, and each four clusters contains a traditional L1 data cache. The processing elements communicate via a shared bus within a cluster, and the inter-cluster communication uses a dynamic routed on-chip network. More implementation prototypes were discussed in 2006 [97]. Compared to AsAP, WaveScalar has a hierarchical organization. Its single processing element has finer granularity compared to AsAP while its cluster is more coarser to AsAP. Furthermore, WaveScalar uses a dataflow programming model which eliminates the traditional program counter to increase the parallelism. WaveScalar was not fabricated but was implemented on a FPGA.

Synchroscalar [98] was proposed in UC Davis in 2004. It is a tile-based multi-core architecture designed for embedded DSP processing. It uses columns of processor tiles organized into statically-assigned frequency-voltage domains, and each column uses a SIMD controller. Each column uses a rationally related clock frequency and can provide some flexibility of adaptive clock

and voltage scaling for each domain to achieve higher energy efficiency; although it is not as flexible as the absolute GALS style used in AsAP. Synchroscalar uses partially synchronous design and global interconnect for the centralized controller. This design can be suitable for those systems with low-frequency and a limited number of tiles, but its scalability is likely difficult if the number of tiles is increased a lot or the clock frequency is scaled up. Synchroscalar has not been fabricated.

Intel 80-core was published in 2007 [37]. It is a network-on-chip (NoC) architecture containing 80 tiles arranged as a 10×8 2D mesh network. The chip uses scalable global mesochronous clocking, which allows for clock-phase-insensitive communication across tiles and synchronous operation within each tile. The chip was fabricated in a 65 nm technology, each tile occupies 3 mm^2 and the whole chip is 275 mm^2. The chip operates at 3.13 GHz at supply voltage 1 V and 4 GHz at 1.2 V, and achieves 1.0 TFLOPS (trillion floating point operations per second) and 1.28 TFLOPS respectively.

There are some other chip multiprocessors such as RaPiD [99] developed in University of Washington, Picochip's PC102 [100], NEC's IMAP-CE [101], Xelerated AB's NPU [102], CISCO's Metro [103], IBM/sony/toshiba's CELL [23], Intellasys's SEAforth proc [104], Mathstar's Arrix FPOA [105], RAPPORT's KC256 [106], Ambric's proc [107], CEA-LETI's FAUST [108], and Cavium networks' 16-core [109].

The following subsections will try to compare those parallel processors using tables instead of describing them separately.

2.4.4 Distinguishing multi-core processors

It is not easy to distinguish and categorize different projects/processors by simply comparing their area, speed, power without looking into them deeply. This section tries to analyze different multi-core processors starting from their objectives since that is the key to motivate their specific designs and features.

Table 2.7 summarizes the objectives of some multi-core processors and their one or two most important features. The first three are pioneering multiprocessor systems proposed/developed before 1990's and the others are modern multi-core processors. Some processors such as Hydra and WaveScalar target to simplify the programming and their architectures are relatively similar

Table 2.7: Comparison of the objectives and distinguished key features of selected parallel processors

Processor	**Targets/objectives**	**Distinguished features**
Transputer [77]	high performance multiprocessor	bit serial channel
Systolic [78]	high performance array processing	regular communication pattern
Wavefront [49]	high performance array processing	data-driven, selftime execution
Hydra [89]	program shared-memory systems	thread speculation, memory renaming
WaveScalar [96]	program dataflow systems	memory ordering
RAW [84]	a universal system	complex static/dynamic route
TRIPS [93]	a universal system	large wide issue configurable core
Smart Memories [91]	a universal system	configurable memory system
PADDI-2 [83]	DSP applications	data-driven control
RaPiD [99]	DSP applications	reconfigurable pipelined datapath
PipeRench [95]	DSP applications	dynamically configurable datapath
Ambric's proc [107]	DSP applications	massively-parallel MIMD fabric
Synchroscalar [98]	some of DSP applications	rationally clocking and global comm.
CELL [23]	multimedia applications	one power proc and 8 synergistic procs
Imagine [86]	stream applications	hierarchical register file system
Pleiades Maia [90]	wireless applications	heterogeneous, reconfigurable connect
Picochip [100]	wireless applications	heterogeneous, reconfigurable
IMAP-CE [101]	video recognition	linear array of VLIW processor
Metro [103]	network applications	embedded DMA engine
Cavium 16-core [109]	network applications	RISC array + network coprocessor
NPU [102]	packet processing in network	linear array of 200 processors
FAUST [108]	telecom baseband applications	heterogeneous, GALS
Intel 80-core [37]	beyond 1 TOPS per chip	NoC, mesochronous clocking
AsAP	DSP applications	fine grain processor array, GALS

to traditional styles. Some processors such as RAW, TRIPS, and Smart Memories try to be as flexible and configurable as possible to make them suitable for wide applications; to achieve this goal, they introduce overhead such as complex interconnect, extremely large cores, and configurable memories. Some processors such as PADDI, RaPiD, PipeRench, Synchroscalar, Imagine, Pleiades, Picochip, and Metro target one domain of applications; they use more specific architectures such as heterogeneous processing elements, specific register file systems, and configurable datapaths to make the architecture efficient to the specific feature of those application domains.

Table 2.8 compares the listed parallel processors including the features of their processing elements, homogeneous/heterogeneous styles, chip sizes and power, etc.. Most parallel processors can be easily differentiated by their processing element architectures which can be categorized into

Table 2.8: Comparison of the selected parallel processors; the data is scaled to 0.18 μm technology and 1.8 V voltage supply; assuming a $1/s^2$ reduction in area, a s^2/v increase in clock rate, and a s/v^3 reduction in power consumption here s is the technology scaling factor and v is the voltage scaling factor; energy per operation is obtained from chip power divided by (issue width × element number × clock rate); the processors are sorted according to the area of each processing element; data for Synchroscalar, WaveScalar and Smart Memories are published estimates .

Processor	**Year**	**Homog. /Heterog.**	**Processing elements**	**Elem. size (mm^2)**	**Mem. size (KB)**	**Elem. issue width**	**Elem. num.**	**Chip size (mm^2)**	**Clock rate (MHz)**	**Chip power (W)**	**Energy per op (nJ/op)**
Arrix FPOA [105]	2006	heterogeneous	ALU/MAC/RF	$<$0.05	0.128	1	400	N/A	782	N/A	N/A
PADDI-2 [83]	1995	homogeneous	proc.	0.09	0.052	1	48	4.66	277	N/A	N/A
RaPiD [99]	1999	heterogeneous	ALU/MULT/RAM	0.65	N/A	1	16	12	277	N/A	N/A
AsAP	2006	homogeneous	proc.	0.66	0.64	1	36	32	530	1.28	0.07
Pleiades Maia [90]	2000	heterogeneous	ASICs/proc.	~0.8	N/A	N/A	22	18	138	0.014	N/A
Metro [103]	2005	homogeneous	proc.	0.96	N/A	1	188	621	180	67	1.98
PipeRench [95]	2002	homogeneous	execution cluster	1.17	0.256	16	16	55	33	0.675	0.08
Imagine [86]	2002	homogeneous	VLIW execution cluster	5.9	9.7	8	8	368	200	7.4	0.58
IMAP-CE [101]	2003	homogeneous	proc. cluster	~6	16	32	16	121	100	2.5–4	~0.06
Stream proc [88]	2007	homogeneous	VLIW execution cluster	6.1	16	10	16	~183	750	~42	~0.35
FAUST [108]	2007	heterogeneous	ASICs	~7	N/A	N/A	20	152	126	2.68	N/A
RAW [84]	2003	homogeneous	proc.	16	128	1	16	331	425	18.2	2.68
Intel 80-core [37]	2007	homogeneous	VLIW proc.	23	5	8	80	2108	735	206	0.44
Niagara1 [26]	2006	homogeneous	multithreaded proc.	50	24	4	8	1512	450	106	7.36
CELL [23]	2005	homogeneous	SIMD proc.	58	256	4	8	840	1500	~120	~2.5
Niagara2 [31]	2007	homogeneous	multithreaded proc.	~75	24	8	8	2622	300	133	6.93
TRIPS [94]	2007	homogeneous	wide-issue proc.	~200	32	16	2	642	264	69	8.2
KC256 [106]	2006	homogeneous	proc.	N/A	N/A	1	256	N/A	100	0.5	0.02
NPU [102]	2004	homogeneous	proc.	N/A	N/A	1	200	N/A	144	18	0.62
SEAforth-24 [104]	2006	homogeneous	proc	N/A	0.256	1	24	N/A	N/A	N/A	N/A
Cavium 16-core [109]	2006	homogeneous	proc.	N/A	40	2	16	N/A	470	61	4.05
Picochip PC102 [100]	2003	heterogeneous	procs./ASICs.	N/A	0.7 - 65	3	322	N/A	125	12	0.10
Ambric's proc [107]	2006	homogeneous	procs. cluster	N/A	8	8	45	N/A	240	N/A	N/A
Synchroscalar [98]	2004	homogeneous	proc.	3.5	32	4	16	60	400	1.84	0.07
Smart Mems [91]	2004	homogeneous	proc.	20	128	3	64	1300	N/A	N/A	N/A
WaveScalar [96]	2006	homogeneous	procs. cluster	80	2	16	16	1280	568	N/A	N/A

Table 2.9: Comparison of inter-element communication of selected parallel processors

Processor	**Inter-connect**	**Details**
Hydra [89]	bus	connect RISC processors and L2 Cache
Cavium 16-core [109]	bus	connect 16 processors and L2 Cache
CELL [23]	bus	high-bandwidth bus composed of four 128b data rings
Niagara [26]	Crossbar	connect between 8 cores and 4 L2 Caches
RaPiD [99]	1-D linear array	linearly connect reconfigurable pipelined datapath
PipeRench [95]	1-D linear array	linearly connect execution clusters
IMAP-CE [101]	1-D linear array	linearly connect 128 processors
NPU [102]	1-D linear array	linearly connect 200 processors
RAW [35]	2-D mesh	including static and dynamic route
Ambric's proc [107]	2-D mesh	configurable switches for distant comm.
TRIPS [50]	2-D mesh	dynamically routed, with ALU operand network
Smart Memories [91]	2-D mesh	packeted-based, dynamically-routed
FAUST [108]	2-D mesh	packeted-based, dynamically-routed
Intel 80-core [37]	2-D mesh	packeted-based, dynamically-routed
Pleiades [90]	hierarchical	2-D mesh for both local and global
PADDI-2 [83]	hierarchical	bus for local and crossbar for global
Imagine [86]	hierarchical	switch for both local and global
WaveScalar [96]	hierarchical	bus for local and 2-D mesh for global
Picochip [100]	hierarchical	special picobus for both local and global
Metro [103]	hierarchical	16 clusters each with 12 processing elements
Synchroscalar [98]	hierarchical	4 SIMD processors, connected by bus
AsAP	2D mesh	statically configurable

three broad types—heterogeneous, multiple execution units (often similar to classic SIMD), and multiple processors (MIMD).

A heterogeneous style such as the one used by Pleiades and Picochip makes the system efficient for specific applications, but results in a non-regular layout and has difficulty scaling.

Some projects such as Imagine, RaPiD, and PipeRench use multiple execution units to improve their performance. Strictly speaking, they can be categorized to parallel processing architectures but not multi-core systems since they have a centralized instruction controller. These architectures normally use hierarchical structures: combining some processing element into groups (called *cluster* in Imagine and *stripes* in PipeRench) and then those groups are organized into chip.

Quite a lot of systems can be categorized into MIMD processors, and AsAP can be distinguished from them by its small processing element granularity alone. One of the main reasons for others' increased processor granularity is because they target a wider range of applications including those which require large data working sets.

Table 2.9 compares the inter-processor communication network. Different processors choose their network for their target applications: from the simple bus, crossbar to 2-D mesh (including those called network-on-chip structure); and some processors use hierarchical network to treat local and long distance communication differently.

In terms of the clocking style, most other projects use a totally synchronous clocking style. Pleiades and FAUST uses GALS clocking style, but they use handshaking scheme to handle the asynchronous boundaries, it is quite different from the source-synchronous interprocessor communication style used in AsAP which is able to sustain a full-rate communication of one word per cycle. The GALS processing array by Ambric was presented but no details were given by regarding its operation. Intel 80-core employs mesochronous clocking where each processor has the same clock frequency while the clock phase can be different. Synchroscalar uses rationally related clocking style to achieve part of the clock/voltage scaling benefit; but not as flexible as GALS.

Partly because of these architectural level difference, these processors target different application domains. For example, Pleiades and Picochip target wireless applications; Imagine targets stream applications; RaPiD, PipeRench, and Synchroscalar target DSP applications; and TRIPS, Smart Memories, and RAW target all kinds of applications.

The key features of the AsAP processor—small granularity of each processor whit small memory and simple processing unit, GALS clocking style, and reconfigurable nearest-neighbor mesh network—distinguish it from other works.

2.5 Summary

The AsAP platform is well-suited for the computation in complex DSP workloads comprised of many DSP sub-tasks, as well as single highly-parallel computationally demanding tasks. By its very nature of having independent clock domains, very small processing elements, and short interconnects, it is highly energy-efficient and capable of high throughput.

Chapter 3

An Low-area Multi-link Interconnect Architecture

Inter-processor communication is an extremely important issue in chip multiprocessor systems, and it is investigated further in this chapter based on the discussion in Section 2.1.5. Both dynamic routing architectures and static nearest neighbor interconnect architectures—they are explained in section 3.0.1—achieve significant success in specific areas, but they have some limitations. Dynamic routing architecture is flexible, but normally sacrifices relatively high area and power overhead on the communication circuitry. The static nearest neighbor interconnect architecture reduces the area and power overhead significantly, but it sacrifices high latency for long distance communication. Architectures to obtain good trade offs between flexibility and cost are desired.

Communications within chip multiprocessors of many applications, especially many DSP algorithms, are often localized [110, 111]: most of the communications are nearest neighbors (or local) while a few are long distance. Motivated by this reality, we propose an *asymmetric* structure: treat the nearest neighbor communication and long distance communication differently, use more buffer resources for nearest neighbor connections, and use fewer buffer resources for long distance connections. Together with the relative simple static routing approach, this asymmetric architecture can achieve low area cost on the communication circuitry.

Under the static asymmetric architecture, there are a couple of design options available such as the number of input ports (buffers) for the processing core; and the number of links at

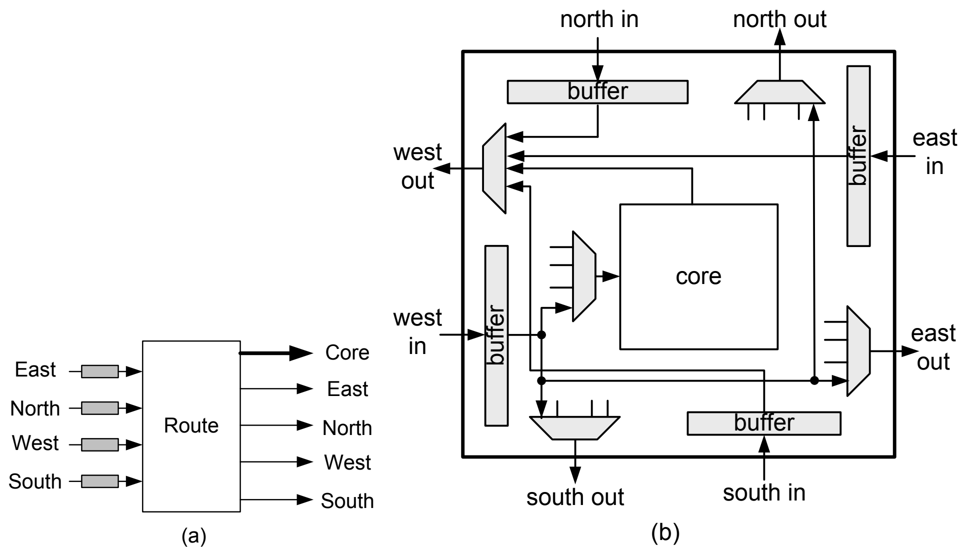

Figure 3.1: (a) An illustration of interprocessor communication in a 2-D mesh, and (b) a generalized communication routing architecture in which only signals related to the west edge are drawn.

each edge. The area, speed and performance of different design options are analyzed, and some conclusions based on the results are drawn. It is found that increasing the number of links between processors is helpful for the routing capability, but it will affect the processor area dramatically at some point. Two or three links are suitable if each processor in the chip utilizes a simple single issue architecture.

Moreover, the proposed architecture supports the GALS clocking style in long distance communication. After examining the characteristics of different approaches, the source synchronous method (it is explained in Section 2.2.2) is extended which transfers the clock with the signals along the entire path to control the data writing.

In this chapter, *nearest neighbor* of a processor represents the 4 nearest neighbors in 2D meshes although results will be similar for others such as the 6 nearest neighbors in 3D cubes; *processing core* is used for the execution unit of each processor and each input port of the processing core is connected to a buffer.

3.0.1 Background: traditional dynamic routing architecture

Figure 3.1 (a) shows the interprocessor communication in 2-D mesh chip multiprocessors using a routing architecture: the router block in each processor receives data from neighboring

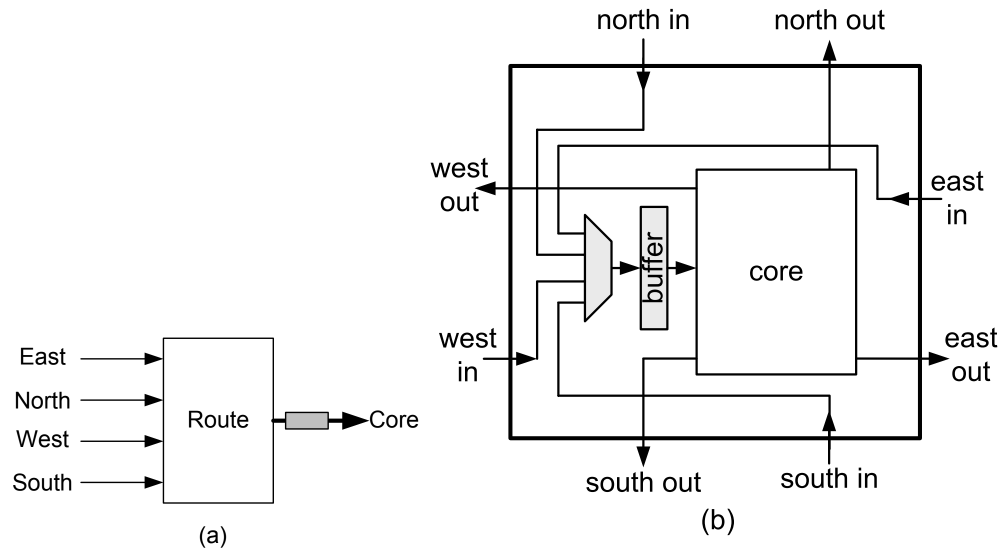

Figure 3.2: The concept and circuitry diagram of the nearest neighbor interconnect architecture. Data from four inputs are transfered only to the processing core to reduce the circuitry cost.

processors (east, north, west, and south) and then sends data to the processor core or to the other neighboring processors. Since the communication links are not always available due to slow data processing speed or link congestion, buffers are inserted at each input edge [112]. Figure 3.1 (b) shows a generalized diagram of the routing circuitry where only signals related to the west edge (*west in* and *west out*) are drawn. Input ports from each side feed data into a corresponding buffer, and the buffers supply data to the processor core or the other output ports. Each output port selects data from processor core and three input buffers. As the diagram shows, the communication logic includes four buffers and five muxes, and there is some control logic to support the communication flow control which is not drawn. Other implementations are possible, for example, each output port can also have a buffer or each input buffer can be split into multiple virtual channels [51] to reduce communication latency. The area of the communication circuitry is normally dominated by the buffers and the logic in four input/output ports are normally the same.

3.0.2 Background: static nearest neighbor interconnect

Most communications in the chip multiprocessors are localized or can be localized, which means the data traffic going into the processor core is much larger than the other paths. To minimize the communication circuit overhead, another interprocessor communication strategy—as used in

the first version of the AsAP processor—is to implement only the nearest neighbor interconnect logic, and long distance communication is fulfilled by extra MOVE instructions in the intermediate processors. Figure 3.2 (a) shows the concept of nearest neighbor interconnect and Fig. 3.2 (b) shows a circuitry diagram. All data from the input ports are transferred to the processing core, so instead of inserting a buffer at each input port, there is little effect on system performance by inserting buffer(s) only at the output(s) to the processing core, and statically configure the routing path. Comparing Fig. 3.1 and Fig. 3.2, the nearest neighbor interconnect reduces the number of buffers from four to one and the muxes for each output port are all avoided; resulting in more than four times smaller area. But clearly it has the limitation that long distance communication has a large latency.

3.1 Low-area interconnect architecture

This section proposes the statically configurable asymmetric architecture to achieve low area cost.

3.1.1 Asymmetric architecture

Asymmetric data traffic exists at the router's output ports universally

J. Hu et al. discovered the asymmetric data traffic in inter-processor communication and proposed using different buffer resources at input ports of routers to match the traffic [113]. One limitation of the asymmetric input ports architecture is that for different applications as well as for different processors of individual applications, the existing asymmetric data traffic on the router input ports is different, which makes the allocation of the buffer resources application specific. Considering the router's output ports instead of its input ports, most of the data from the input ports are delivered to the core and very few are to edges, which makes the asymmetric data traffic on the route's output more general and universal. Allocating asymmetric buffer resources at the output ports is applicable in a much wider range of applications, which is important since nowadays NoC architectures are used more widely than just specific domains.

Table 3.1 shows the data traffic of each processor for a JPEG encoder as shown in Fig. 2.18 which can demonstrate the different asymmetric data traffic on the input and output ports of routers. On the input ports, although each processor shows a clear asymmetric communication data traffic;

Table 3.1: Data traffic of router in a 9-processor JPEG encoder to process one 8×8 block. Assuming an architecture with four-inputs and five-outputs like Fig. 3.1. 80% of the data from inputs are delivered to the processing core which dominates the traffic at the output ports.

Processor No.	Network data words of input ports of router				Network data words of output ports of router				
	East	North	West	South	Core	East	North	West	South
1	0	64	0	0	64	0	0	0	0
2	0	64	0	0	64	0	0	0	0
3	0	64	0	0	64	0	0	0	0
4	0	0	64	0	64	64	0	0	0
5	0	0	96	0	64	0	32	0	0
6	0	0	0	64	1	0	0	63	0
7	0	0	0	3	3	0	0	0	0
8	63	0	0	0	63	0	0	0	0
9	4	0	0	252	256	0	0	0	0
Total	67	192	160	319	643	64	32	63	0
Relative overall (input)	9%	26%	22%	43%					
Relative overall (output)					**80%**	8%	4%	8%	0%

the major input direction for different processors are different which makes the overall traffic at the input ports quite uniform. On the output ports, however, each processor shows the similar asymmetric data traffic and overall about 80% data are delivered to the core. Similar results exist in other applications.

Proposed asymmetric architecture

An architecture is proposed which has asymmetric output ports as shown in Fig. 3.3 to achieve good trade offs between cost and flexibility. As shown in Fig. 3.3 (a), instead of equally distributing buffer resources to each output port, sufficiently large buffers are allocated to the processing core port, and the other ports use small buffers (one or several registers). Figure 3.3 (b) shows the circuitry diagram where the connections of the west signals *west in* and *west out* are completely drawn while the other edges are simplified. From the point view of the area and logic cost, this scheme is similar to the static nearest neighbor interconnect, as shown in Fig. 3.2, by adding a couple of registers and muxes. From the point view of the routing capability, this scheme is similar to the traditional dynamic routing architecture, as shown in Fig. 3.1, since reducing the buffers in ports for long distance communication does not significantly affect system performance when the

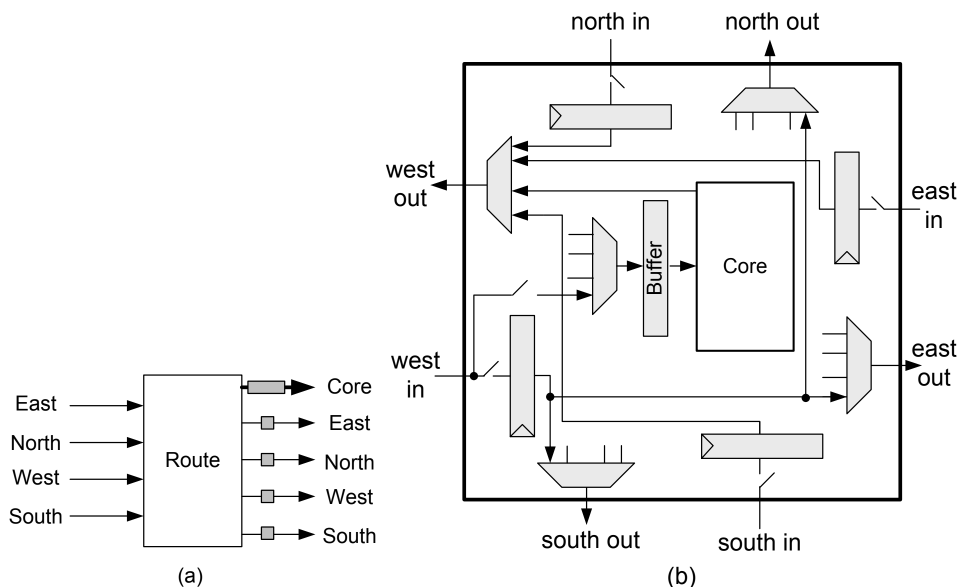

Figure 3.3: The concept and circuitry diagram of the proposed inter-processor communication architecture; it has the asymmetric buffer resources for the long distance interconnect and the local core interconnect

communication is localized. If using one large buffer for the processing core, the proposed architecture can save about 4 times area compared to the Fig. 3.1 architecture. If using two large buffers which will be discussed in Section 3.2.1, the area saving can still be about 2 times.

3.1.2 Theoretical analysis

This section gives a quantitative analysis about how the buffer size affects system performance. Also, another interesting question is that if the total buffer size is kept the same, what will be the best distribution of the buffer resources on channels with different traffic patterns.

Performance effect analysis

The buffer writing procedure is generally halted when the buffer is full, so that the buffer size has a strong effect on system performance. The quantitative effect of buffer size is highly dependent on the probability function of the data traffic in the buffer and Poisson distribution is one of the best models for the traffic behavior [114, 115]. Assuming the number of data in the port (channel) follows the Poisson function with average number of data λ, the probability of having k

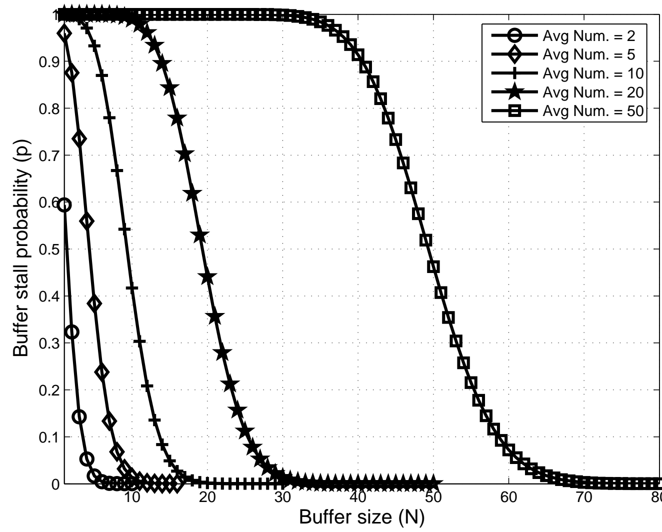

Figure 3.4: Buffer stall probability (p) along with buffer size (N) for channels with different average number of data modeled as Possion function

number of data is shown in Equation 3.1. Although the model is not quite possible to fit the reality exactly, it can provide some useful information.

$$f(k,\lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.1}$$

If using a buffer with size N for that channel, then the probability of buffer full follows Equation 3.2.

$$p(\lambda, N) = 1 - \sum_{k=0}^{N} \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.2}$$

Fig. 3.4 shows buffer stall probability (p) along with buffer size (N) for channels with different average number of data. When the buffer size is 2 times the average number of data in the channel, the buffer stall probability is close to zero. Table 3.2 shows the required buffer sizes for different target stall probabilities and different channels. These results show that the required buffer size is approximately linearly related to the traffic in the channel. It means that reducing the buffer size linearly along with traffic has nearly no effect on the performance, if the original buffer size is sufficiently large.

Table 3.2: The required buffer sizes for the defined stall probabilities and channels with different average numbers with Possion distribution function

Stall probability	Required buffer sizes with different average number of data in channels				
	2	5	10	20	50
10%	4	8	14	26	59
1%	6	11	18	31	67
0.1%	8	13	21	35	77

Optimal buffer resource distribution

It is also interesting to see the optimal buffer distribution if the total buffer resource is fixed as Q, and the average number of data in the processing core channel and other channels are λ and $n\lambda$ respectively (here n is the traffic ratio between the other channels and the processing core channel).

If allocating buffer size N to the processing core channel and mN to the other channels (here m is the buffer size ratio between the other channels and the processing core channel), the question is to minimize the total buffer full probability as shown in Equation 3.3:

$$\left(1 - \sum_{k=0}^{N} \frac{e^{-\lambda}\lambda^k}{k!}\right) + 4 \times \left(1 - \sum_{k=0}^{mN} \frac{e^{-n\lambda}(n\lambda)^k}{k!}\right) \tag{3.3}$$

And m and N need to meet Equation 3.4:

$$Q = N + 4mN \tag{3.4}$$

which means $N = \frac{Q}{1+4m}$. Replacing it into Equation 3.3 and the goal becomes to minimize Equation 3.5:

$$(1 - \sum_{k=0}^{\frac{Q}{1+4m}} \frac{e^{-\lambda}\lambda^k}{k!}) + 4 \times (1 - \sum_{k=0}^{\frac{mQ}{1+4m}} \frac{e^{-n\lambda}(\lambda)^k}{k!}) \tag{3.5}$$

Fig. 3.5 shows the overall system buffer stall probability at different buffer size ration (m) for different total buffer size (Q), assuming $\lambda = 50$ and $n\lambda = 5$ as a representative system. When there is sufficient buffer resource ($Q = 200$ in our case), the system has nearly no stall as long as the ratio is not too small ($m < 0.1$ in our case, which makes small buffer not large enough) or too large ($m > 0.5$ in our case, which makes the large buffer not sufficient enough). When there is moderate buffer resource ($Q = 100$ in our case), the system achieves best overall performance

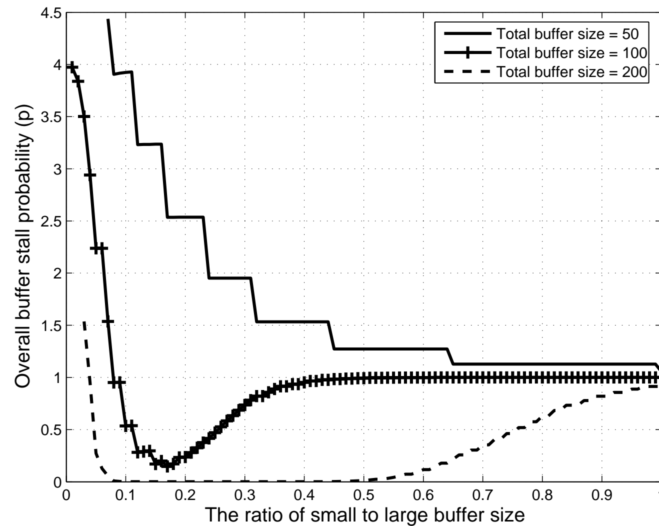

Figure 3.5: The overall system stall probability (the addition of stall probabilities of 5 buffers including the buffers for the processing core and other four edges) along the size ratio between small buffer to large buffer with different fixed total buffer resources; assuming a traffic pattern where the average number of data in the processing core is 50 and the other four channels each has average number 5.

when the buffer size ratio is a little larger than the channel traffic ratio (the optimal buffer ratio point $m = 0.18$, and the traffic ratio is $n = 0.1$). When there is very limited buffer resource ($Q = 50$ in our case), the system always has non-negligible performance loss, and the overall buffer stall probability is smallest when each buffer has the same size.

For systems with different traffics to our example, the exact results will be different, but the general conclusion drawn above should still be valid. Also, these discussions are for dynamic routing architectures; if using static routing as will be discussed in Section 3.1.3, only one register is necessary in each bypass channel for the pipelining and no additional buffer is required.

3.1.3 Static routing vs. dynamic routing

The inter-processor connections can be configured statically before the runtime (static routing), or dynamically at the runtime (dynamic routing). Traditional multiprocessor systems such as message passing architectures and the current chip multiprocessor research on NoC have widely covered the dynamic routing network, but the static configuration architecture was not intensively

studied. The key advantage of the static configuration is that for applications with predictable traffic, such as most of DSP applications, it can provide an efficient solution with small area cost and communication latency. The dynamic configuration solution has more widely suitable applications because of its flexibility, but it has non-negligible overhead in terms of the circuitry area and the communication latency; the main overhead comes from the routing path definition, the arbiter of multiple clock independent sources, and the signal recognition at the destination processor.

Dynamic routing and its overhead

In dynamic routing, the data transferring path should be defined by the source processor and propagated to the corresponding downstream processors, or dynamically decided by intermediate processors. The circuitry to define and control routing path has the area overhead, and to propagate the routing path might cost extra instructions and increase the clock cycles for the data transfer.

Since each link in the dynamic routing architecture is shared by multiple sources, an arbiter is required to allow only one source to access the link at one time. Furthermore, in GALS chip multiprocessors, this arbiter becomes more complex since it has to handle the sources with unrelated clock domains. An obvious overhead is that some synchronization circuitry is required for the arbiter to receive the link occupying request from different sources, and some logic is required to avoid the glitches when the occupying path changes.

Another question is how the destination processor can identify the source processors of the receiving data. Since data can travel through multiple processors with unknown clock domains, it is not possible to assume a particular order for the coming data. One method is that an address is assigned to each processor and sent along with the data, and the destination processor uses the address to identify the source processor through software or hardware.

Combining these overheads, the communication latency for dynamic routing is normally larger than 20 clock cycles [116], and this value will increase further for GALS dynamic routing networks due to the additional synchronization latency.

Table 3.3: Comparison of a couple of routing approaches. The data for Area are obtained at 0.18 μm technology; and the values can be slightly different in specific implementations due to other effects such as routing schemes

	Suitable applications	Latency (clk cycles)	Area (mm^2)
Static Systolic route	limited	$\sim$1	–
Dynamic route	broad	>20	–
RAW (static + dynamic)	broad	3	$\sim$4
proposed static route	broad	$\sim$5	$\sim$0.1

Static routing

Instead of discussing dynamic routing strategy, only the static routing approach is investigated in our design.

Not a lot of systems use static routing and the Systolic [78] system is one of the pioneers. Systolic systems contain synchronously-operating processors which "pump" data regularly through a processor array, and the data to be processed must reach the processing unit at the exact predefined time. Due to this strict requirement for the data stream, the systolic architecture is only suitable for applications with highly regular communication such as matrix multiplying. Its suitable applications are quite limited.

Releasing the strict timing requirement of the data stream can significantly broaden the application domain. To release the timing requirement, each processor must 'wait' for data when the data is late, and the data must 'wait' to be processed when it comes early. Inserting FIFO(s) at the input(s) of processing cores for the coming data can naturally meet these requirements, in which the processor is stalled when the FIFO is empty, and the data is buffered in the FIFO when it comes early. In this way, the requirement for the data stream is only its order, not the exact time.

RAW [116] is a chip multiprocessor with extremely low latency for interprocessor communication (3 clock cycles) using both static routing and dynamic routing, but it achieves this goal with a large area cost of about 4 mm^2 in 0.18 μm technology. Our communication system is suitable for broad applications, with low latency (about 5 clock cycles), and low area overhead (about 0.1 mm^2 in 0.18 μm technology).

Table 3.3 compares the interconnect architectures described in this subsection.

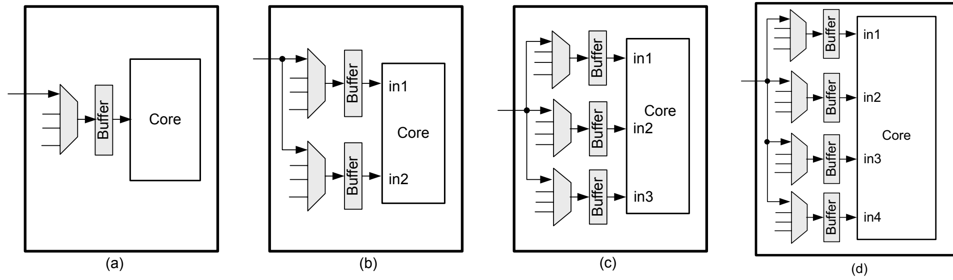

Figure 3.6: Diagrams of architectures with various numbers of input ports (and therefore various numbers of buffers) for the processing core: (a) single port, (b) two ports, (c) three ports, and (d) four ports

3.2 Design space exploration

Under the staticall configurable asymmetric architecture as discussed in Section 3.1, there are a couple of options for the communication logic realization. Two important options are investigated in this section, including:

- The number of ports (buffers) for the processing core
- The number of links at each neighboring processor pair

3.2.1 Single port vs. multiple ports for the processing core

In Fig. 3.3, each processing core has one input port (and one corresponding buffer). When using the dynamic routing approach, a single port might be sufficient since the processing core can fetch data from all directions by dynamically configuring the input mux. If using the static routing approach, a single port means each processing core can only fetch data from one source, which might be inefficient when multiple sources are required. Using multiple ports (buffers) for the processing core is considered, as shown in Fig. 3.6. The area of the communication circuitry roughly doubles if using two ports (buffers) instead of single port (buffer); and the area becomes three times and four times larger if using three and four ports (buffers).

Performance evaluation

The effect of the number of ports (buffers) on the routing capability is highly dependent on the application communication patterns. The *basic* communication patterns, including one-to-

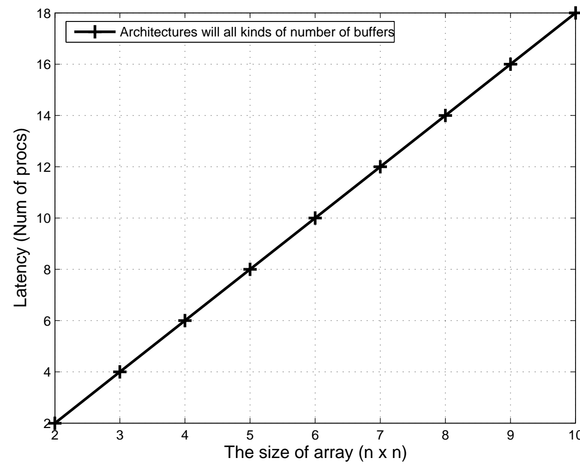

Figure 3.7: Architectures with different numbers of ports (buffers) for the processing core always travel through $2n-2$ processors to fulfill one-to-one and one-to-all communication

one communication (in which two processors at opposite corners communicate with each other), one-to-all broadcast (in which one corner processor sends data to all processors), all-to-one merge (in which all processors send data to a processor at the middle), and all-to-all communication are used to evaluate the performance of different architectures; the real applications can normally be obtained by the combination of these basic patterns.

For the one-to-one and one-to-all communications in which the destination processor(s) only need one source, the single-port architecture has the same performance as other multiple-port architectures and the communication latency is the time to travel through $2n-2$ processors in a n $\times$ n array, as shown in Fig. 3.7.

For the all-to-one communication case, as shown in Fig. 3.8. increasing the number of ports (buffers) has the benefit. For the single port (buffer) architecture, each processor can receive data from only one source and the furthest processor needs to propagate the data through all the n^2 processors, like all the processors are arranged in a linear array. For architectures with multiple ports (buffers), the communication can be distributed in multiple directions and the furthest processor only needs to propagate through around n processors. The architectures with 2, 3, or 4 ports (buffers) perform similarly; the two-port architecture is slightly worse in the 5 $\times$ 5 array and the four-port architecture is slightly better in the 3 $\times$ 3 array. Here the longest traveling distance is used to present the communication latency. When the array of processors becomes very large or if the

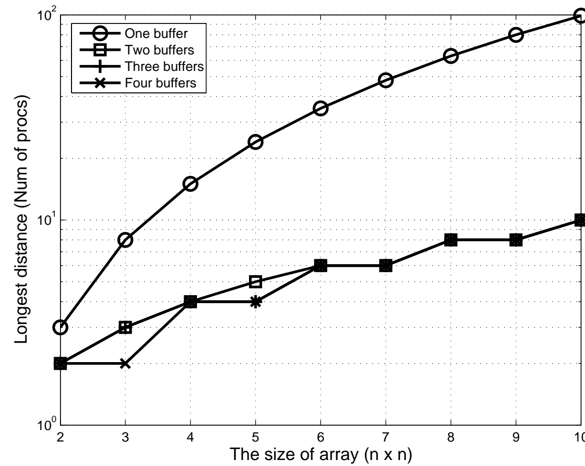

Figure 3.8: The longest traveling distance from source to destination for architectures with different numbers of ports (buffers) for the processing core, by varing the size of the array and using the all-to-one communication pattern

latency to propagate through each processor is very small, the communication latency might be limited by how fast the destination processor can *accept* the data not by how fast the furthest processor can reach the destination processor. For example, if passing one processor takes 5 clock cycles, in a n $\times$ n processor array the longest traveling latency is about $5n$ clock cycles; for a two-port architecture, it is able to accept no more than 2 data at each clock cycle, so the time to accept all the data is at least $n^2/2$ clock cycles. When $n < 10$, the traveling time dominates the latency; when $n > 10$, the time for the destination processor to accept the data dominates the latency. Besides how fast the destination processor can *accept* the data, another consideration is how fast it can *process* the received data. A single issue simple processor can normally consume (process) no more than two data at one clock cycle, which means it has little help to accept more than 2 data at each cycle, hence it also means to use 3 or 4 ports (buffers) for the processing core can not provide significant help.

The all-to-all communication differs to the all-to-one communication in that each processor needs to reach all the other processors. The architecture with one port (buffer) need to arrange all processors linearly and the latency is the time needed to propagate through all the n^2 processors; the architectures with multiple ports (buffers) can communicate in two dimensions and all of their latency is the time needed to propagate through one row and one column of totally $2n - 1$

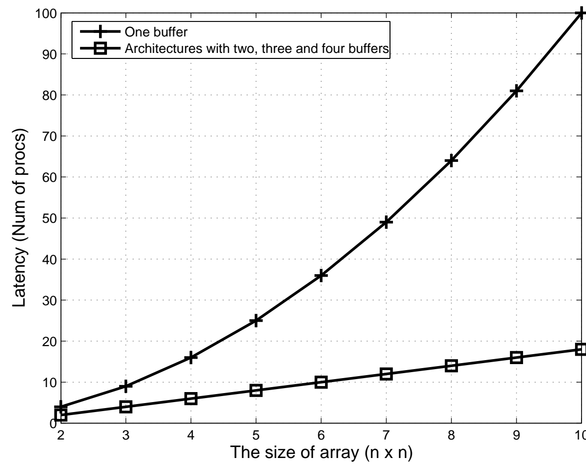

Figure 3.9: The longest traveling distance for architectures with different numbers of ports (buffers) for the processing core, by varying the size of the array and using the all-to-all communication pattern

processors. Fig. 3.9 shows the result. Similarly to the all-to-one communication discussion, here the longest traveling distance is used as the communication latency. How fast each processor can accept and process the received data can also affect communication and computation latency.

Considering the trade off between area and performance, using two ports (buffers) for the processing core is a good solution. A further extended consideration is that if the chip is in a 3-dimensional shape, then the best option will be three-port architecture.

3.2.2 Single link vs. multiple links

One of the key differences between interconnects of inside-chip and inter-chip is that there is more connect wire resources inside the chip while the inter-chip connect is normally limited by the available chip IOs. W.J. Dally [51] suggested increasing the word width of each link in NoC to take this advantage. Another option that can be used is to increase the number of links at each edge to increase the connection capability and flexibility.

Single link

When there is one link (one output as well as one input) at each edge, each link potentially can receive data from other three edges and the processing core. And each processing core

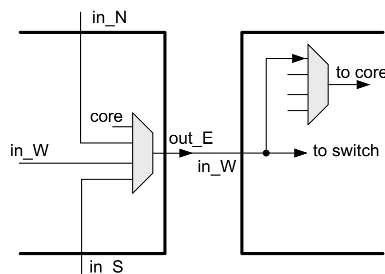

Figure 3.10: Diagram of inter-processor connection with one link at each edge

can receive data from four neighbors. Fig. 3.10 shows its diagram. It is a relatively simple and straightforward architecture.

Double links

If using double links at each edge, the source of the processing core becomes 8; and each edge has two outputs and each output potentially has 7 sources (6 from input edges and one from the processing core). Comparing the two-links architecture as shown in Fig. 3.11 (a) and the single-link architecture as shown in Fig. 3.10, the overhead of increasing the link is significant; not only because of the increased control logic, more importantly is that the semi-global wires (the wires in each processor from one edge to another edge) increase a lot which can affect system area/speed/power significantly in submicron technologies.

Some methods are available to simplify the fully connected architecture. Considering the router's logic at the east edge which receives data from *North*, *West*, *South* and *Core* and sends data to *East* output, in1_N, in2_N, in1_W, in2_W, in1_S, in2_S, sig_core, out1_E, out2_E are used to define these signals. There is a large exploration space at a first glance since 7 inputs and 2 outputs have totally 2^{14} connecting options. Since three input edges are symmetric, {in1_N, in1_W, in1_S} are grouped together as input in1 and another input group is named as in2, so that the input number is reduced to 3 and the exploration space is reduced to 2^6=64 as shown in Table 3.4. In options 1-8, the out1 does not have any connections so they are not considered as two-links architectures. Option 9 is neglected by similar reason. Option 10 only connects processing core to the outputs and basically it is the same as the nearest neighbor architecture. Option 11 can not be realized since out2 is only connected with in2 which means this link has no original source. Option 12 can be a

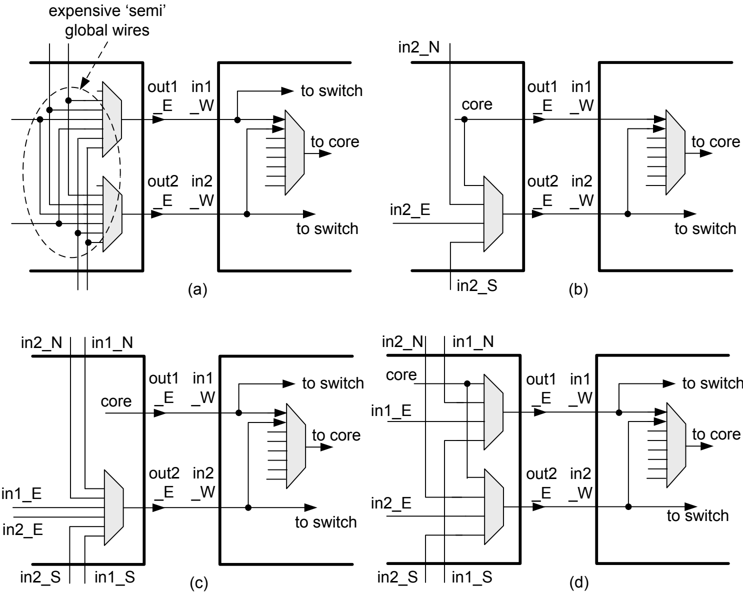

Figure 3.11: Inter-processor connections with double links: (a) fully connected; (b) separated nearest neighbor link and long distance link; (c) separated link from core and link from edges; (d) connections exist between 'corresponding' links; (b) (c) (d) are related to option 12, 15 and 44 in Table 3.4.

potential choice, where out1 is connected only to the core served as the nearest neighbor link, and out2 is connected to in2 and core served as the long distance connect link. Fig. 3.11 (b) shows the circuit diagram; each edge contains only one 4-input mux. By examining all the other options, it is found that option 15 and 44 can be potential good choices. In option 15, out1 (link1) receives data from the core while out2 (link2) receives data from edges, both of them send data to consumer core and routers. Fig. 3.11 (c) shows the circuit diagram; each edge contains one 6-input mux. In option 44, each link receives data from the core and a *corresponding* link (out1 corresponds to in1 while out2 corresponds to in2), and sends data to the processing core and routers. Fig. 3.11 (d) shows the circuit diagram; each edge contains two 4-input muxes.

In terms of the area cost and the routing flexibility of the four architectures shown in Fig. 3.11, architecture (a) has the most flexible connections while it has the biggest cost; architecture (b) has the most limited connections while it has the smallest cost; architecture (c) has the connection flexibility and cost in between (a) and (b). Architecture (d) has the area cost similar with (c), and interestingly, its routing capability is the same with architecture (a). This concept can be demonstrated by Fig. 3.12 where A needs to communicate with B, while path1 and path2 occupy

Table 3.4: Interconnect architecture options for double links. *Yes* means a connection exists between input and output, *No* means no connection exists, and *xx* means don't care.

options	in1-out1	in2-out1	core-out1	in1-out2	in2-out2	core-out2	corresponding circuitry
1-8	No	No	No	xx	xx	xx	
9	No	No	Yes	No	No	No	
10	No	No	Yes	No	No	Yes	
11	No	No	Yes	No	Yes	No	
12	No	No	Yes	No	Yes	Yes	Fig. 3.11 (b)
13	No	No	Yes	Yes	No	No	
14	No	No	Yes	Yes	No	Yes	
15	No	No	Yes	Yes	Yes	No	Fig. 3.11 (c)
16	No	No	Yes	Yes	Yes	Yes	
17-32	No	Yes	xx	xx	xx	xx	
33-40	Yes	No	No	xx	xx	xx	
41	Yes	No	Yes	No	No	No	
42	Yes	No	Yes	No	No	Yes	
43	Yes	No	Yes	No	Yes	No	
44	Yes	No	Yes	No	Yes	Yes	Fig. 3.11 (d)
45-48	Yes	No	Yes	Yes	xx	xx	
49-64	Yes	Yes	xx	xx	xx	xx	

some parts of the links between them. For the fully connected architecture, the path between A and B is easy to setup since it can first use link2 and then switch to link1. Using architecture as Fig. 3.11 (d) has difficulty at the first glance but it can be handled by re-routing path1 and then use link2 as the path between A and B, achieving the same routing purpose as fully connected architecture.

According to these discussions, Fig. 3.11 (d) architecture is a good option due to its routing flexibility and moderate circuitry cost; and Fig. 3.11 (b) can also be a good option due to its small circuitry cost.

Increasing the number of links further

Increasing the number of links further can increase the routing capability further. Fig. 3.13 shows architectures with three and four links enhanced from Fig 3.11 (b) and (d) architectures. The circuitry clearly becomes more complex along with the increased number of links. Each edge contains 2 or 3 4-input muxes in three links architecture and contain 3 or 4 4-input muxes in four links architecture. Also, the source of the processing core becomes 12 and 16 in three and four

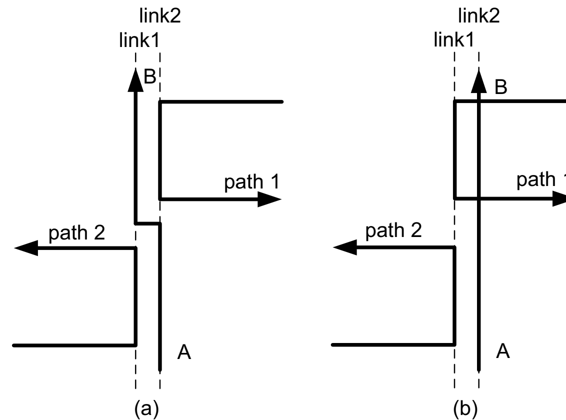

Figure 3.12: Setup the interconnect path from point A to B: (a) using fully connected architecture as Fig. 3.11(a); (b) reorganize the paths and using architecture shown in Fig. 3.11(d)

links architectures respectively. Since the wires connected to these logics are all semi-global, the overhead of these additional logic might has non-negligible effect on the system area and speed.

Since the architectures with various links have a large number of options and is not easy to judge according to the qualitative analysis, more quantitative analysis will be given in the following subsections. The evaluated architectures include the single-link architecture shown in Fig. 3.10, the double-links architectures shown in Fig. 3.11 (b) and (d), and three and four links architectures shown in Fig. 3.13.

Area and speed

Increasing the number of communication links requires additional control logic, which is expected to increase the circuitry area and effect the processor speed. The synthesis tool reports that the communication logic area for the discussed seven architectures are 0.013, 0.032, 0.047, 0.058, 0.067, 0.075, and 0.081 mm^2 respectively, in a 0.18 μm technology.

The result from the synthesis can not tell the whole story if not putting the communication circuitry into an entire processor environment and considering the physical layout effect. In the sub-micron technologies, wires introduce non-negligible delay compared to the logic gate; in addition, complex wire connections might require extra area for the routing. These communication circuitry is embedded in a simple single issue processor with the original area about 0.66 mm^2 in a 0.18 μm technology, and do the physical layout design use Cadence tool Encounter. The layout utilization

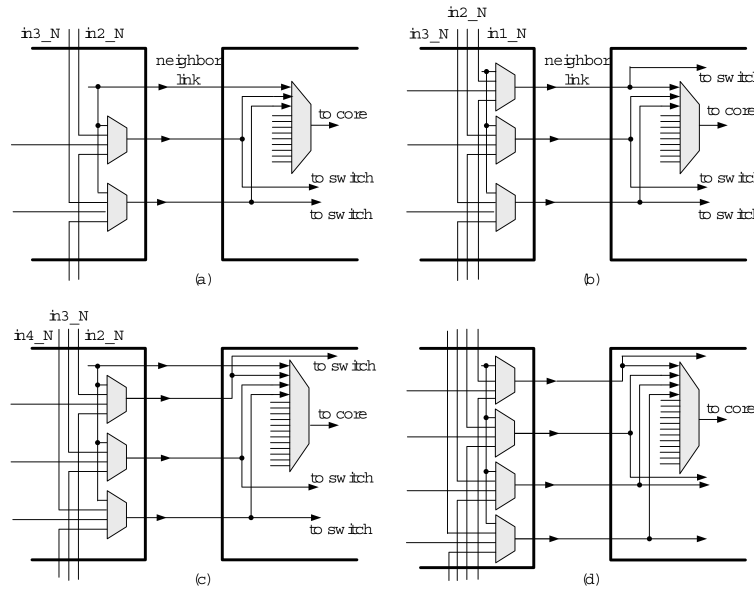

Figure 3.13: Inter-processor interconnect with three and four links. (a) and (c): separated nearest neighbor link and long distance link, similar with Fig. 3.11 (b); (b) and (d): connections exist between 'corresponding' links; similar with Fig. 3.11 (d)

in the physical design can have a strong impact on the design result. In terms of the area, higher utilization is always good as well as the chip is routable. In terms of the speed, too low utilization will introduce some unnecessary long wires and reduce system speed; too high utilization can result wiring congestion and complicate or prohibit the routing. It is found that setting 70% utilization initially is a good option, and it normally goes up to around 85% after clock tree insertion and inplacement optimization.

Fig. 3.14 shows the layouts of seven processors containing different numbers of communication links. Fig. 3.15 (a) shows their area, and Fig. 3.15 (b) shows the relative area increment of each architecture compared to type 1 architecture. The types 6 and 7 (four links architectures) have a noticeable increase of the area (close to 25% area increment). The reason is that these two architectures can not successfully finish the routing using the initial 70% utilization as others, and is reduced to 64% and 65% respectively. This result provides some interesting insight about how many global wires can fit into a chip. For a processor with a 0.66 mm^2 area and a 0.8 mm edge, assuming minimum 1 μm pitch between IO pins, an optimistic estimation is that each edge can fit 800 IO pins, beyond this range the chip size will become *IO pin or wire dominated*. This estimation

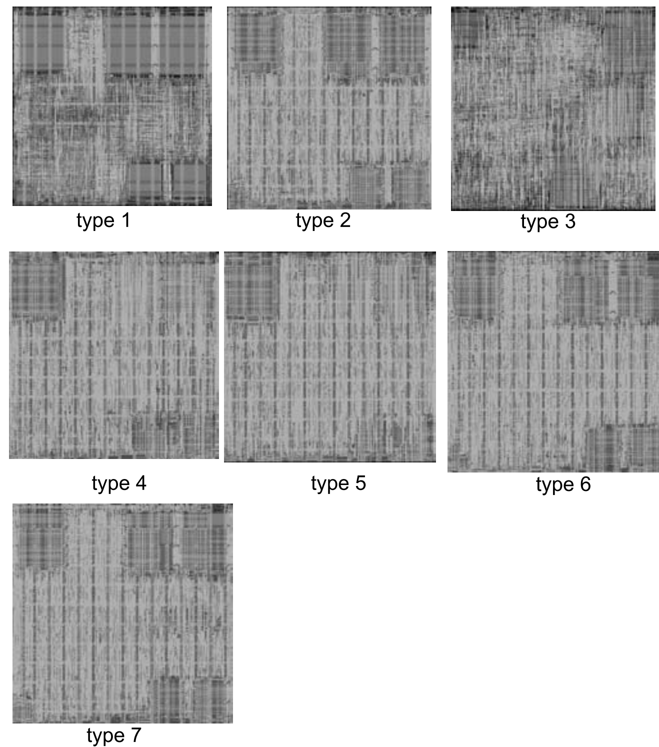

Figure 3.14: Scaled layouts of seven processors containing different numbers of communication links; type 1 to 7 correspond to the architectures shown in Fig. 3.10, Fig. 3.11(b), Fig. 3.11(d) and Fig. 3.13

can be true if these IO pins are all connected to short wires. For global wires as used for communication routers, increasing the number of wires will quickly result in routing congestion and increase the chip size. In our example, each processor edge in four-link architecture has about 160 IO pins, much less than the optimistic 800 IO pins. Four or more communication link architectures are not considered due to their area cost.

Fig. 3.15 (c) shows the processor's speed and Fig. 3.15 (d) shows the relative speed difference of each architecture compared to type 1 architecture. Each processor has similar speed and the difference is within a negligible 2%. Type 6 and 7 have a little faster speed compared to type 1 because the released area helps to simplify the routing.

Performance

Again the basic communication patterns are used to evaluate the performance of architectures with different numbers of links.

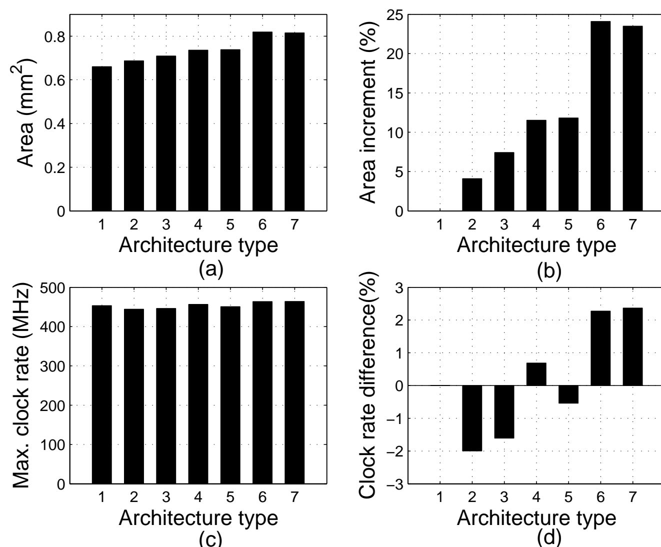

Figure 3.15: Comparing the area and the speed of processors with different communication circuitry containing different numbers of links; (a) the absolute area; (b) the relative area increment to architecture 1; (c) the absolute speed; (d) the relative speed decrement to architecture 1; type 6 and 7 (four links architecture) have a noticeable area increase due to their semi-global wire effect on the processor area

The communication latency between two processors can be expressed as $k \times p$ clock cycles where p is the distance between source and destination expressed as the number of processors and k is the clock cycles across one processor. For the nearest neighbor interconnect architecture, crossing one processor requires extra instruction and the latency (k) is dependent on the asynchronous synchronization time and the processor pipeline depth; it is 13 in our experimental system. For the proposed routing architecture, k is 1 if the data can use the routing circuitry and be registered in each processor.

For one-to-one or one-to-all communication, each processor only requires one source so that single link architecture is sufficient and has the same communication latency with other architectures, as shown in Fig. 3.16. A little surprisingly, all architectures have the same latencies for all-to-all communication, as shown in Fig. 3.17, because each processor needs data from both horizontal and vertical neighboring processors and both of the two ports (buffers) of each processor are occupied by the nearest neighbor connections, prohibiting the usage of the additional links for long distance communication.

These architectures have different results in all-to-one communication as shown in Fig. 3.18.

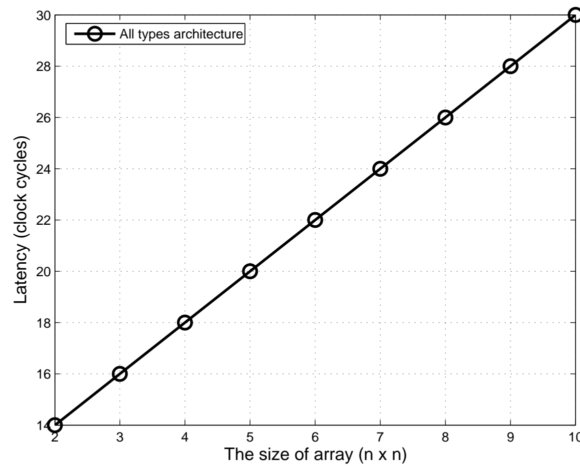

Figure 3.16: The communication latencies (clock cycles) of architectures with different numbers of links for one-to-one and one-to-all communication patterns, they all have the same latencies since single link is sufficient for those communications.

For type 1 (single link) architecture, most or all of the link resources are occupied by the nearest neighbor interconnect and little can be used for the direct switch, so the latency is relatively high. Increasing the number of links helps when the latency is limited by the link resources. Type 2 (double links with separated nearest neighbor link) has little advantage to type 1 but with a relatively much higher area, and type 4 (three links with separated nearest neighbor link) has little advantage to type 3 (double links) but with a relatively much higher area, so that type 2 and 4 are not considered. Type 3 architecture is about two times faster than the single link architecture, which makes it a good candidate. Comparing type 5 (three links) architecture with type 3, they have the same latency within a small communication domain (2×2 and 3×3 arrays), while the three-link architecture benefits when the array grows. For 4×4 to 6×6 arrays, three-link architecture has about 25% smaller latency; for 7×7 to 9×9 arrays, it has about 30% smaller latency, and the benefit increases along with the larger array.

3.3 Supporting long distance communication with GALS clocking style

The design methodology to support nearest neighbor communication with GALS style is already discussed in Section 2.2.2. The design becomes more complex when considering the long

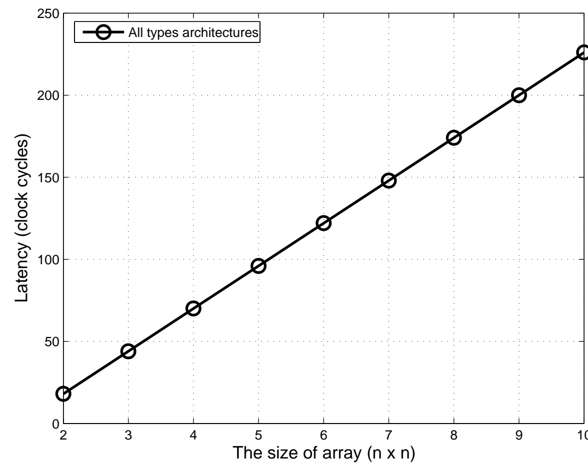

Figure 3.17: The communication latencies (clock cycles) of architectures with different numbers of links for the all-to-all communication patterns; they all have the same latencies since two ports are always occupied by the nearest neighbor connections and therefor an increased number of links can not be used.

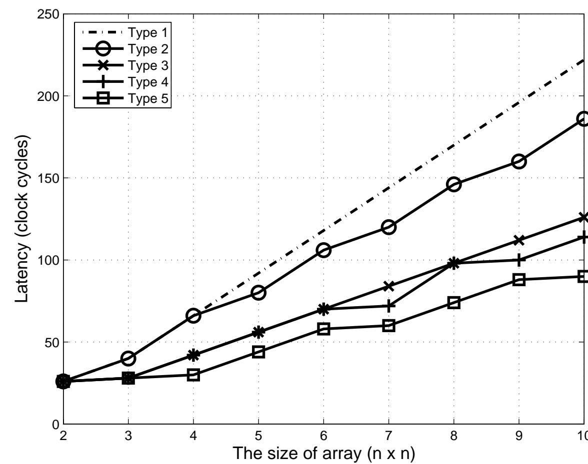

Figure 3.18: Comparing the communication latency (clock cycles) of interconnect architectures type 1 to 5, by varing the size of the array and using the all-to-one communication pattern. Type 6 and 7 architectures are not included in the comparison due to their high area cost as shown in Fig. 3.15

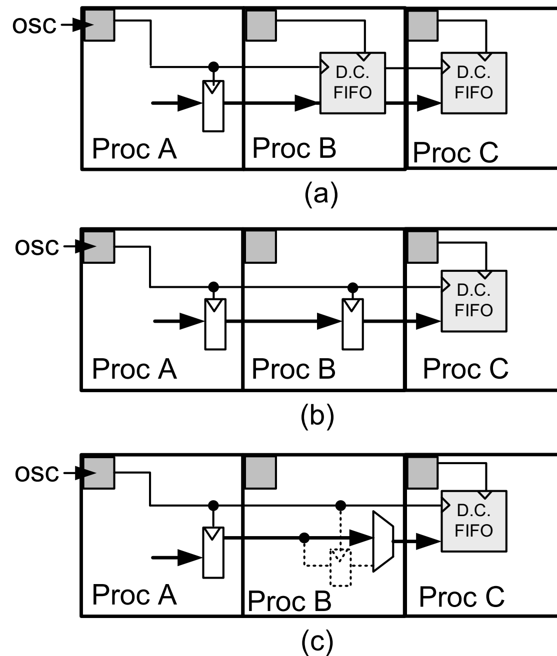

Figure 3.19: Synchronization strategies for long distance communication between processors with unrelated clocks: (a) using intermediate processors' clocks with dual-clock FIFOs, (b) using source synchronous clocking with pipeline stages in each intermediate processors, and (c) using source synchronous clocking with selectable registering in intermediate processors

distance communication. For the asynchronous handshake scheme, each bit of the signal might reach the destination in a very different time, and some specific logic is necessary to guarantee the arrival of all bits of the signal [117]. The logic overhead to handle this asynchronous communication is not negligible, and the propagation delay can be significant.

Significant research exists in investigating the clock issue on the network on chip. Intel proposed 2D array with packet-switched routers and employs mesochronous clocking where each processor has the same clock frequency while the clock phase can be different [37]. ST proposed an asynchronous network-on-chip which uses handshake scheme to handle the asynchronous communication [108].

3.3.1 Source synchronization for long distance communication

For the source synchronous scheme in long distance communication, in order to avoid different data bits reaching the destination in different clock period, prevent the signal delays larger than the clock period, and minimize the antenna and/or crosstalk effect from long distance wires,

the data signals likely need to be registered (pipelined) in intermediate processors along its path. Two options exist for this task, as shown in Fig. 3.19 (a) and (b), where processor A sends data to processor C through processor B. The first option is to register the signals using intermediate processors' clock by a dual-clock FIFO, as shown in Fig. 3.19 (a). This scheme should work well under most of the situations, but is not efficient due to significant additional circuits and increased path latency. In addition, if an intermediate processor's clock is running slowly, it will be a bottleneck to the link's throughput.

Extending the source synchronous method that routes the initial source clock along the *entire* path is proposed, as shown in Fig. 3.19 (b). In the case when the source processor is running at slow speed, the registers in the intermediate processors are unnecessary and the link latency can be reduced with non-pipelined paths as shown in Fig. 3.19 (c). Besides the data and the clock, some information indicating the FIFO full and empty are also required to be transferred to guarantee the correct FIFO operation.

3.3.2 Care more about the clock delay, less about skew or jitter

The dual-clock FIFO design allows arbitrary clock skew and drift, which greatly simplifies the circuit design. The most important parameter of the clock used for the synchronization is no longer the skew or jitter, but is the propagation delay.

Firstly, the delay of clock must relate to the delay of data to meet the setup/hold time requirement during the register writing. Some configurable delay logic might be needed in the path of the data [45] or the path of the clock for this purpose. Secondly, the delay of clock (and data) should be minimized if possible, since increasing the communication delay not only increases the application computation latency, but also can reduce the application computation throughput.

3.4 Implementation and results

The communication circuitry is implemented using the topology shown in Fig. 3.3, the static routing with two ports (buffers) for the processing core as shown in Fig. 3.6 (b) and double links at each edge as shown in Fig. 3.11 (d). and the extended source synchronous strategy shown in Fig. 3.19 (c). Each processor with the communication circuitry occupies 0.71 mm^2 in 0.18 μm

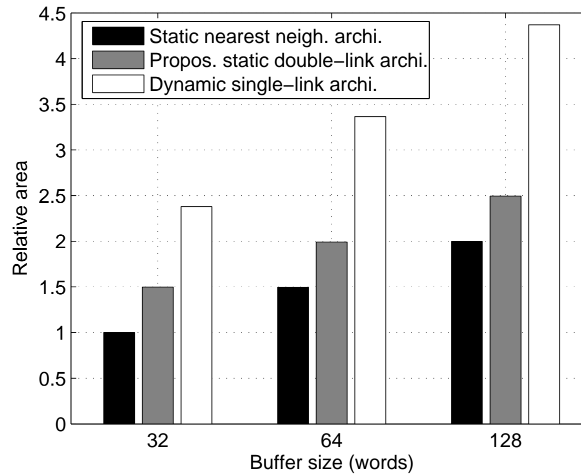

Figure 3.20: The relative communication circuit area of several interconnection architectures. The static single-link nearest neighbor and proposed static double-link asymmetric architecture contain two ports (buffers) for the processing core, and dynamic routing architecture contains four buffers for the router's input ports with a single link at each edge.

CMOS technology, and the critical path delay is 2.24 ns.

The implemented example is compared to a traditional dynamically configurable interconnect architecture with symmetric buffer allocation and single-link between each neighboring processor pair; we use 20 clock cycles as the communication latency [116], and we also assume the destination processor receives one data and its address separately in two clock cycles (sending address together along with the data and using hardware can receive and recognize the source in one clock cycle but it requires extra hardware).

3.4.1 Area

Different communication architectures, including the static nearest neighbor interconnect, the proposed double-link routing architecture, and the traditional dynamic routing architecture are modeled in 0.18 $\mu\mathrm{m}$ CMOS technology. Fig. 3.20 compares the communication circuitry area (including the buffers and the control logic) with different sizes of each buffer, and the area of nearest neighbor connect architecture is scaled to 1. When the buffer is not large, the control logic plays an important role in the area. For example, when the buffer size is 32 words, the double-link architecture is about 1.5 times larger than the nearest neighbor architecture. Along with the increased buffer

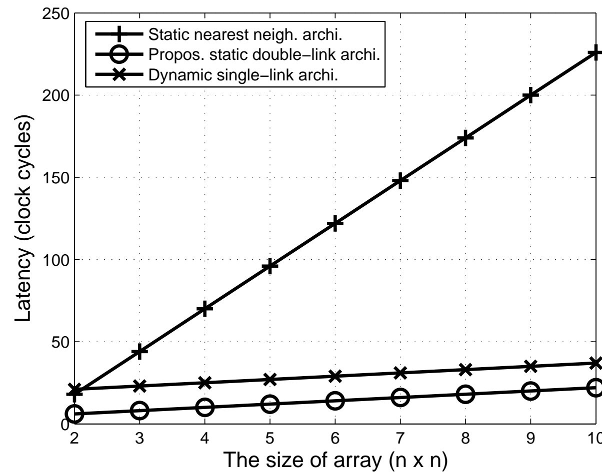

Figure 3.21: Comparing the communication latency of the static nearest neighor architecture, dynamic single-link routing architecture , and the proposed static double-link architecture, by varing the size of the array and using one-to-one and one-to-all communication patterns

size, the area of the communication circuitry will be dominated by the buffer size. When the buffer is 128 words, the proposed double-link architecture has approximately 25% larger area compared to the nearest neighbor architecture, while the traditional dynamic routing architecture is more than 2 times larger.

3.4.2 Performance comparison

Performance of the basic communication patterns

Fig. 3.21, 3.22, and 3.23 shows the latency of the basic communication patterns mapped onto different architectures along with different array sizes. The one-to-one communication has the same result as the one-to-all broadcast.

The proposed double-link routing architecture has significant lower communication latency compared to the nearest neighbor architecture. The latency of dynamic single-link routing architecture is similar to the static double-link architecture; it is a little worse in the one-to-one communication and all-to-one patterns; it is a little better in the all-to-all communication since where the flexibility of dynamic routing overcomes its disadvantages.

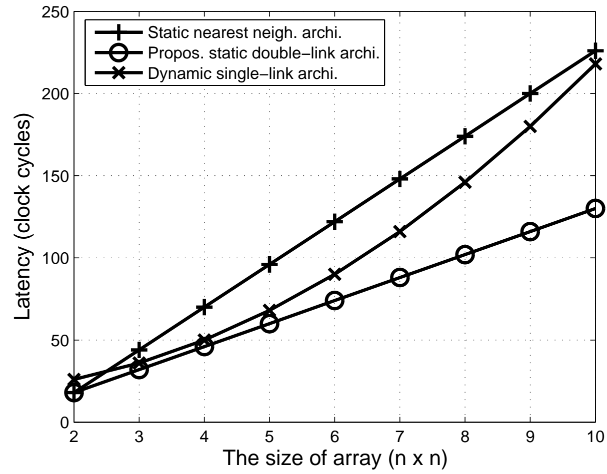

Figure 3.22: Comparing the communication latency of the static nearest neighor architecture, dynamic single-link routing architecture , and the proposed static double-link architecture, by varing the size of the array and using all-to-one communication pattern

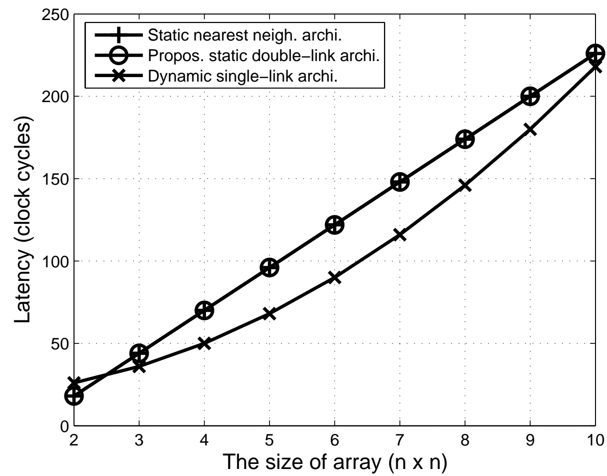

Figure 3.23: Comparing the communication latency of the static nearest neighor architecture, dynamic single-link routing architecture , and the proposed static double-link architecture, by varing the size of the array and using all-to-all communication pattern

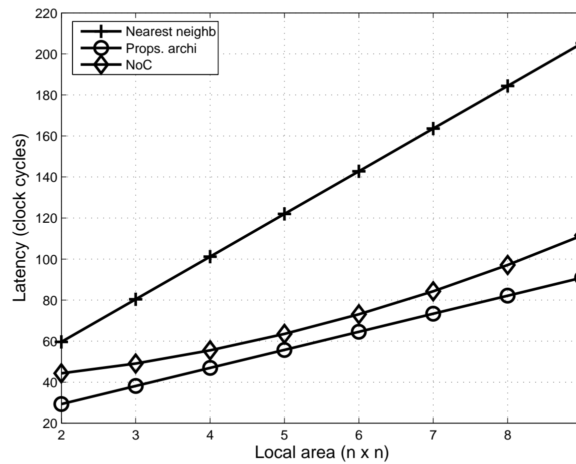

Figure 3.24: Comparing the communication latency of of static nearest neighbor architecture, dynamic single-link routing architecture, and the proposed static double-link architecture by mapping various application models to a 10×10 array. The application models are obtained by uniformly combining four basic communication patterns and assume 80% of the communication is within local area, and the meaning of *local area* varies from 2×2 to 9×9 array.

Combining the basic patterns

Although the communications of real applications have a wide range of variation, they can normally be modeled using some combination of basic communication patterns. This subsection investigates one application model mapped onto a 10×10 array.

The modeled communication is organized uniformly by the four basic communication patterns and assumes 80% of the communication is within the local area which is the value normally used in literatures [118]; in order to generally cover the localization parameters of different communication patterns, eight data points present the situations where the definition of *local* area varies from 2×2 to 9×9 arrays. Fig. 3.24 shows the results of latency.

Not surprisingly, static nearest neighbor architecture performs worst and the proposed static double-link routing architecture is more than 2 times faster than it, and dynamic single-link routing architecture is a little worse than the static double-link architecture.

3.5 Summary

An asymmetric inter-processor communication architecture which uses more buffer resources for nearest neighbor connections and fewer buffer resources for long distance interconnect is proposed; the architecture also provides the ability to support long distance GALS communication by extending the source synchronous method. Several design options are explored. Static routing is emphasized due to its low cost and low communication latency. Inserting two ports (buffers) for the processing core and using two or three links at each edge can achieve good area/performance trade offs for chip multiprocessors organized by simple single issue processors; and the optimal number of links is expected to increase if the chip is organized by larger processors.

Chapter 4

Physical Implementation of the GALS Multi-core Systems

Tile-based multi-core systems with GALS clocking styles provide great benefits, such as high performance, energy efficiency and scalability, as discussed in Section 2.1.4, but they impose some design challenges during the physical implementation regarding the robust handling of timing issues and limits in taking full advantage of system scalability.

GALS multi-core systems introduce challenges in handling the timing of signals due to the multiple clock domains. The timing issues include signals within a single processor, signals between processors, and signals between chips. A dual clock FIFO can reliably handle the asynchronous interface within a single processor and is already described in Section 2.2.2, and this chapter will discuss the inter-processor and inter-chip timing issues in Section 4.1. Section 4.2 investigates how to take full advantage of system scalability, mainly concerning some unavoidable global-style signals such as the configuration signals, power distribution and processor IO pins. Finally, Section 4.3 discusses some other issues of AsAP processor implementation such as design flow, high speed implementation, and testing etc.

4.1 Timing issues of GALS multi-core systems

Figure 4.1 contains an overview of important timing issues in GALS chip multiprocessors using coarse grain source synchronous flow control. All signals in such interfaces can be classified

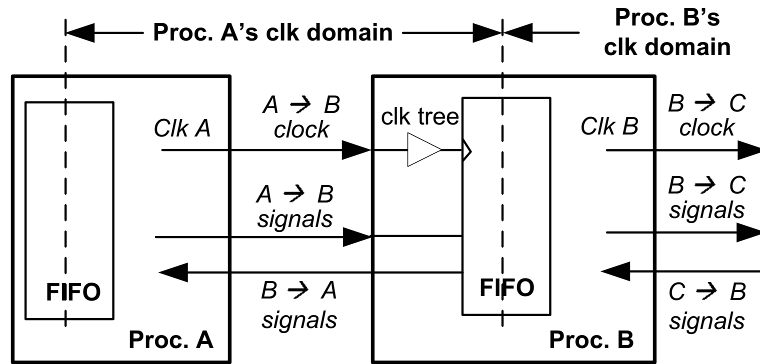

Figure 4.1: An overview of timing issues in GALS chip multiprocessors; each clock domain covers multiple processors and each processor contains multiple clock domains

into three categories. Here is 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: these include 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.

4.1.1 Inter-processor timing issues

Figure 4.2 shows three strategies to send signals from one processor to another. Fig. 4.2 (a) is an aggressive method where the clock is sent *only* when there is valid data (assuming 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. 4.1.

$$t_{hold} < D_{data} - D_{clk} < T - t_{setup} - t_{clk_to_Q} \tag{4.1}$$

Fig. 4.2 (c) is a conservative method where the clock is always active. With this method, the delay of data does not have to satisfy Eq. 4.1, as long as *data* and *valid* reach processor B within the

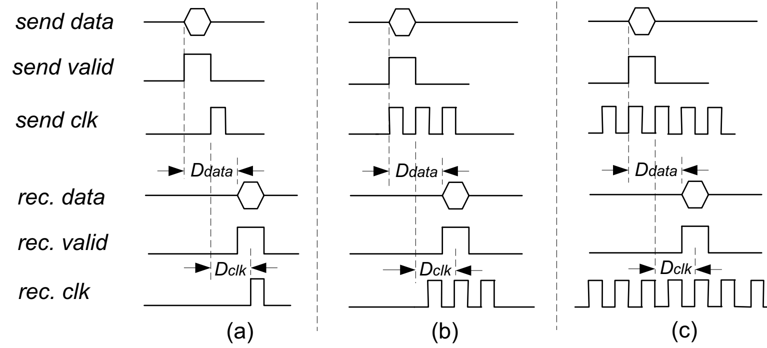

Figure 4.2: Three methods for inter-processor communication: (a) sends clock only when there is valid data; (b) sends clock one cycle earlier and one cycle later than the valid data; (c) always sends clock

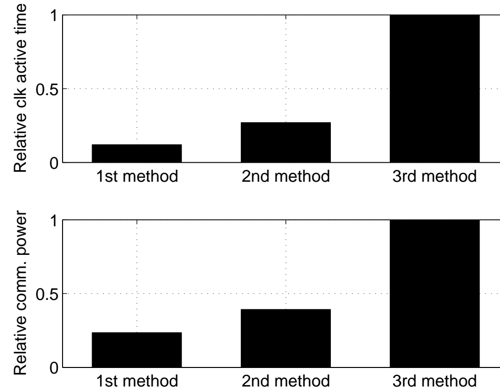

Figure 4.3: Relative clock active time and communication power consumption for the three inter-processor communication methods described in Fig. 4.2 for a 2-D 8×8 DCT application

same clock period, which results in a system similar to wave pipelined systems [119] with a slightly modified set of timing equations and of course additional constraints on minimum path times. A compromise method as shown in Fig. 4.2 (b) is proposed, 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 4.3 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. 4.2.

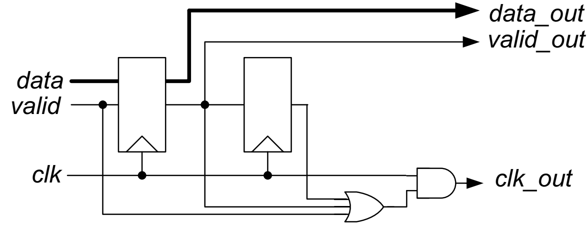

Figure 4.4: Circuit for the Fig. 4.2 (b) inter-processor communication method

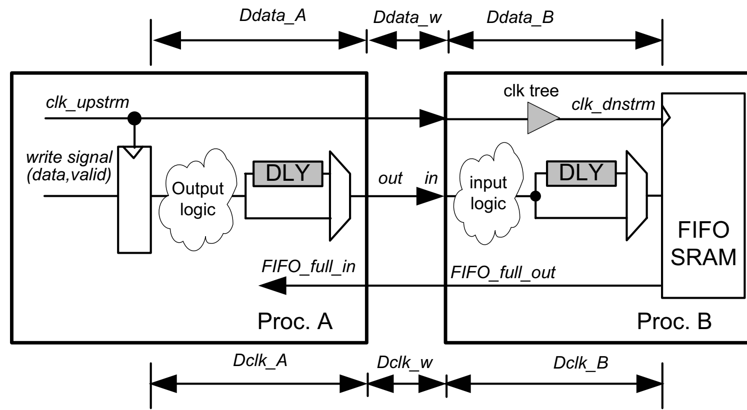

Figure 4.5: Configurable logic at the inter-processor boundary where one clock domain covers two processors; GALS style brings an additional clock tree at the consumer processor

Figure 4.4 shows a circuit to realize this second scheme.

$$-T < D_{data} - D_{clk} < 2T \tag{4.2}$$

Figure 4.5 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. 4.5. Equation 4.1 can then be simplified as follows:

$$t_{hold} < D_{insert} - D_{clk_tree} < T - t_{setup} - t_{clk_to_Q} \tag{4.3}$$

Here D_{insert} and D_{clk_tree} are the delays of inserted logic and clock tree respectively. Normally

the t_{hold}, t_{setup}, and $t_{clk_to_Q}$ can be neglected since they are generally small compared to the clock period. Equation 4.4 lists three delay possibilities for the inserted delay logic, and the optimal delay for each inserted gate is shown in Eq. 4.5 and Eq. 4.6.

$$D_{insert} = 2D_{mux} + \{0, D_{DLY}, 2D_{DLY}\} \tag{4.4}$$

$$D_{mux} = D_{clk_tree}/2 \tag{4.5}$$

$$D_{DLY} = T/2 \tag{4.6}$$

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. 4.5, 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. 4.7 and 4.8 for the architecture shown in Fig. 4.5.

$$output_delay = T - D_{data_A} \tag{4.7}$$

$$input_delay = T - D_{data_B} \tag{4.8}$$

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 4.1 lists delays for one example case study.

4.1.2 Inter-chip timing issues

Similarly to the case of inter-processor communication, *inter-chip* communication presents timing challenges as illustrated in Fig. 4.6. Besides the delay at the producer processor (D_{A1}) and consumer processor (D_{B1}), there is also delay at the chip A boundary (D_{A2}), chip B boundary

Table 4.1: 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

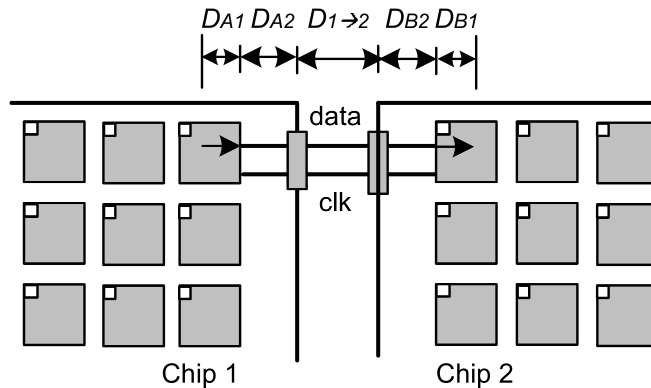

Figure 4.6: Inter-chip communication

(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. 4.2 also apply to inter-chip communication and the configurable delay logic embedded into each processor shown in Fig. 4.5 is still valuable to adjust the data delay for inter-chip communication. Due to the more complex environment with inter-chip communication, Fig. 4.2 (b) and (c) methods are preferred. Synthesis timing constraints for the chip IO signals can also help the inter-chip timing issue.

4.2 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 issue is that some signals unavoidably have some *global* features and have to be considered carefully. Correspondingly, the

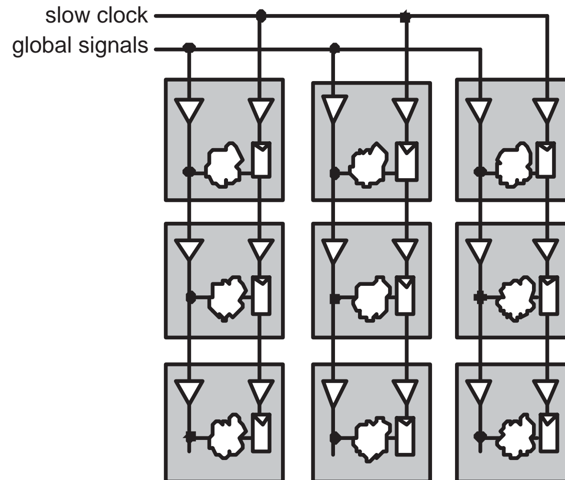

Figure 4.7: Global signals controlled by a low-speed clock are buffered with inserted buffers and route through processors with internal wiring

key idea is to try to avoid or isolate those signals if possible, so that multiple processors can be directly tiled without further changes.

4.2.1 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 (used to configure the connections between processors, or to configure the local oscillator frequency, etc.) and test signals. These global signals can be pipelined into multiple segments [120] and still run at full speed, or can use totally asynchronous communication to avoid clock constraints [121]—both of these methods significantly increase design difficulty. There is a fact 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 4.7 illustrates this scheme.

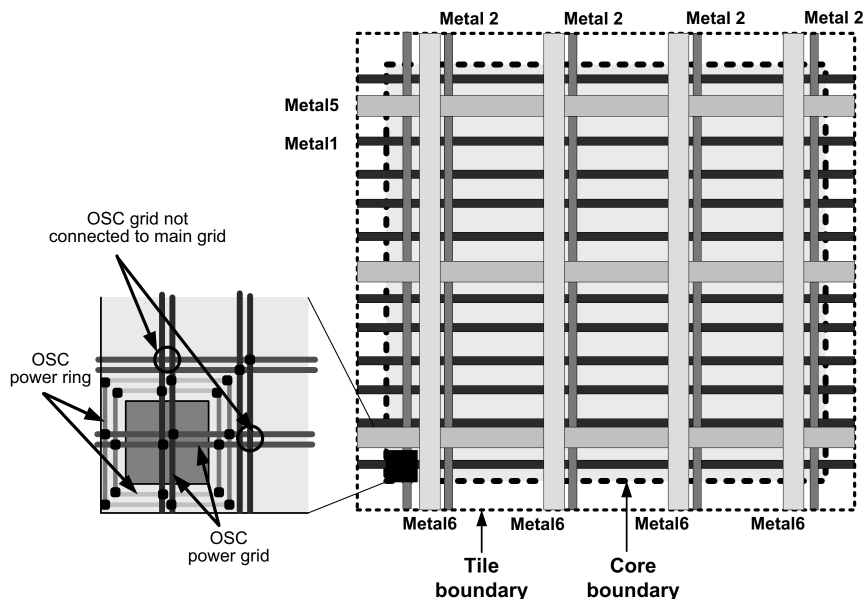

Figure 4.8: An example power distribution scheme with power wires reaching beyond the processor core to allow it to directly abut other processors

4.2.2 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. 4.9, where $V_2 - V_{orig}$ is the allowable voltage drop, I is the estimated current for the entire chip (not a single processor) which can be obtained from the Encounter, L and w are the length and width of the metal wire, and ρ is the metal's resistivity per unit length and width.

$$I \times \frac{\rho L}{w} = V_2 - V_{orig} \tag{4.9}$$

Figure 4.8 shows a complete power distribution example for a single processor using 6 metal layers, which is common in 0.18 μm 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.

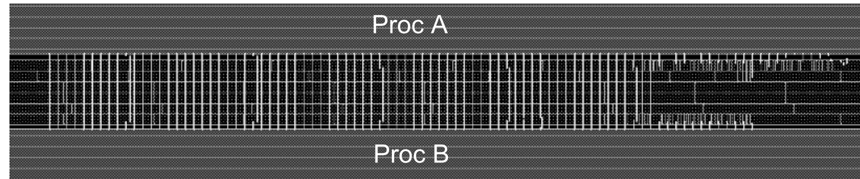

Figure 4.9: Pins connections between two processors vertically; nearly directly abutting each other enables very short wires

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. 4.8 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.

4.2.3 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 4.9 and 4.10 show example connections of two processors in vertical and horizontal directions, where IO pins directly abut each other with very short connecting wires.

4.3 A design example — implementation of AsAP

The single-chip tile-based 6 $\times$ 6 GALS AsAP multiprocessor is designed and implemented in a 0.18 μm CMOS technology [36]. Data enters the array through the top left processor and exits through one of the right column processors, selected by a mux. Input and output circuits are available on each edge of all periphery processors but most are unconnected in this test chip due to package I/O limitations.

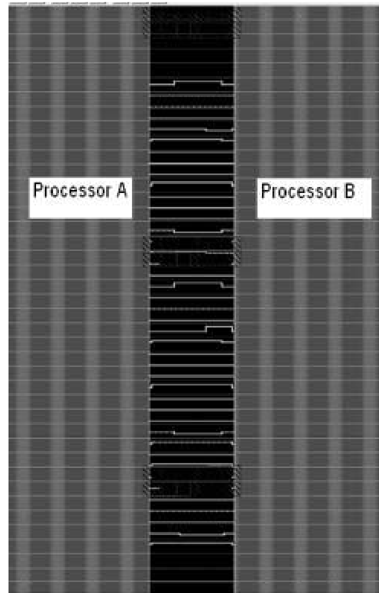

Figure 4.10: Pins connections between two processors horizontally

The chip utilizes the Artisan standard cell library and was auto placed and routed. Figure 4.11 shows the die micrograph. The chip is fully functional at a clock rate of 520–540 MHz under typical conditions at 1.8 V, and achieves a peak performance of 19 GOPS with a power consumption of approximately 3.4 W. The size of each processor is 0.66 mm^2. Each processor contains 230,000 transistors, and dedicates approximately 8% of its area to communication circuits, and less than 1% to each local clock oscillator.

4.3.1 Physical design flow

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. 4.12 shows the hierarchical physical design flow. A local oscillator is used to provide the clock for each processor, and it is designed separately for more robust operation. After a single processor is auto placed and routed, processors are arrayed across the chip, and the small amount of global circuitry is auto placed and routed around the array. The right column of Fig. 4.12 shows the physical design flow for a single processor, and a similar flow also applies for the oscillator and the entire chip.

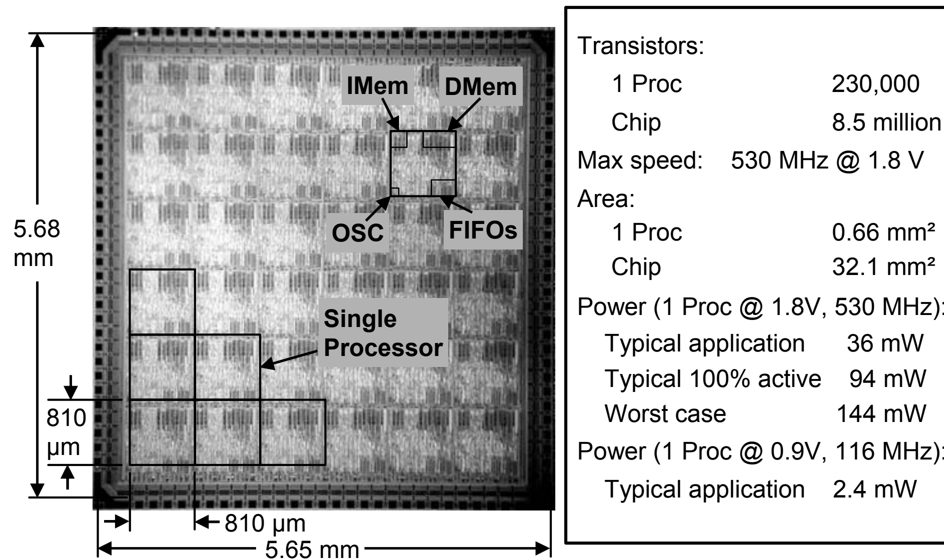

Figure 4.11: Chip micrograph of a 6×6 GALS array processor

The chip is fully synthesized from verilog, except the clock oscillator which was designed by hand from standard cells. The Imem, Dmem, and 2 FIFO memories are built using memory macro blocks. 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 as shown in Fig. 4.13 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.

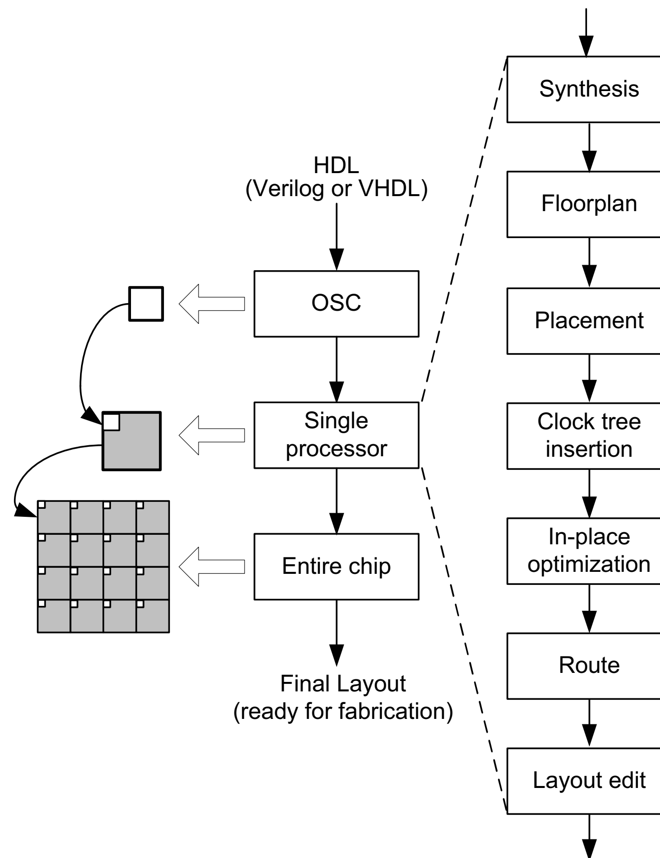

Figure 4.12: Hierarchical physical design flow of a tile-based GALS chip multiprocessor, with details specific to a synthesized standard cell design. A single processor tile can be replicated across the chip.

4.3.2 Implementation for high speed

Several methods were used to speed up the processor clock frequency during the synthesis. The MAC unit was synthesized separately by Module-compiler; the synthesis script was optimized for high speed such as setting high constraint clock period and using high effort compile; the RTL code of some function modules were rewritten using parallel architecture to replace serial style. Using these methods, the reported clock period from synthesis was reduced from 2.49 ns to 1.82 ns. The effect of these methods is shown in the upper part of Fig. 4.14. The critical path comes from ALU and its feedback path.

Along with the more advanced technology, the wire delay is becoming an important issue. Placement and routing has a large impact on the system speed. Timing driven placement and routing

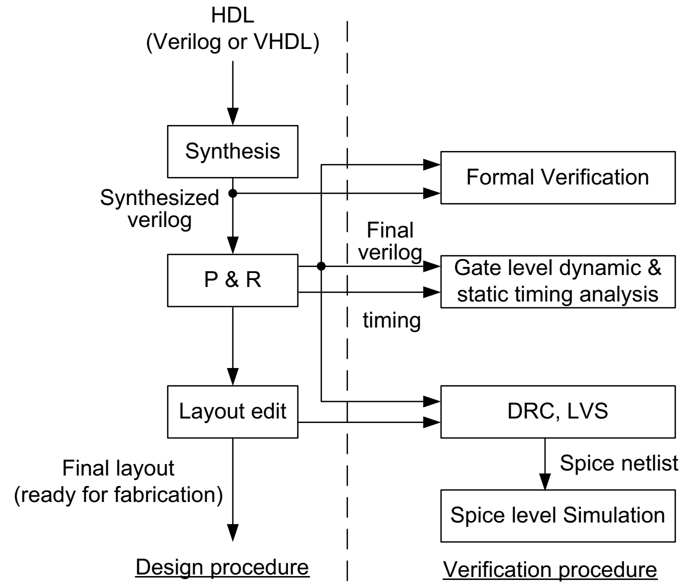

Figure 4.13: A standard cell based back end design flow emphasizing the verification

was used in our flow. In-placement optimization was used, which enlarges some gates to compensate the wire delay introduced during the place and routing. Careful metal fill parameters were chosen to meet the metal density requirement with little speed effect. The active spacing (the space between signal metal and filler metal) was constrained to bigger than 1 um and the spacing between metal fills was constrained to bigger than 0.6 um. Using these methods, the reported clock period after placement and routing was reduced from 4.26 ns to 2.22 ns. The effect of these methods is shown in the lower part of Fig. 4.14.

The clock tree has to be carefully concerned for the correct function and high speed, especially when containing multiple clock domains as in our case. There are four clock trees in each single processor: the main clock generated by local oscillator, two FIFO write clocks, and the configuration clock. The reported clock skew for these four are 44 ns, 10 ns, 20 ns and 39 ns respectively, and the transition rise time at the buffer and end registers is less than 120 ns.

4.3.3 Testing

For testing purposes, 27 critical signals from each of the 36 processors, totaling 972 signals in the chip, can be selectively routed to eight chip pads for real-time viewing of these key signals

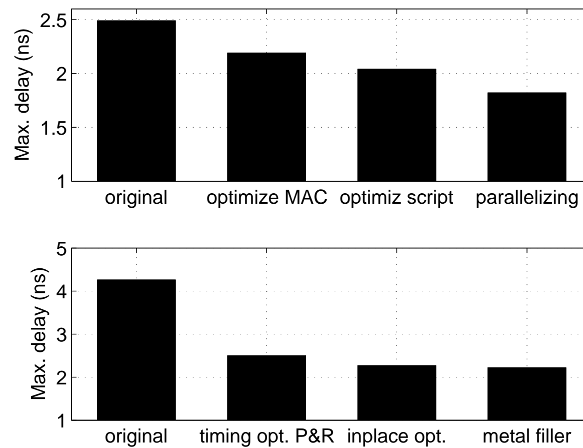

Figure 4.14: Speed-up methods during synthesis (upper figure) and place and routing (bottom figure)

which include: clocks, stall signals, FIFO signals, PC, etc. Figure 4.15 shows the test environment for the AsAP prototype, built by group member Jeremy Webb, including a printed circuit board hosting an AsAP processor and a supporting FPGA board to interface between AsAP's configuration and data ports and a host computer. There is one SPI style serial port designed in the AsAP processor which receives external information and commands for configuration and programs.

4.4 Summary

This chapter discusses implementation techniques for tile-based GALS chip multiprocessors. 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 interprocessor 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.

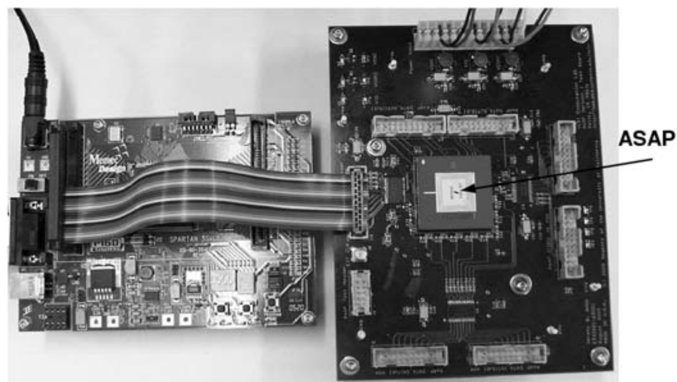

Figure 4.15: AsAP board and supporting FPGA-based test board

Chapter 5

Results and Evaluation of the Multi-core System

This chapter presents and evaluates the measurement results of the fabricated chip including performance, area and power consumption. In addition, a thorough evaluation of its performance and energy efficiency for several DSP applications by comparing to some related processors is presented.

5.1 Area, speed, and power

Each processor occupies 0.66 mm^2 and the 6$\times$6 array occupies 32.1 mm^2 including pads. The chip operates at 520–540 MHz under typical conditions at 1.8 V, and 116 MHz at 0.9 V. Since AsAP processors dissipate zero active power when idle for even brief periods of time (it is common in complex applications), and because different instructions dissipate varying amounts of power, it is useful to consider several power measurements. The average power consumption for each processor is 36 mW when processors are executing applications such as a JPEG encoder or an 802.11a/g baseband transmitter. Processors that are 100% active and executing a “typical application” mix of instructions dissipate 94 mW each. The absolute worst case power per processor is 144 mW and occurs when using the MAC instruction with all memories active every cycle.

Table 5.1: Area breakdown in a single processor

	Area (μm^2)	Area percentage
Core and others	433,300	66.0 %
Data memory	115,000	17.5 %
Instruction mem.	56,500	8.6 %
Two FIFOs	48,000	7.4 %
Oscillator	3,300	0.5 %
Single processor	656,100	100.0 %

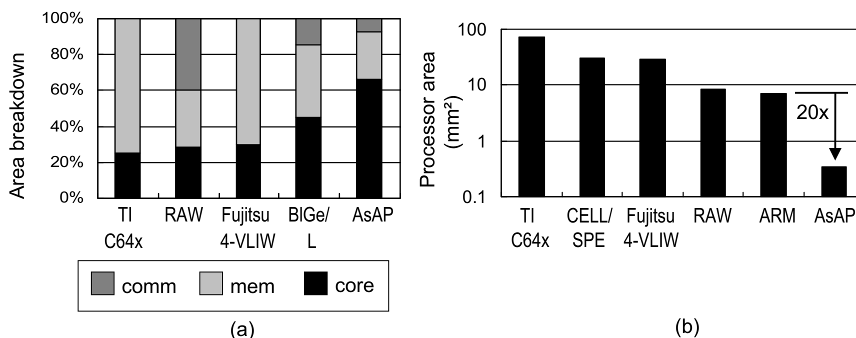

Figure 5.1: Area evaluation of AsAP processor and several other processors; with technology scaled to 0.13 μm

5.1.1 Small area and high area efficiency

Due to its small memories and simple communication scheme, each AsAP processor devotes most of its area to the execution core and thereby achieves a high area efficiency.

Table 5.1 shows the area breakdown for each AsAP processor. Each one dedicates 8% to communication circuits, 26% to memory circuits, and a favorable 66% to the core. These data compare well to other processors [7, 35, 122, 25, 23, 123], as shown in Fig. 5.1(a), since the other processors use 20% to 45% of their area for the core. Each AsAP processor occupies 0.66 mm^2 and the 6×6 array occupies 32.1 mm^2 including pads, global power rings, and a small amount of chip-level circuits. Figure 5.1(b) compares the area of several processors scaled to 0.13 μm, assuming area reduces as the square of the technology's minimum feature size. The AsAP processor is 20 to 210 times smaller than these other processors.

The processor area, or say the level of granularity of each processing element, is one of the most important variables in chip multiprocessor architecture. A wide range of granularities are possible, as shown in Fig. 5.2 [36, 35, 23, 13] which has similar information as Fig. 5.1(b) but shows

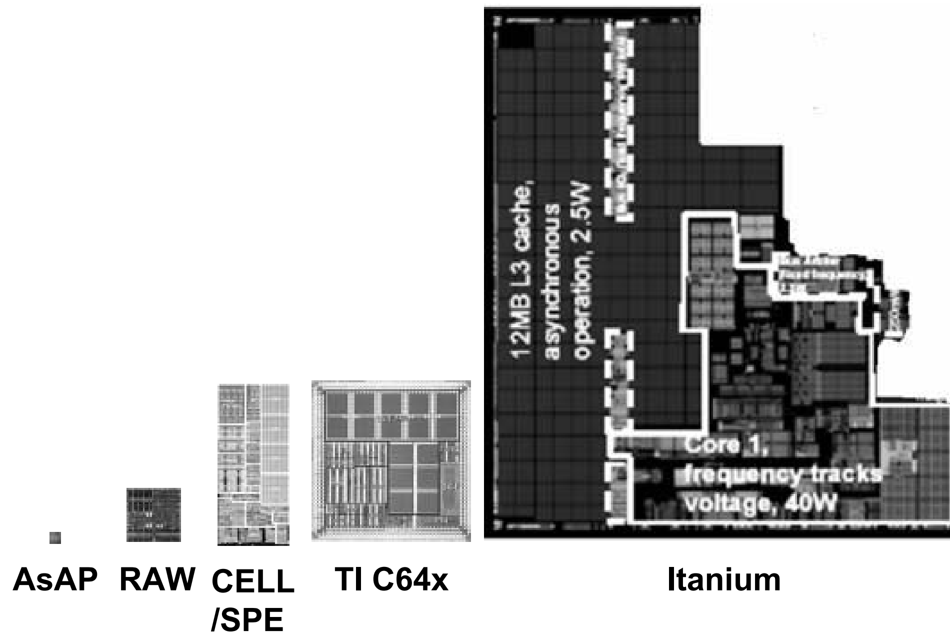

Figure 5.2: Approximate scale die micrographs of the multicore processors AsAP, RAW, and CELL SPE, and the TI C64x processor scaled to the same technology.

real layouts. The coarse grain 2-core Itanium [13] contains large wide-issue processors each close to 300 mm^2 in 90 nm technology, while the fine grain AsAP contains single-issue processors each less than 1 mm^2 in 0.18 μm technology. Size differences of tens and hundreds make strong impacts on system behavior.

Most chip multiprocessors target a broad range of applications, and each processor in such systems normally contains powerful computational resources—such as large memories, wide issue processors [13], and powerful inter-processor communication [35]—to support widely varying requirements. Extra computational resources can enable systems to provide high performance to a diverse set of applications, but they reduce energy efficiency for tasks that can not make use of those specialized resources. Most DSP applications (targets of the AsAP) are made up of computationally intensive tasks with very small instruction and data kernels, which makes it possible to use extremely simple computational resources—small memory, simple single issue datapath, and nearest neighbor communication—to achieve high energy efficiency while maintaining high performance.

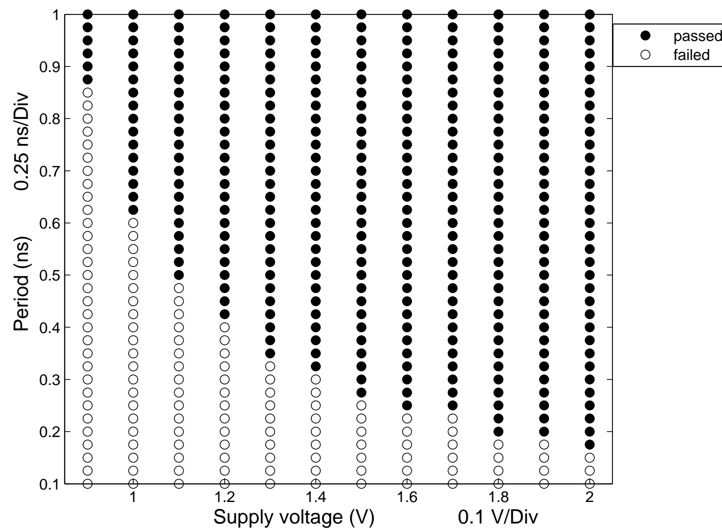

Figure 5.3: Processor shmoo: voltage vs. speed

5.1.2 High speed

Small memories and simple processing elements enable high clock frequencies. The fabricated processors run at 520–540 MHz at a supply voltage of 1.8 volts. The shmoo plot of Fig. 5.3 shows processor operation as a function of supply voltage and clock speed. The AsAP processor operates at frequencies among the highest possible for a digital system designed using a particular design approach and fabrication technology. The clock frequency information listed in Table 5.3 supports this assertion.

5.1.3 High peak performance and low average power consumption

High clock speed and small areas result in a high peak performance with a fixed chip size; also its average power consumption is low due to its simple architecture. Figure 5.4 compares the peak performance density and energy efficiency of several processors [23, 122, 123, 35, 7]. All data are scaled to 0.13 μm technology. Energy efficiency is defined as the power divided by the clock frequency with a scale factor to compensate for multiple issue architectures. These processors have large differences that are not taken into account by these simple metrics—such as word width and workload—so this comparison is only approximate. The AsAP processor has a high peak

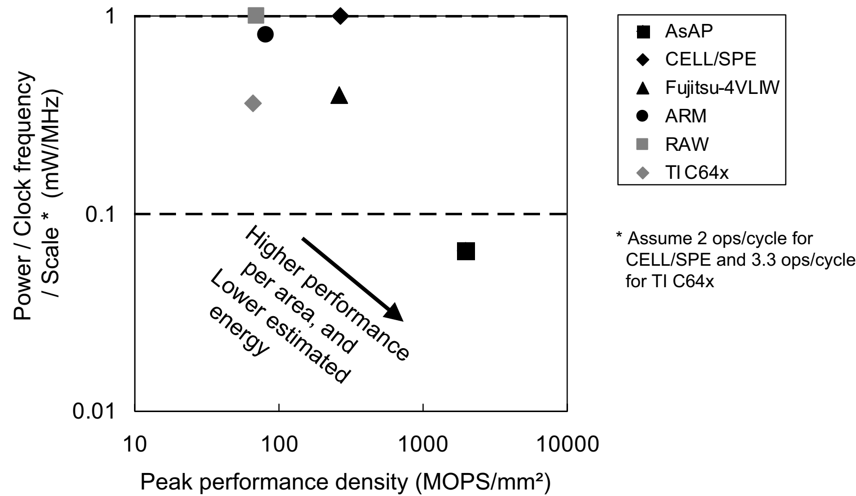

Figure 5.4: Power and performance evaluation of AsAP processor and several other processors; with technology scaled to 0.13 μm

Table 5.2: Estimates for a 13 mm $\times$ 13 mm AsAP array implemented in various semiconductor technologies

CMOS Tech (nm)	Processor Size (mm^2)	Num Procs per Chip	Clock Freq (GHz)	Peak System Processing (Tera-Op)
180	0.66	256	0.53	0.13
130	0.34	500	0.73	0.36
90	0.16	1050	1.06	1.11
45	0.04	4200	2.12	8.90

performance density that is 7 to 30 times higher than the others. Also, the AsAP processor has a low power per operation that is 5 to 15 times lower than the others.

With advancing semiconductor fabrication technologies, the number of processors will increase with the square of the scaling factor and clock rates will increase approximately linearly—resulting in a total peak system throughput that increases with the *cube* of the technology scaling factor. Table 5.2 summarizes the area and performance estimates for several technologies with the corresponding peak performance. It shows that in 90 nm technology, an AsAP array can achieve 1 Tera-op/sec with a 13 mm$\times$13 mm chip, but dissipates only 10 W typical application power in addition to leakage. Real applications would unlikely be able to sustain this peak rate, but tremendous throughputs are nonetheless expected.

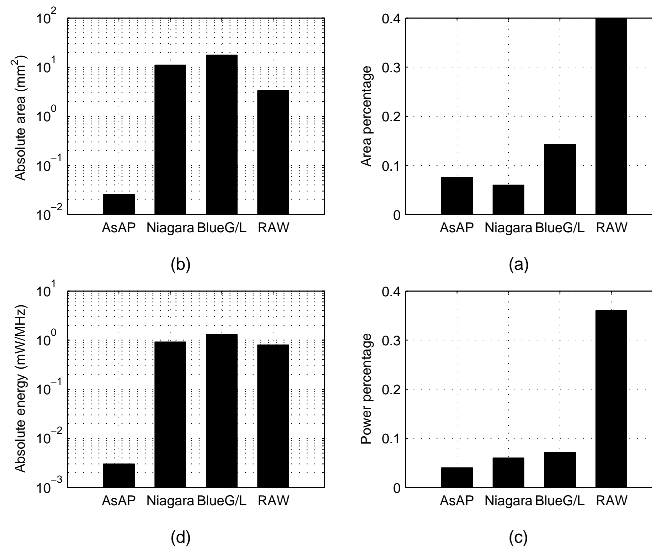

Figure 5.5: Comparison of area and power consumption for the communication circuit of four chip multiprocessors: (a) absolute comm. area; (b) area percentage of comm. circuit to processor; (c) absolute comm. circuit power consumption; and (d) power consumption percentage of comm. circuit to processor. Values are scaled to 0.13 $\mu\mathrm{m}$ technology.

5.1.4 Result of communication circuitry

Nearest neighbor communication simplifies the inter-processor circuitry and two dual-clock FIFOs present the major cost in this case, which results in high area and energy efficiencies. Figure 5.5 compares AsAP to three other chip multiprocessors [26, 25, 124]. The communication circuitry in the AsAP processor occupies less than 0.08 mm^2 in 0.18 $\mu\mathrm{m}$ CMOS, which is approximately 8% of the processor area, and is more than 100 times smaller than the others. Under the worst case conditions when maximizing possible communication, the communication circuity in the AsAP processor consumes around 4 mW at 470 MHz, which is about 4% of the processor power and 200 times lower than the others.

5.2 High performance and low power consumption for DSP applications

Table 5.3 lists area, performance, and power data for a number of general-purpose (MIPS VR5000 [53, 125], NMIPS [126], ARM [127]), programmable DSP (TI C62x [53, 125], PipeRench [95]),

Table 5.3: Area, performance and power comparison of various processors for several key DSP kernels and applications; all data are scaled to 0.18 μm technology assuming a $1/s^2$ reduction in area, a factor of s increase in speed, and a $1/s^2$ reduction in power consumption. The area is the chip core area when available.

Benchmark	Processor	Scaled area (mm^2)	Scaled clock freq. (MHz)	Clock cycles	Scaled execution time (ns)	Scaled power (mW)	Scaled energy (nJ)
40-tap FIR	**AsAP (8 proc.)**	**5.28**	**530**	**10**	**19**	**803**	**15**
	MIPS VR5000	N/A	347	430	1239	2600	3222
	TI C62x	>100	216	20	92	3200	296
	PipeRench	55	120	2.87	24	1873	45
8x8 DCT	**AsAP (8 proc.)**	**5.28**	**530**	**254**	**479**	**402**	**192**
	NMIPS	N/A	78	10772	137000	177	24400
	CDCT6	N/A	178	3208	18000	104	1950
	TI C62x	>100	216	208	963	3200	3078
	DCT processor	1.72	555	64	115	520	60
Radix-2 complex 64-pt FFT	**AsAP (13 proc.)**	**8.6**	**530**	**845**	**1593**	**759**	**1209**
	MIPS VR5000	N/A	347	15480	44610	2600	115988
	TI C62x	>100	216	860	3981	3200	12739
	FFT processor	3.5 (core)	27	23	852	43	37
JPEG encoder (8x8 block)	**AsAP (9 proc.)**	**5.94**	**300**	**1443**	**4810**	**224**	**1077**
	ARM	N/A	50	6372	127440	N/A	N/A
	TI C62x	>100	216	840	3900	3200	12400
	ARM+ASIC	N/A	50	1023	20460	N/A	N/A
802.11a/g Trans. (1 symbol)	**AsAP (22 proc.)**	**14.52**	**300**	**4000**	**13200**	**407**	**5372**
	TI C62x	>100	216	27200	126800	3200	405760
	Atheros	4.8 (core)	N/A	N/A	4000	24.2	96.8

and ASIC (DCT processor [128], FFT processor [129], ARM+ASIC [127], Atheros [130]) processors for which their data could be obtained. The TI C62x was chosen as the reference programmable DSP processor since it belongs to the TI VLIW C6000 series, which is TI's highest performance series. The enhanced TI C64x VLIW DSP processor [7] is also in the C6000 series and has an architecture similar to the C62x, but it contains substantial circuit level optimizations that achieve more than 4 times higher performance with less than half the power consumption compared to the C62x. The C62x should be a fair comparison with the first version AsAP processor and thus a better comparison at the architecture level, without tainting from circuit level optimizations.

In support of the assertion that the AsAP prototype has significant room for improvement, there is a fact that measurements show approximately 2/3 of AsAP's power is dissipated in its clocking system. This is largely due to the fact that clock gating was not implemented in this first

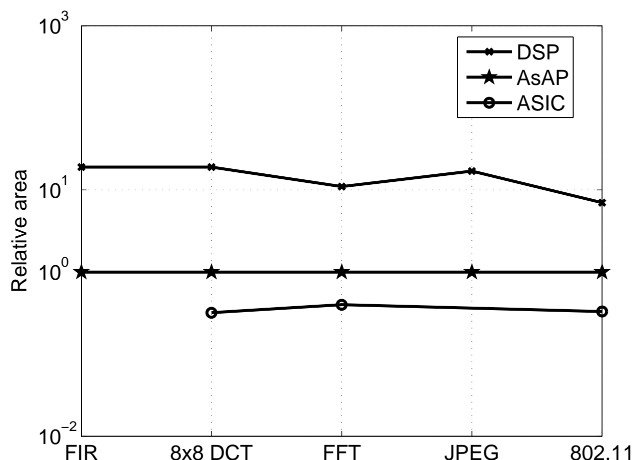

Figure 5.6: Relative area for various implementations of several key DSP kernels and applications. Source data are available in Table 5.3.

prototype. All circuits within each processor are clocked continuously—except during idle periods when the oscillator is halted. The future addition of even coarse levels of clock gating (distinguished from oscillator halting) are expected to significantly reduce power consumption further.

The area used by AsAP, shown in Table 5.3, is the combined area required for all processors including those used for communication. Data for the FIR, 8×8 DCT, and FFT are deduced from measured results of larger applications. The performance of the JPEG encoder on the TI C62x was estimated by using the relative performance of the C62x compared to MIPS processors [125], and a reported similar ARM processor [127].

Figure 5.6, 5.7, and 5.8 compares the relative performance and power of an AsAP processor to other processors in Table 5.3. These comparisons make use of 8, 8, 13, 9, and 22 processors respectively. A larger numbers of processors (through parallelization) would increase performance further. AsAP achieves 26–286 times higher performance and 96–215 higher energy efficiency than RISC processors (single issue MIPS and ARM). AsAP also achieves 0.8–9.6 times higher performance and 10–75 times higher energy efficiency than high-end programmable DSPs (TI C62x). For ASIC implementations, AsAP achieves performance within a factor of 2–4.2 and energy efficiency within a factor of 3–50 with an area within a factor of 2.5–3. As a whole, RISC processors do not compare well, but this is to be expected since they are not optimized for DSP applications.

Another source of AsAP's high energy efficiency comes from its haltable clock, which

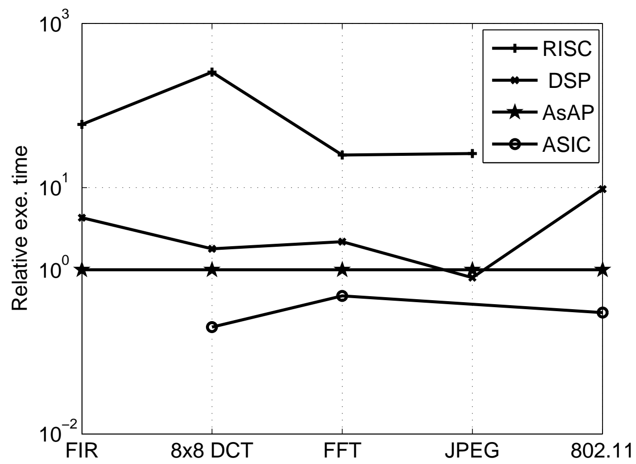

Figure 5.7: Relative execution time for various implementations of several key DSP kernels and applications. Source data are available in Table 5.3.

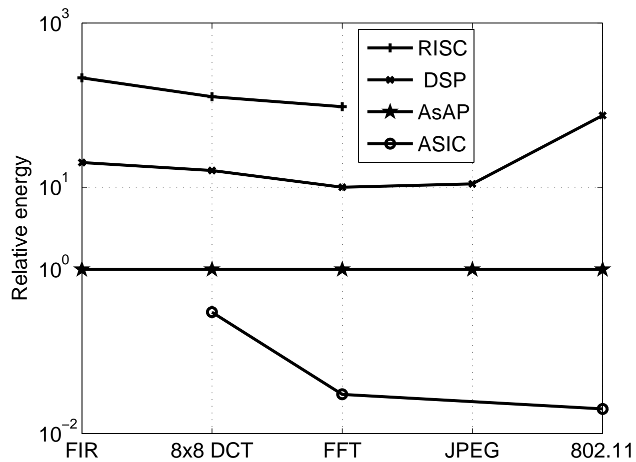

Figure 5.8: Relative energy for various implementations of several key DSP kernels and applications. Source data are available in Table 5.3.

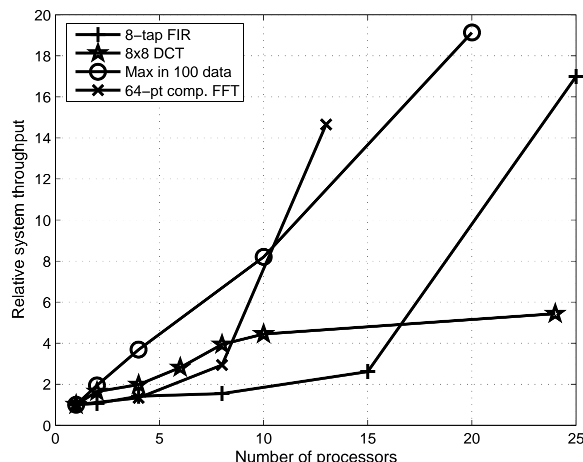

Figure 5.9: Increase in system throughput with increasing number of processors

is greatly aided by the GALS clocking style. Halting clocks while processors are momentarily inactive results in power reductions of 53% for the JPEG core and 65% for the 802.11a/g baseband transmitter.

Supply voltage scaling can be used to further improve power savings in very low power applications. Processors dissipate an average of 2.4 mW at a clock rate of 116 MHz using a supply voltage of 0.9 V while executing the described applications.

5.2.1 Performance scaling with the processor number

Figure 5.9 shows how throughput scales for four single tasks relative to the throughput of a single processor. Programs were written in assembly by hand but are lightly optimized and unscheduled. The memory requirement for the 8×8 DCT and 64-pt complex FFT exceeds the available memory of a single AsAP processor, so data points using one processor are estimated assuming one single processor had a large enough memory. An analysis of scaling results from a 16-tap FIR filter implemented in 85 different designs using from 1–52 processors shows a factor of 9 variations in throughput per processor over this space [131].

When all processors have a balanced computation load with little communication overhead, the system throughput increases close to linearly with the number of processors, such as for the task of finding the maximum value of a data set (*Max in 100 data* in Fig. 5.9). Clearly, ap-

plications that are difficult to parallelize show far less scalability at some point. For example, the performance of the 8×8 DCT increases well up to 10 processors where 4.4 times higher performance is achieved, but after that, little improvement is seen and only 5.4 times higher performance is seen using 24 processors. However, there is significant improvement in the FIR filter and FFT after a certain number of processors is reached. The reason for this is because increasing the number of processors in these applications avoids extra computation in some cases. For example, the FFT avoids the calculation of data and coefficient addresses when each processor is dedicated to one stage of the FFT computation. On average, 10 processor and 20 processor systems achieve more than 5 times and 10 times higher performance compared to a single processor system, respectively.

5.3 Summary

The AsAP scalable programmable processor array is designed for DSP applications and features a chip multiprocessor architecture, simple architecture with small memories, GALS clocking style, and nearest neighbor communication. These and other features make AsAP well-suited for future fabrication technologies, and for the computation of complex multi-task DSP workloads.

The AsAP processing array is implemented in $0.18\ \mu\mathrm{m}$ CMOS, and runs at 520–540 MHz at 1.8 V. It is highly energy efficient and each processor dissipates 36 mW while executing applications, and 94 mW when 100% active. It achieves a high performance density of 530 Mega-ops per 0.66 mm^2.

Chapter 6

System Feature Analysis: GALS vs. Synchronous

The effect of GALS style to the system features is investigated in this chapter. GALS chip multiprocessors under the right conditions like AsAP can hide much of the GALS performance penalty and at the same time, take full advantage of its scalability and high energy efficiency. Along with a thorough investigation of GALS effects on system performance, it is shown that such GALS chip multiprocessors have small performance penalties compared to corresponding synchronous systems. Furthermore, the small performance penalty can be completely eliminated by using sufficiently large FIFOs for inter-processor communication and programming without multiple-loop communication links. Scalability is enhanced due to the lack of a need for a global clock tree. In additional, the potential energy savings from joint adaptive clock and supply voltage scaling is increased in the common situation when workloads have highly unbalanced computational loads for each processor, thereby increasing the the probability processors can be tuned to save power. This work is distinguished from previous GALS multiprocessor evaluations [132] by not restricting the analysis to systems with global communication schemes.

After investigating several key design choices which impact the behavior of GALS systems in Section 6.1, Section 6.2 introduces the simulation platform. Section 6.3 investigates the effect of the asynchronous communication penalty to the performance of this GALS chip multiprocessor. Its scalability is shown in Section 6.4 compared to the corresponding synchronous system.

Section 6.5 investigates the power efficiency of this GALS chip multiprocessor when using adaptive clock/voltage scaling.

6.1 Exploring the key GALS chip multiprocessor design options

Several design options impact the behavior of GALS chip multiprocessors. This section presents the three most fundamental parameters: the clock domain partition, asynchronous boundary communication, and the inter-processor network.

Using dual-clock FIFO to reliably and efficiently move data across asynchronous clock boundaries is a key component in GALS systems and is already discussed in Section 2.2.2. Furthermore, as will be shown in Section 6.3.3, the relatively larger buffer has the benefit of hiding some communication latency. Dual-clock FIFOs introduce extra communication delay compared to a corresponding synchronous design, and exact values vary depending on many circuit, fabrication technology, and operating condition variables.

6.1.1 Clock domain partition of GALS chip multiprocessors

The most basic issue in designing a GALS chip multiprocessor system is how to partition the system into multiple clock domains. One method is to partition each processor into a single clock domain. A more fine grain method is to partition each processor into several clock domains, which is called a *GALS uniprocessor*. A more coarse grain method is to group several processors together into a single clock domain. Figure 6.1 illustrates these three methods.

Partitioning each processor into several clock domains has been studied by several researchers [48, 133]. Its key advantage is providing opportunities to use clock/voltage scaling not only to individual processors, but also to modules inside each processor. Its primary disadvantage is a relatively significant performance penalty and design overhead.

Upadhyay et al. [134] investigated coarse granularity clock domain partitioning for a GALS chip multiprocessor. They compared power savings from the simplified clock tree versus the power overhead from the local clock generator plus asynchronous communication for different partition methods. Minimum power consumption of these function blocks can be achieved by partitioning the system according to their method, assuming no other power saving methods such as

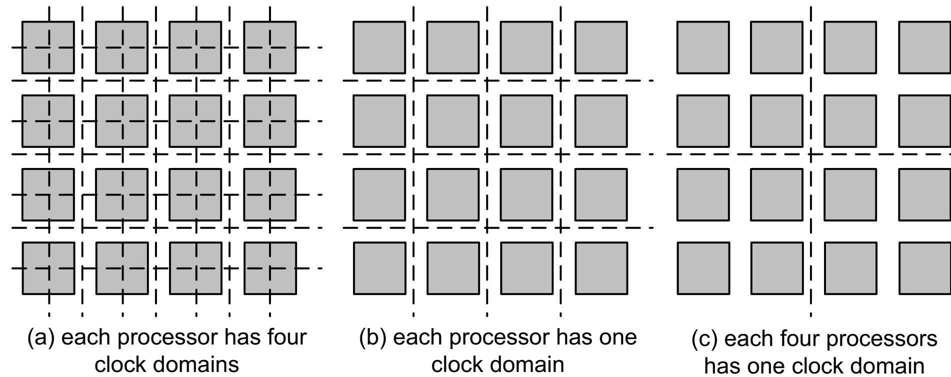

Figure 6.1: Three example clock domain partitions, from a sub-processor fine grain partition to a multi-processor coarse grain partition. The dotted lines show the clock domain partition boundaries. Analysis and results are independent of whether modules are homogeneous or heterogeneous.

clock and voltage scaling are used.

Placing each processor in its own clock domains is a simple but efficient strategy. Although power savings from simplified clock trees may not be able to compensate fully for overhead due to the local clock generator and asynchronous communication interface, it makes each processor highly uniform and simplifies the physical design. Furthermore, as shown in Section 6.5, it provides the flexibility to adaptively scale the clock and voltage for each processor which can achieve more than 40% power savings compared to a fully synchronous design.

6.1.2 Inter-processor network

The networking strategy between processors also strongly impacts the behavior of GALS chip multiprocessors.

Smith [132] proposes and analyzes a GALS chip multiprocessor with multiple processors and shared memory communicating through a global bus. This scheme provides very flexible communication, but places heavy demands on the global bus; thus the system performance is highly dependent on the level and intensity of the bus traffic. Furthermore, this architecture lacks scalability since increased global bus traffic will likely significantly reduce system performance under high traffic conditions or with a large number of processors.

A distributed network strategy such as "nearest neighbor" mesh distributes the interconnections across the whole chip and the communication load for each inter-processor link can be

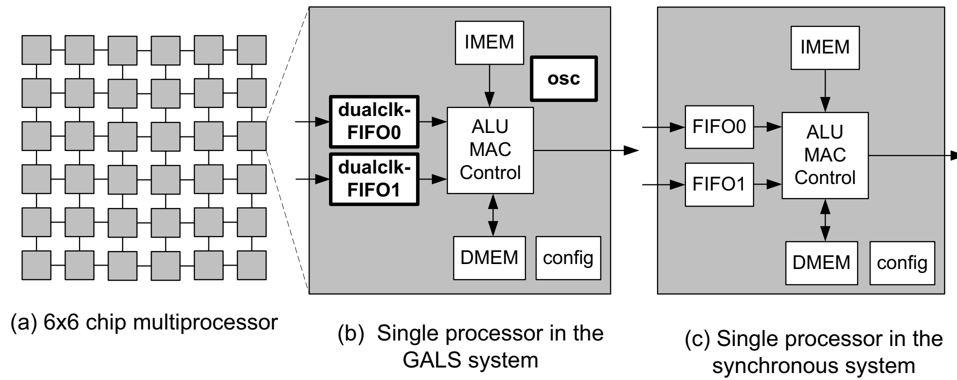

Figure 6.2: Two chip multiprocessors: one using a fully synchronous style and the other using a GALS style with per-processor clock domains

highly reduced. This architecture also provides perfect physical scalability due to its tile-based architecture without global wires.

6.2 Simulation platform — the GALS and non-GALS chip multiprocessors

The AsAP platform is used to illustrate principles presented in this chapter. A special feature of the design is that it provides a mode where all processors operate fully synchronously, thereby permitting an excellent testbed for a GALS versus non-GALS comparison.

Figure 6.2(b) shows a single processor in the GALS system. In the dual-clk FIFO, two synchronization registers are used in the experiments shown in section 6.3. The synchronous processor is shown in Fig. 6.2(c), and local clock oscillators are unnecessary since a global clock is provided to all processors. Processors' FIFOs are fully synchronous for this system. The synchronous chip multiprocessor is emulated using special configurations in the chip, and uses a global clock signal without synchronization registers between asynchronous boundaries inside the inter-processor FIFOs.

The extra circuitry for supporting the GALS clocking style—the local oscillator and logic in the FIFOs related to dual-clock operation—occupies only about 1% of the processor area. Considering the fact the GALS system has a simplified clock tree, the area difference between a GALS system and a synchronous-only system is negligible.

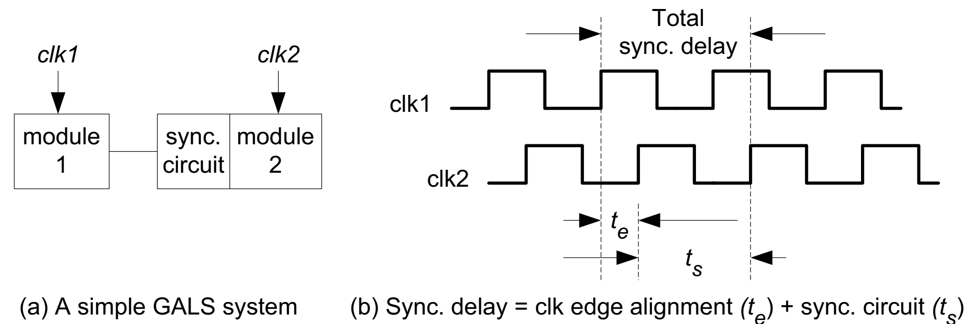

Figure 6.3: A GALS system boundary and timing of the synchronization delay accross the boundary

6.3 Reducing and eliminating GALS performance penalties

GALS systems require synchronization circuits between clock domains to reliably transfer data. Clock phase edge alignment time for unmatched clocks and synchronization circuitry introduces a synchronization delay as illustrated in Fig. 6.3. The delay normally results in a reduction of performance (throughput).

This section discusses in depth principles behind how GALS clocking affects system throughput and find several key architectural features which can hide the GALS effects. Fully avoiding any GALS performance penalties is possible for the described GALS chip multiprocessor. To simplify the discussions, in this section both GALS and synchronous systems use the same clock frequencies.

6.3.1 Related work

Significant previous research has studied the GALS uniprocessor—in which portions of each processor are located in separate clock domains. Results have shown GALS uniprocessors experience a non-negligible performance reduction compared to a corresponding synchronous uniprocessor. Figure 6.4 shows the control hazard of a simple DLX RISC processor [135] and can give an intuitive explanation for the performance reduction of a GALS uniprocessor. The upper subplot shows a branch penalty for a normal synchronous pipeline. The lower subplot shows a GALS uniprocessor's pipeline where each stage is in its own clock domain. During taken branch instructions, 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. Other pipeline hazards

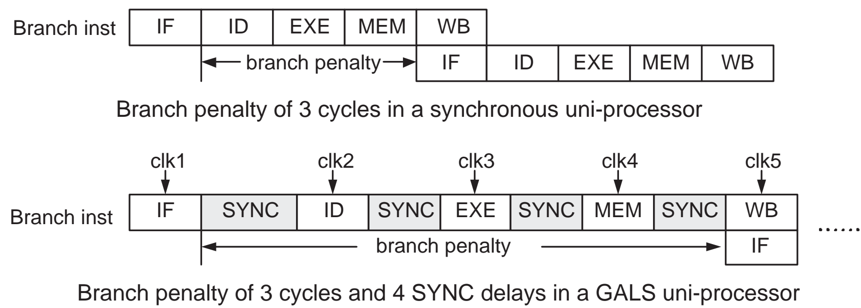

Figure 6.4: Pipeline control hazard penalties of a 5 stage synchronous uniprocessor and a 5 stage GALS uniprocessor

generate similar performance impacts. Studies of GALS uniprocessor performance have reported reductions of 10% [136] and 7%–11% [137] compared to fully synchronous designs. Semeraro et al. found that the performance penalty of GALS uniprocessor can be as low as 4% [48] by adjusting the synchronization logic to minimize the synchronization delay, and this performance degradation can be further reduce to 2% [138] by adding out-of-order superscalar execution features. However, minimizing the synchronization delay is generally not easy to implement, and fully avoiding the GALS performance penalty is still not achievable using this approach.

Other related work includes the performance analysis reported by Smith [132] for a GALS chip multiprocessor with a shared memory and a global bus. The GALS multiprocessor has additional latency when data transfers between processors and the shared memory through the global bus, thus introducing a performance penalty compared to the equivalent synchronous design. The reported performance penalty was typically higher than 5%, and varies across applications due to the different loadings of the global bus.

6.3.2 Comparison of application performance: GALS versus synchronous

The applications mentioned in Section 2.3 are mapped and simulated onto the verilog RTL model of our chip to investigate the performance of synchronous and GALS chip multiprocessor [46]. The RTL model is cycle-accurate with the fabricated chip and therefore exactly models its performance. The synchronous system uses a single global clock and has no synchronization registers in its communication boundaries. The GALS system uses a local oscillator and two synchronization registers inside each processor.

Table 6.1: Clock cycles (1/*throughput*) of several applications mapped onto a synchronous array processor and a GALS array processor, using 32-word FIFOs

	Synch. array (cycles)	GALS array (cycles)	GALS perf. reduction (cycles)	GALS perf. reduction (%)
8-pt DCT	41	41	0	0%
8×8 DCT	498	505	7	1.4%
zig-zag	168	168	0	0%
merge sort	254	254	0	0%
bubble sort	444	444	0	0%
matrix multiplication	817.5	819	1.5	0.1%
64-pt complex FFT	11439	11710	271	2.3%
JPEG encoder	1439	1443	4	0.3%
802.11 a/g TX	69700	69971	271	0.3%

The first two columns of Table 6.1 show the computation time in clock cycles when mapping these applications onto the synchronous and GALS chip multiprocessors. The third and fourth columns list the absolute and relative performance penalty of the GALS chip multiprocessor. 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 [136, 137], or the 5% performance reduction of the GALS chip multiprocessor with a shared memory and global bus [132].

6.3.3 Analysis of the performance effects of GALS

The very small performance reduction of the GALS chip multiprocessor motivates us to understand the factors that affect performance in GALS style processors. Figure 6.5 shows the chain of events that allow synchronization circuit latency to finally affect application throughput. It is a long path, so several methods are available to hide the GALS effect. To take a closer look at the effect in each block in Fig. 6.5:

- *Synchronization circuit latency* is inherent in every asynchronous boundary crossing due to mismatch in clock phases and overhead of synchronization circuits.
- *Average communication latency* takes into account the fact that the synchronization circuit path is normally not active every cycle. Thus, the synchronization latency should be weighted

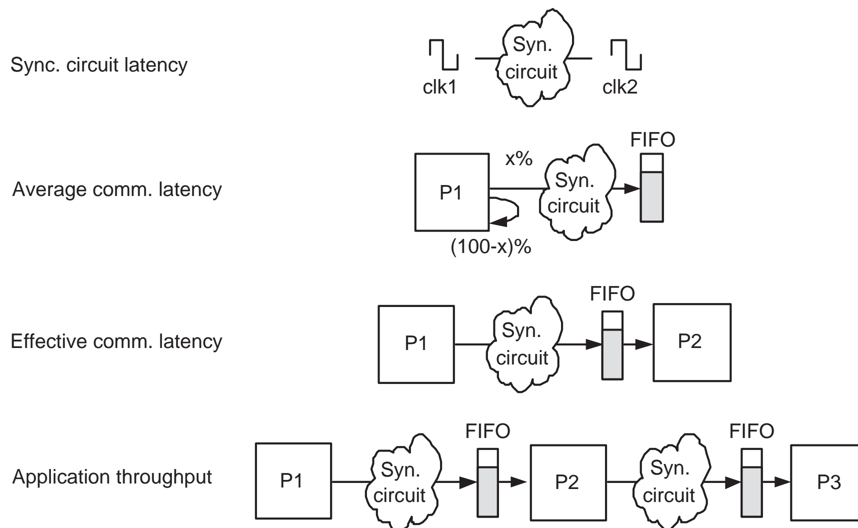

Figure 6.5: Illustration of three key latencies and application throughput in GALS multiprocessor systesms

by the fraction of the time the path is active ($x\%$ in Fig. 6.5), which is the average communication latency.

- *Effective communication latency* takes into account cases where the downstream processor does not consume data immediately after it is received. In other words, the average communication latency has its impact only when a program is waiting for these delayed signals, and the impact is zero otherwise.

- *Application throughput* is the metric of highest importance. The effective communication latency impacts application throughput only when there is communication feedback. As more fully shown in sec. 6.3.3, one-way communication does not result in throughput penalties under certain conditions.

The following three subsections discuss in further detail each step from the synchronization circuit latency to the application throughput, and show methods to hide the GALS performance effect.

Table 6.2: The fraction of the time the inter-processor communication is active for each processor executing several applications; P1–P9 represent processors 1–9 respectively

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Ave.
8-pt DCT	0.19	0.19	–	–	–	–	–	–	–	0.19
8×8 DCT	0.12	0.12	0.12	0.12	–	–	–	–	–	0.12
zig-zag	0.57	0.38	–	–	–	–	–	–	–	0.47
merge sort	0.06	0.02	0.05	0.03	0.03	0.06	0.03	0.03	–	0.04
bubble sort	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	–	0.03
matrix multip.	0.11	0.18	0.26	0.31	0.33	0.03	–	–	–	0.20
64-pt FFT	0.01	0.02	0.03	0.02	0.01	0.02	0.03	0.02	–	0.02
JPEG encod.	0.04	0.04	0.04	0.07	0.04	0.05	0.003	0.18	0.06	0.06

Hiding synchronization circuit latency by localizing computation

An obvious but nonetheless worthwhile point is that the asynchronous boundary circuit affects performance only when signals cross it, so the effect from this circuit latency can be highly decreased if data communication across the asynchronous boundary is reduced. This may initially sound like a difficult goal, but in fact key parameters such as clock domain granularity and application partitioning can easily affect this coefficient by more than an order of magnitude.

In many GALS systems the asynchronous boundaries have frequent traffic. For example, in a GALS uniprocessor each instruction crosses several asynchronous boundaries while flowing through the pipeline. But in a GALS chip multiprocessor, computation is much more localized at each single processor and communication through the asynchronous boundaries is less frequent and thus the effect of GALS is much lower.

Table 6.2 shows the fraction of the time the inter-processor communication path is active for several applications. The inter-processor communication is often infrequent, especially for complex applications such as the 64-point complex FFT and JPEG encoder that show communication time percentages of only 2% and 6% respectively.

Hiding average communication latency by FIFO buffering

Not every data word transfered across an asynchronous boundary results in effective latency. As Fig. 6.6(a) shows, the GALS system has the same computational flow as the synchronous system when the FIFO is neither empty nor full.

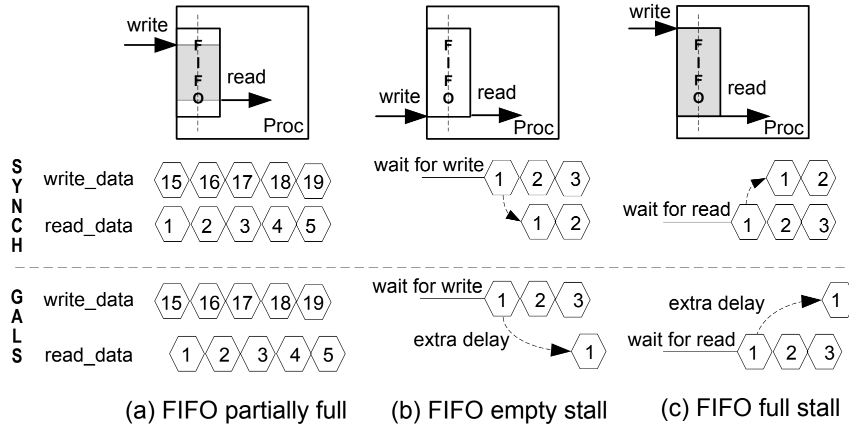

Figure 6.6: FIFO operation when FIFO is: (a) neither full nor empty, (b) empty, and (c) full. Correct operation requires delaying reads when the FIFO is empty, and delaying writes when the FIFO is full. Full speed operation is permitted when the FIFO is partially full.

FIFO stalls caused by full and empty conditions introduce extra latency in GALS systems. A *FIFO empty stall* occurs when a processor reads an empty FIFO and must wait (stall) until data is available, as illustrated in Fig. 6.6(b). In this case, reads in GALS systems have extra delay compared to synchronous systems. A *FIFO full stall* occurs when a processor writes a full FIFO and must wait until there is writable space. As shown in Fig. 6.6(c), writes in GALS systems have extra delay compared to synchronous systems.

Table 6.3 shows the effective latency difference of the GALS chip multiprocessor and its corresponding synchronous system over several applications. Due to the adequately large FIFO buffer of 32 words, there are relatively few resulting FIFO stalls, and the effective latency penalty of the GALS chip multiprocessor is small with an average of less than 2%. Interestingly, this latency penalty is larger than the throughput penalty—which is less than 1% as shown in Table 6.1.

The key point is this: latency penalties do not always result in throughput penalties. This result is further explained in the following subsection.

Throughput penalty caused by latency penalty from communication loops

Reading an empty FIFO or writing a full FIFO results in extra computation latency, but does not always reduce application throughput. Generally speaking, simple *one-way communication* does not affect system throughput, what really matters is *communication loops*—in which two units

Table 6.3: Effective latency (clock cycles) of several applications mapped onto a synchronous array processor and a GALS array processor, with 32-word FIFOs

	Synchronous array latency (cycles)	GALS array latency (cycles)	Difference (cycles)	Difference (%)
8-pt DCT	78	82	4	5%
8×8 DCT	996	1003	7	0.7%
zig-zag	221	224	3	1.3%
merge sort	542	552	10	1.8%
bubble sort	1966	1984	18	0.9%
matrix multiplier	1705	1720	15	0.8%
64-pt complex FFT	26741	27319	578	2.1%
JPEG encoder	2726	2741	15	0.5%
802.11a/g	115477	117275	1798	1.5%

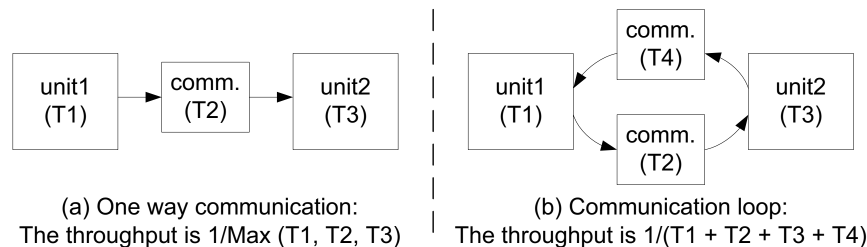

Figure 6.7: System throughput in a) one way communication path, and b) communication loop path. For a GALS system, *unit1* and *unit2* are assumed in different clock domains and therefor the *comm.* communication delays are significant. For both synchronous and GALS systems, throughout is not reduced with one-way communication (assuming communication time is less than computation time), but is reduced in the loop case.

wait for information from each other.

In a *one way communication path* as shown in Fig. 6.7(a), 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. 6.7(b). 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 uniprocessors. The GALS overhead

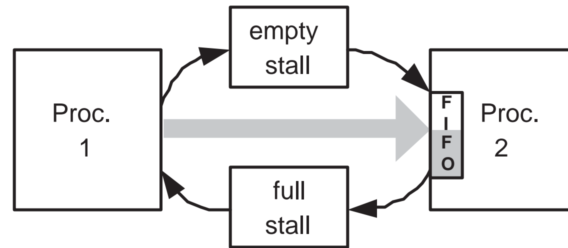

Figure 6.8: A common stall communication loop exists when the data producer *proc.1* and data consumer *proc.2* are alternatively busy and idle, the FIFO alternates between being empty and full, and processors stall appropiately. The wide arrow is the direction of data transfer, and thin arrows show how FIFO stalls are generated

increases the communication latency between pipeline stages. In instructions without pipeline hazards, the GALS uniprocessor maintains the same throughput as the synchronous uniprocessor although with larger latency, 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, and thus the GALS style brings a throughput penalty.

In our GALS chip multiprocessor, pure FIFO-full stalls or FIFO-empty stalls alone as in Fig. 6.6(b),(c) generate one way communication and have no effect on system throughput. Figure 6.8 shows the FIFO stall communication loop. Sometimes processor 1 is too slow and results in FIFO-empty stalls. Sometimes processor 2 is too slow and results in FIFO-full stalls. Coexisting FIFO-full stalls and FIFO-empty stalls (obviously at different times) in a link 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.

Results in Table 6.1 show that the GALS chip multiprocessor has nearly the same performance as the synchronous chip multiprocessor. This performance reduction is much less compared to the GALS uniprocessor's reduction. This implies that the performance penalty sources—the communication across asynchronous boundary, the FIFO stalls, and FIFO stall loops—are much smaller than the probability of a pipeline hazard in a uniprocessor. These results match well with our model. In Table 6.1, the 8-pt DCT, zig-zag, mergesort and bubblesort have no GALS performance penalties since they have only one-way FIFO stalls. The other applications have about 1% performance penalty on average due to FIFO stall loops.

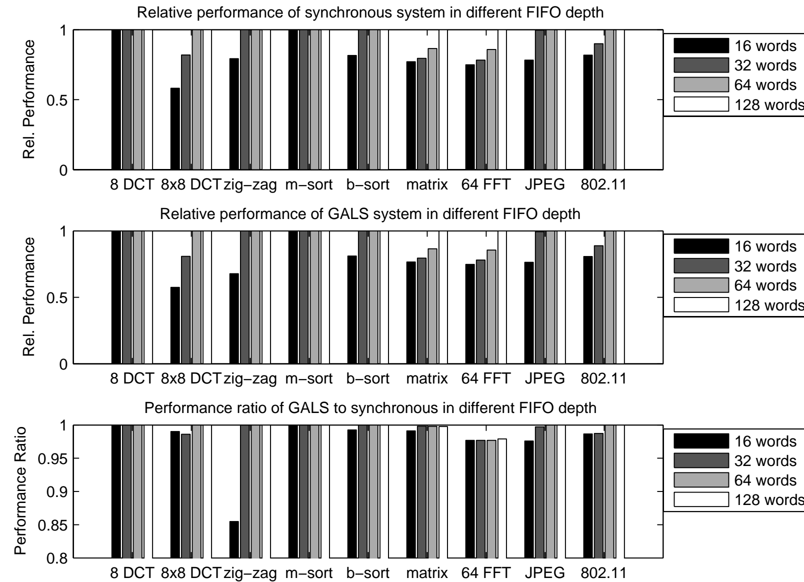

Figure 6.9: Performance of synchronous and GALS array processors with different FIFO sizes

6.3.4 Eliminating performance penalties

Increasing FIFO sizes

Increasing the FIFO size will reduce FIFO stalls as well as FIFO stall loops, and hence increase system performance and reduce the GALS performance penalty. With a sufficiently large FIFO, there will be no FIFO-full stalls and the number of FIFO-empty stalls can also be greatly reduced; then the communication loop in Fig. 6.8 will be broken and no GALS performance penalties will result.

The top and middle subplots of Fig. 6.9 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. 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 additional 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 802.11a/g 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.

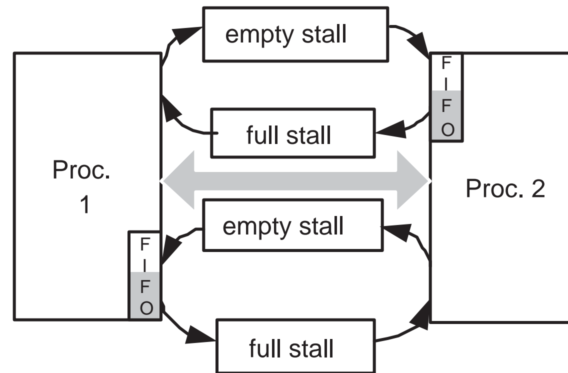

Figure 6.10: Examples of multiple-loop links between two processors

The bottom subplot of Fig. 6.9 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 FIFO stalls and FIFO stall loops. The ratio normally reaches 1.0 at the threshold, which means the FIFO stall 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 has a noticeable performance penalty of approximately 2%. This comes from the multiple-loop links and will be explained in the following subsection.

Breaking multiple-loop links

Figure 6.10 shows the multiple-loop communication links. 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. When the FIFO is large enough, there will be no FIFO-full stalls, but FIFO-empty stalls can still occur. So, the likelihood of the communication loop in Fig. 6.10 is still possible since FIFO-empty stalls alone can generate a stall loop. In the FFT application, several processors are used as data storing (Memory) coprocessors and they send and receive data to computation (Butterfly) processors [139], as shown in Fig. 6.11, thus generate multiple-loop links.

In order to avoid any performance penalties, programmers must avoid these types of multiple-loop link implementations. For example, in the FFT case, the *Memory* processor and *Butterfly* processor can be combined into one processor.

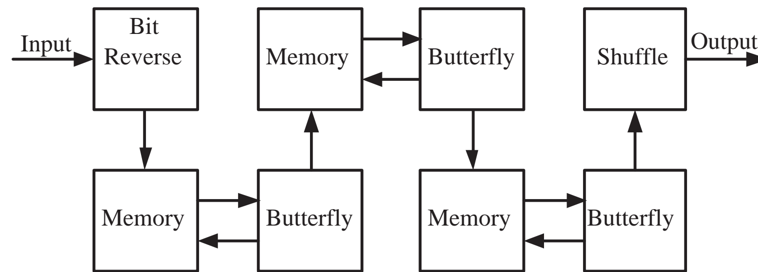

Figure 6.11: 64-pt complex FFT implementation containing multiple-loop links [139]

6.4 Scalability analysis of GALS chip multiprocessors

One of the key benefits of a GALS chip multiprocessor with distributed communication is its scalability: it allows simply inserting processors into the chip to expand an array. For a synchronous system, the clock tree has to be redesigned for different processor arrays and the design difficulty increases quickly along with the chip size. In addition, the clock skew normally becomes larger due to the more complex clock tree and more circuit parameter variation effects.

6.4.1 Auto generated clock trees for different sizes of chip multiprocessors

Multiple clock trees are generated using Cadence Encounter with an Artisan 0.18 μm standard cell library. An example clock tree for a single processor is shown in Fig. 6.12. It uses 37 buffers arranged in 3 levels, has 47 ps worst-case clock skew, around 555 ps delay, 120 ps buffer transition time and 97 ps sink transition time. The clock tree *delay* is the time from clock root to registers, the *buffer transition time* and *sink transition time* are the signal rise time at the inserted buffers and clocked registers respectively. The constrained parameters for the clock tree include: 50 ps clock skew, 2000 ps delay, and 120 ps buffer and sink transition times.

As the number of processors increases, the synchronous global clock tree becomes more complex and therefore more difficult to design. Table 6.4 lists several key parameters of clock trees for arrays made up of different numbers of processors. In general, all listed parameters tend to increase as the number of processors increases. However, in some cases, the parameters do not increase monotonically. This is due to the complex nature of the clock tree optimization problem which is affected by many discrete parameter (e.g., there can be only an integer number of buffer

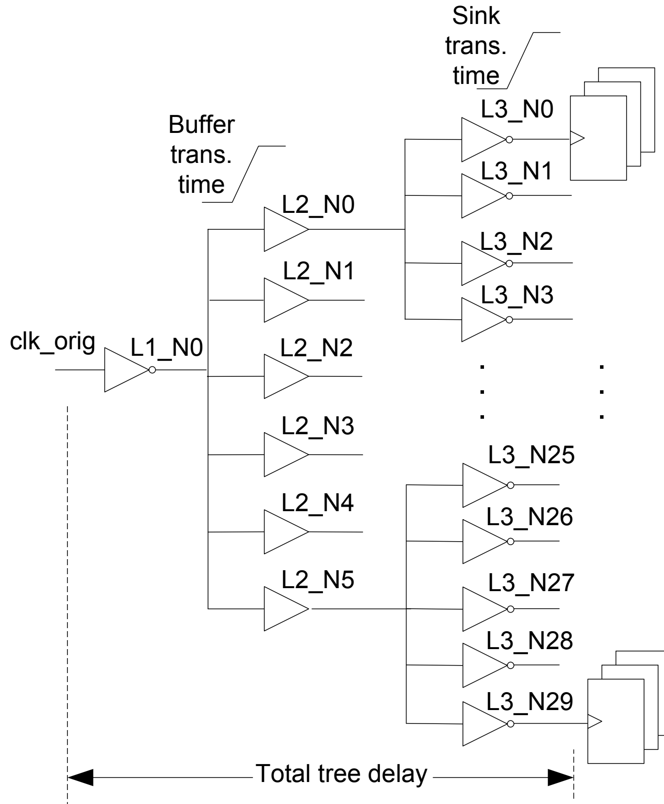

Figure 6.12: An example clock tree for a single processor

levels), and also by the time-limited non-optimal Cadence Encounter optimization runs. For example, 10 × 10 array has lower skew than the 9 × 9 array, but is also has a significantly higher Buffer Transition time and Sink Transition time—which likely come from increased load of buffers since they both have 13 levels of buffers.

Different design methods may have different results from our experiments. Using full custom design and deskewing technologies [140] can certainly generate better results but will dramatically increase the design effort, and also may increase the area and power consumption. For example, active deskewing and scan-chain adjustments used for Itanium [141] produced a clock tree with skew less than 10 ps, but it consumed around 25 W for clock generation and distribution circuits. Whichever methods are used, the trend that larger chip sizes result in more complex clock trees and larger clock skew, should always be true; and furthermore, impacts from circuit parameter variations are also increasing.

Table 6.4: Data for globally synchronous clock trees with different number of processors in the array

Processor array size	Num of buffer levels	Buffer trans. time (ps)	Sink trans. time (ps)	Total num of buffers	Max. skew (ps)	Total tree delay (ps)
1 × 1	3	120	97	37	47	555
2 × 2	6	120	115	157	49	1081
3 × 3	8	120	117	320	49	1150
4 × 4	10	121	133	633	59	1400
5 × 5	9	177	123	787	70	1700
6 × 6	10	174	143	1097	72	1800
7 × 7	12	170	171	1569	85	1900
8 × 8	14	251	119	1992	92	2100
9 × 9	13	204	119	3012	103	2200
10 × 10	13	228	141	3388	97	2300
11 × 11	15	169	120	3762	116	2450

6.4.2 The effect of clock tree on system performance

The clock period of modern processors expressed as fanout-of-4 (FO4) delay normally ranges between 10 to 20 [10], and is determined by the pipelining of the processor as well as clock distribution factors such as clock skew and jitter. Each FO4 delay in 0.18 μm is about 65 ps. Assuming the maximum logic delay (including register setup time and clock-to-Q time) within one clock period is 1000 ps, which is 15 FO4 delays and is in the range of modern high performance processor designs [4], the effect of clock tree design on system performance can be investigated. For the example of a single processor with 47 ps of clock skew, the smallest clock period (fastest clock speed) available will be 1047 ps. The relative system peak performance for different processor arrays is shown in Fig. 6.13. The peak performance of a GALS chip multiprocessor increases linearly with the number of processors, and the synchronous chip multiprocessor behaves similarly when the number of processors is not very large. For example, when containing 49 processors (around 33 mm^2 in 0.18 μm technology), the performance of the synchronous array processor is 96.5% of the GALS array processor. The performance of the synchronous system increases slower when the number of processors is larger since its clock skew increases along with the chip size. When it contains 121 processors (around 80 mm^2 in 0.18 μm technology), the performance of the synchronous array processor is reduced to 93.8% of the GALS array processor.

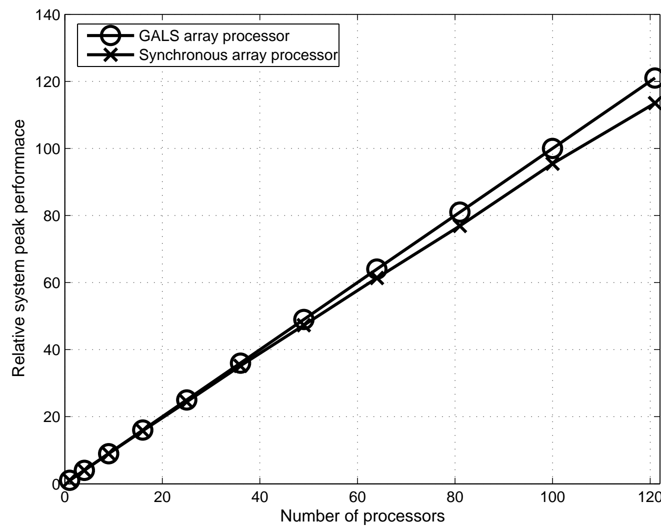

Figure 6.13: The peak performance of GALS processor arrays scale linearly with the number of processors, and they scale almost linearly for synchronous array processors, assuming the clock period is 15 FO4 delays for logic, plus clock skew only

The globally synchronous clock becomes worse when system parameter variations which are not included in the Encounter tool simulation are included. Parameter variations have increasingly impacted system performance along with advancing technologies. The main parameter variations include process variation, voltage variation and temperature variation [18]. These variations affect circuit delay and hence affect clock tree uncertainty. Different clock trees are affected differently by parameter variations. For example, clock trees with fewer inserted buffers [142] and fewer fanout loads [143] are less affected by parameter variations.

Supply voltage variation is one of the most important parameter variations. The simulation result from Encounter shows the peak voltage drop is 0.041 V for our single processor. There will be a larger voltage drop when the chip size increases, and the situation when the voltage is reduced from 1.8 V to 1.7 V due to the voltage variation is investigated. Figure 6.14 shows the increased clock tree delay at the reduced supply voltage for different clock trees by varying clock tree stages from 3 to 10, and clock buffer loads from 4 FO4 to 7 FO4. As shown in Fig. 6.14, the increased clock delay is around 5.5% of the original clock tree delay. It means the voltage drop (or variation) can increase the clock skew by 5.5% of the clock delay.

M. Hashimoto et al. [143] show that voltage variation accounts for approximately 40% of

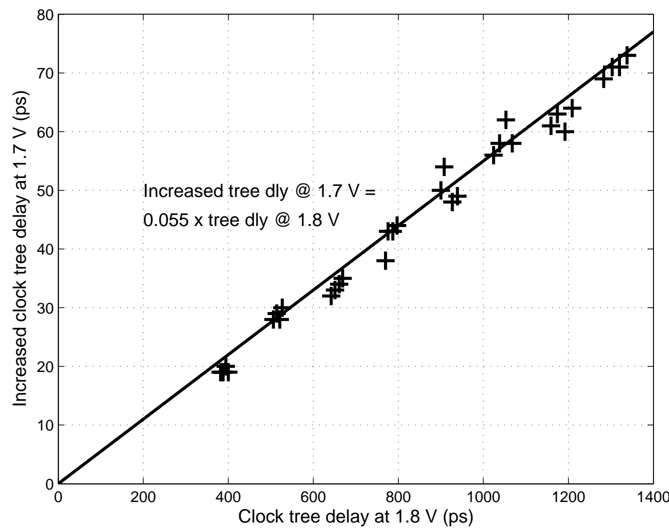

Figure 6.14: Increased clock tree delay from reducing supply voltage from 1.8 V to 1.7 V for different clock trees with varying total delays. Data is from transistor-level spice circuit simulations. Absolute clock skew (in ps) dramatically increases for lager clock trees.

the parameter variation effects among the primary sources: voltage variation, transistor variation, and temperature variation. According to the voltage variation effect discussed above, it is assumed that all parameter variations bring an extra clock skew which is 5.5%/0.4 = 13.75% of the clock delay. Figure 6.15 shows the performance scaling of GALS and synchronous chip multiprocessor along with the number of processors, including the parameter variation effect. It clearly shows that the GALS chip multiprocessor can achieve much better performance than the synchronous chip multiprocessor. When containing 49 processors, the performance of the synchronous multiprocessor is 82.7% of the GALS multiprocessor. This performance gap becomes larger as the number of processors increases. When containing 121 processors, the performance of the synchronous multiprocessor is only 76.8% of the GALS multiprocessor.

6.5 Energy efficiency analysis of adaptive clock frequency scaling

The GALS clocking style provides opportunities for using adaptive clock and voltage scaling for system submodules [144] and several researchers have reported its high energy efficiency for GALS uni-processors [136, 48, 137, 133]. The computation load in a GALS chip multiprocessor

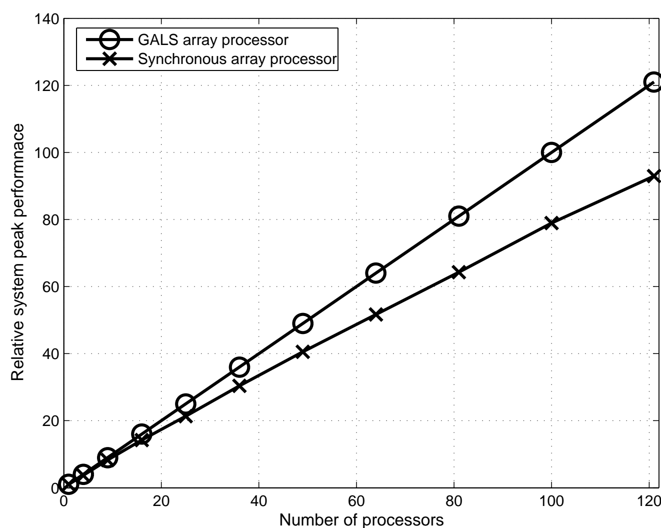

Figure 6.15: The peak performance of GALS and synchronous array processor where the clock period calculation includes: 1) the 15 FO4 logic delay, 2) static clock skew, and 3) clock skew equal to 13% of the total clock tree delay due to variations as shown in Fig. 6.14

can be quite unbalanced which potentially increases the energy efficiency benefit when using clock and voltage scaling for each processor. It is also found that it is possible to scale down the clock frequencies for some processors without reducing system performance whatsoever.

Some other clock styles such as rationally-related clocking used by Synchroscalar [98] can have similar benefits. While this approach avoids the extra hardware for asynchronous communication, it requires extra circuitry such as PLLs to guarantee the same clock phases between different clock domains and changing the clock frequency can take hundreds of clock cycles.

6.5.1 Related work—adaptive clock scaling of the GALS uniprocessor

Figure 6.16 shows an example of a GALS uniprocessor implementation to illustrate the concept of adaptive clock scaling in GALS systems, which achieves high energy efficiency by reducing the clock frequency and voltage of modules that are less heavily used. In the figure, the frequency of the Mem module clock *clk4* is reduced when executing the first code block since it has few Mem instructions. The scheme can be called *static clock/voltage scaling* if the clock/voltage won't be changed at runtime, which is the method addressed in this paper. In order to obtain greater power savings, *dynamic clock/voltage scaling* can be used where the clock and voltage are changed

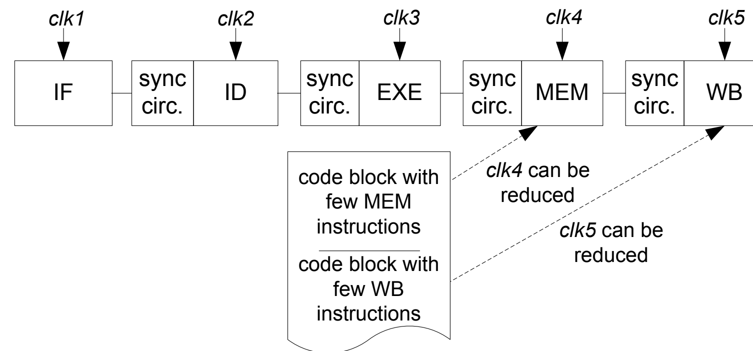

Figure 6.16: Clock scaling in a GALS uniprocessor

dynamically at runtime along with the program execution; then the reduced clock domain can become *clk5* when executing the second code block since it has few WB instructions. Unfortunately, reducing the clock of some modules for uniprocessors reduces system performance. The static scaling method reduces energy dissipation by approximately 16% with an approximately 18% reduction in performance [136]. The dynamic scaling method achieves 20%–25% energy savings along with a 10%–15% performance reduction [48, 137].

6.5.2 Unbalanced processor computation loads increases power savings 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 more opportunities to reduce the clock frequency and supply voltage of some processors to achieve further power savings without degrading system performance. 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 6.17 shows the unbalanced computational load among processors when mapping our applications onto the chip multiprocessor. The computational load difference for different processors in complex applications such as JPEG encoder and 802.11a/g transmitter can be more than 10 times.

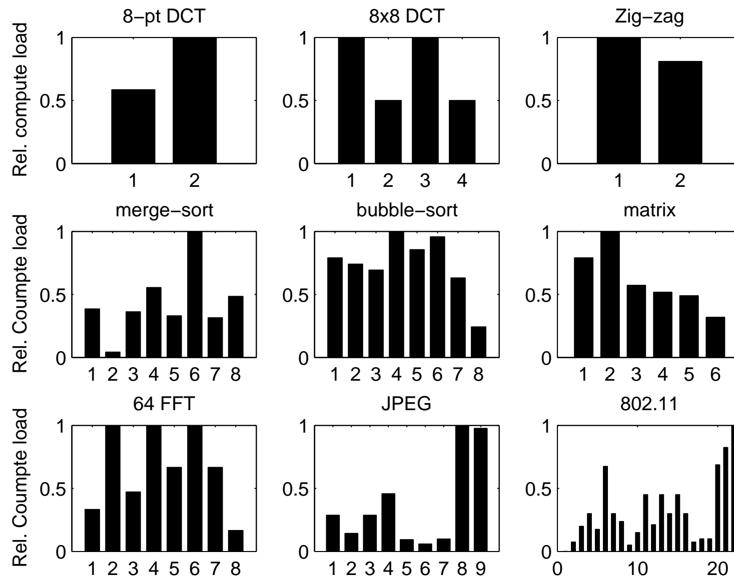

Figure 6.17: Relative computational load of different processors in nine applications illustrating unbalanced loads

6.5.3 Finding the optimal clock frequency—computational load and position

When using adaptive clock/voltage scaling, the system performance will be normally further reduced; but it is possible to scale the clock frequencies of some processors in the GALS multiprocessor while still maintaining the same performance. The optimal clock frequency for each processor depends strongly on its computational load, and also depends on its position and relationship with respect to other processors.

Figure 6.18 shows the system throughput versus the clock frequencies of four processors in the 8$\times$8 DCT, its implementation is shown in Fig 2.17. The computational load of the four processors is 408, 204, 408 and 204 clock cycles respectively. The throughput changes with the scaling of the 2nd and 4th processor much more slowly than the scaling of the 1st and 3rd 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 2nd and 4th processors have the same light computational load, but the throughput changes with the 4th processor scaling much more slowly than the 2nd processor's scaling. Minimal power consumption is achieved with full throughput when the relative clock frequencies are 100%, 95%,

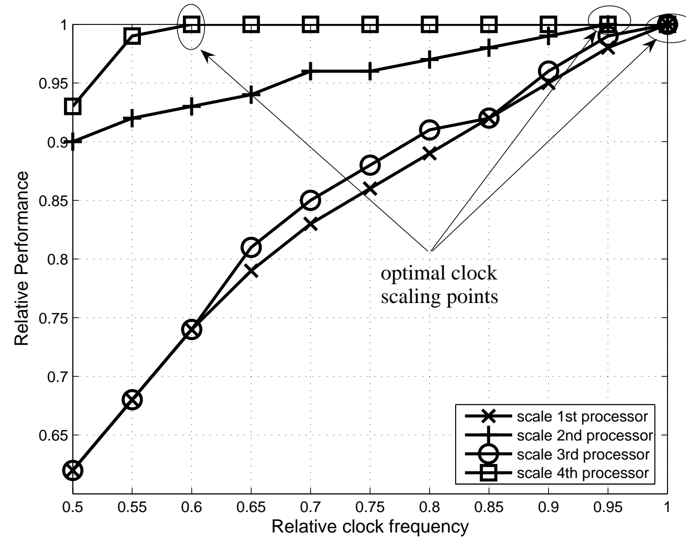

Figure 6.18: Throughput changes with statically configured processor clocks for the 4-processor 8×8 DCT application

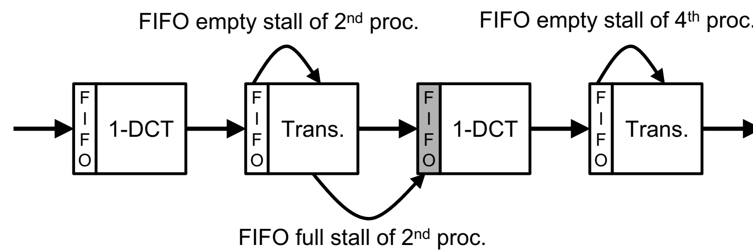

Figure 6.19: Relationship of processors in the 4-processor 8×8 DCT application illusting the differing scaling possibilities for the $2nd$ and $4th$ processors

100%, and 57% of full speed respectively.

The reason for the different behavior of the 2nd and 4th processors comes from their different positions and FIFO stall styles as shown in Fig. 6.19. The 2nd processor has FIFO-empty stalls when it fetches data too fast from the 1st processor, and it has FIFO-full stalls when it sends data too fast to the 3rd processor. The 4th processor has only FIFO-empty stalls.

6.5.4 Power reduction of clock/voltage scaling

Reducing the clock frequency allows for a reduction in voltage to obtain further power savings. The relationship between clock frequency, voltage and power has become much more

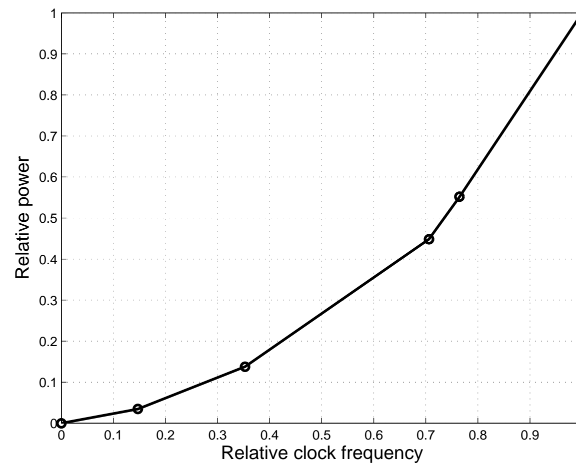

Figure 6.20: The relationship between a processor's power consumption with a varying clock frequency when the supply voltage is the minimum possible voltage for the clock speed [145]

complex in the deep submicron regime because of other parameters such as leakage power. A model derived from measured data from a 0.18 μm technology [145] is used to estimate power consumption. Figure 6.20 shows the relationship between clock frequency and its corresponding power consumption after using clock/voltage scaling.

Finding the optimal clock frequency for each processor as described in sec. 6.5.3, and using the frequency-power model, the relative power consumption of the GALS multiprocessor compared to the synchronous multiprocessor is estimated after using static clock frequency and supply voltage scaling for several applications. The result is shown in Fig. 6.21. The GALS system achieves an average power savings of approximately 40% without affecting performance. This power savings is much higher than the GALS uniprocessor which is reported to save approximately 25% energy when operating with a performance reduction of more than 10% [136, 48, 137].

6.6 Summary

It shows that the application throughput reduction of the GALS style comes from the asynchronous boundary communication and the communication loops, and that it is possible to design GALS multiprocessors without this performance penalty. A key advantage of the GALS chip multiprocessor with distributed interconnection compared to the other GALS systems is that

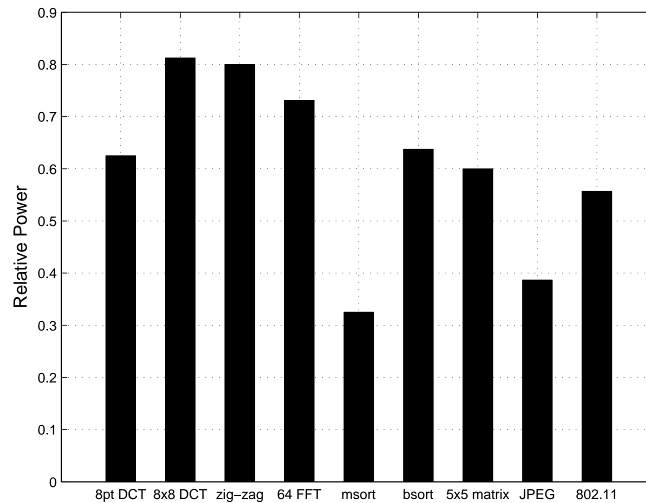

Figure 6.21: Relative power over several applications for the GALS array processor with static clock and supply voltage scaling compared to a synchronous array processor

its asynchronous boundary communication and 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% over a variety of DSP workloads, and this small penalty can be further avoided by large enough FIFOs and programming without multiple-loop communication links.

Local clock oscillators in GALS multiprocessors simplify the clock tree design and enable nearly perfect system scalability. More processors can be inserted into the chip without any clock tree redesign. As the number of processors increases, the clock skew in the synchronous multiprocessor system increases quickly due to the more complex clock trees and parameter variations. When containing 121 processors (around 80 mm^2 in 0.18 μm technology), the peak performance of the GALS multiprocessor can be approximately 20% higher than the synchronous multiprocessor.

Unbalanced computational loads in chip multiprocessors increases the opportunity for independent clock frequency and voltage scaling to achieve significant power consumption savings. The GALS chip multiprocessor can achieve around 40% power savings without any performance penalty over a variety of DSP workloads using static clock and voltage scaling for each processor. These results compare well with a reported 25% energy reduction and 10% performance reduction

of GALS uniprocessors for a variety of general purpose applications.

Data presented in this chapter are based on the fabricated AsAP chip. Results from this work apply to systems with three key features as discussed in section 6.1—namely, 1) multi-core processors (homogeneous and heterogeneous) operating in in independent clock domains; 2) source synchronous flow control for asynchronous boundaries communication; and 3) distributed interconnect, such as mesh. Which results certainly vary over different applications and specific architectures, systems with these features should still exhibit: good scalability, small performance reductions due to asynchronous communication overhead, and large power reductions due to the adaptive clock/voltage scaling; over many workloads.

Chapter 7

Conclusion

This dissertation discusses the architecture design, physical design, and feature analysis of a scalable programmable processor array for DSP applications.

The processor features a multi-core architecture, simple single core with small memories, GALS clocking style, and mesh internetwork. These and other features make the processor array highly energy-efficient and capable of high throughput, well-suited for future fabrication technologies, and for the computation of complex multi-task DSP workloads. It is promising that such systems can compete traditional DSPs to obtain 10 times higher performance with 10 times less power dissipation in 10 times smaller area.

GALS multi-core systems improve 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.

An asymmetric inter-processor communication architecture which uses more buffer resources for nearest neighbor connections and fewer buffer resources for long distance connections can save 2 to 4 times area compared to the traditional NoC while maintaining similar routing capability. Extending the source synchronization method can support the long distance GALS communication. Several design options are explored. Inserting two ports (buffers) for the processing

core, and using two or three links at each edge, can achieve good area/performance trade offs for multi-core systems organized by simple single issue processors; and the optimal number of links is expected to increase if the chip is organized by larger processors.

The application throughput reduction of the GALS style comes from the asynchronous boundary communication and the communication loops. A key advantage of the GALS multi-core systems with distributed interconnection compared to the other GALS systems is that its asynchronous boundary communication and 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% over a variety of DSP workloads, and this small penalty can be further avoided by large enough FIFOs and programming without multiple-link loops. Furthermore, the GALS array can achieve around 40% power savings without any performance penalty over a variety of DSP workloads using static clock and voltage scaling for each processor. GALS multi-core systems have nearly perfect system scalability. When containing 121 processors (around 80 mm^2 in 0.18 μm technology), the peak performance of the GALS multiprocessor can be approximately 20% higher than the synchronous multiprocessor whose clock skew increase quickly due to the more complex clock tree and parameter variation.

7.1 Future work

There are quite a few interesting research topics on many-core processors which are worthwhile for the further investigation.

Many of them are already implemented in our second version of AsAP chip. For example, supporting static/dynamic clock/voltage scaling for each individual processor using hardware and software; implementing the asymmetric double-links interconnect architecture as proposed in Chapter 3; adding large shared memory modules to support applications requiring memory more than AsAP 1 can provide [42]; adding some hardware accelerators such as FFT, Viterbi decoder, and H.264 motion estimation to broaden the application domain of AsAP platform, etc.

Below are some of the topics I can see in the next couple of years for the research of many-core processors.

- *The design of the optimal single processor.* It is found in AsAP project that the array orga-

nized by simple processors can achieve good performance and energy efficiency for many DSP applications. To answer the question that what is the optimal parameters for the single processor, such as the pipeline stage, instruction issue width, and instruction set, it requires more quantitative investigations. In addition, for different domains of applications, the answer is expected to be different.

- *On-chip interconnects*. In AsAP project it is found that simple static interconnect architecture is efficient for many DSP applications. For applications requiring complex communication, novel interconnection architectures are required to achieve better tradeoffs between flexibility and cost.

- *Clocking*. The GALS clocking, which is used in AsAP, is a promising clocking strategy for many-core processors. In the future, it is worthwhile to investigate the totally asynchronous clocking style. It has the potential to achieve higher speed and lower power consumption, although it is not a solid solution yet.

- *Power distribution*. How to distribute the power supply efficiently will be an important design issue in the future, due to the larger chip size and lower supply voltage. It is important not only for many-core processors, but also for other chips.

- *Fault tolerance computation*. Along with the larger chip size and lower supply voltage, the probability of failures in chips will increase significantly, and the fault tolerance computation is expected to be much more important. Many-core processors can be a good solution for the fault tolerance computation since it provides sufficient duplicated processors, and can use some of the processors for the failure checking and recovery.

Glossary

AsAP For *Asynchronous Array of simple Processors.* A parallel DSP processor consisting of a 2-dimensional mesh array of very simple CPUs clocked independently with each other.

BIPS For *billion instructions per second* which is used to indicate the processor performance

CMP For *Chip Multi-processor*, a computer architecture which integrates multiple processors into a single chip to improve processor performance.

CPI For *Cycles-per-instruction.* Normally the CPI for pipelined processor in larger than 1 due to the pipeline hazard or missed Cache fetch.

DCT For *Discrete Cosine Transform*, it is used to transforms a signal or image from the spatial domain to the frequency domain.

dual-clock FIFO A *First In First Out.* queue with a read clock and a write clock, used to synchronize data across a clock boundary.

DSP For *digital signal processing or the processors for DSP.*

FFT For *Fast Fourier transform*, an efficient algorithm to compute the discrete Fourier transform and its inverse.

FO4 For *Fanout 4.* A method to define the circuit delay using the delay of an inverter with 4 inverters load.

GALS For *Globally Asynchronous Locally Synchronous*. A design methodology in which major design blocks are synchronous, but interface to other blocks asynchronously.

H.264 A standard for video compression. It is also known as MPEG-4 part 10.

JPEG For *Joint Photographic Experts Group*, is a commonly used standard method of compression for photographic images.

Mbps For *Megabit per second*, a unit of data transfer rate.

MIMD For *Multiple instruction multiple data*, a parallel computer architecture where different instructions with different data can be executed at the same time.

MMX For *MultiMedia extension*, one of the SIMD techniques designed by Intel in 1990's in their Pentium microprocessor.

MTTF For *Mean Time to Failures*, a common measures of reliability.

NoC For *Network on Chip*. A in-chip communication architecture which communicates between modules in the chip using switches/routes, as in the network.

SIMD For *Single Instruction, Multiple Data*. A data parallelism technique where one single instruction can execute multiple data in parallel.

VLIW For *Very long instruction word*, a computer architecture which fetches multiple independent instructions at the same clock cycle to execute them in parallel, to improve the system performance.

Viterbi decoder An algorithm to decode a bitstream that has been encoded using forward error correction based on a convolutional code, developed by Andrew J. Viterbi in 1967.

Related publications

1. Zhiyi Yu, Michael Meeuwsen, Ryan Apperson, Omar Sattari, Michael Lai, Jeremy Webb, Eric Work, Tinoosh Mohsenin, Mandeep Singh, Bevan Baas, An Asynchronous Array of Simple Processors for DSP Applications, *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2006, pp:428-429.

2. Bevan Baas, Zhiyi Yu, Michael Meeuwsen, Omar Sattari, Ryan Apperson, Eric Work, Jeremy Webb, Michael Lai, Tinoosh Mohsenin, Dean Truong, Jason Cheung, AsAP: A Fine-grain Multi-core Platform for DSP Applications, *IEEE Micro*, March/April 2007, pp:34-45.

3. Ryan Apperson, Zhiyi Yu, Michael Meeuwsen, Tinoosh Mohsenin, Bevan Baas, A Scalable Dual-Clock FIFO for Data Transfers between Arbitrary and Haltable Clock Domains, *IEEE Transactions of Very Large Scale Integration Systems (TVLSI)*, October 2007, pp:1125-1134.

4. Michael Meeuwsen, Zhiyi Yu, Bevan Baas, A Shared Memory Module for Asynchronous Arrays of Processors, *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 86273, 13 pages, 2007.

5. Zhiyi Yu, Bevan Baas, Implementing Tile-based Chip Multiprocessors with GALS Clocking Styles, *IEEE International Conference of Computer Design (ICCD)*, October 2006, pp:174-180.

6. Bevan Baas, Zhiyi Yu, Michael Meeuwsen, Omar Sattari, Ryan Apperson, Eric Work, Jeremy Webb, Michael Lai, Daniel Gurman, Chi Chen, Jason Cheung, Dean Truong, Tinoosh Mohsenin, Hardware and Applications of AsAP: An Asynchronous Array of Simple Processors, *IEEE HotChips Symposium on High-Performance Chips, (HotChips)*, August 2006.

7. Zhiyi Yu, Bevan Baas, Performance and Power Analysis of Globally Asynchronous Locally Synchronous multi-processor systems, *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, March 2006, pp:378-384.

Bibliography

[1] N. Zhang and R. Brodersen. The cost of flexibility in systems on a chip design for signal processing applications. In *UCB EEC225C*.

[2] J. Wawrzynek. Reconfigurable computing. In *UCB CS294-3*, January 2004.

[3] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *International Symposium on Computer Architecture (ISCA)*, pages 14–24, May 2002.

[4] B. Flachs, S. Asano, S. H. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A streaming processing unit for a CELL processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 134–135, February 2005.

[5] A. Harstein and T.R. Puzak. Optimum power/performance pipeline depth. In *IEEE international Symposium on Microarchitecture (MICRO)*, pages 117–125, December 2003.

[6] G.E.Moore. Cramming more components onto integrated circuits. *Electronics Maganize*, April 1965.

[7] S. Agarwala, T. Anderson, A. Hill, M. D. Ales, R. Damodaran, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, A. Rajagopal, A. Chachad, M. Agarwala, J. Apostol, M. Krishnan, D. Bui, Q. An, N. S. Nagaraj, T. Wolf, and T. T. Elappuparackal. A 600-MHz VLIW DSP. *IEEE Journal of Solid-State Circuits (JSSC)*, pages 1532–1544, November 2002.

[8] R.P. Preston, R.W. Badeau, D.W. Balley, S.L. Bell, L.L. Biro, W.J. Bowhill, D.E. Dever, S. Felix, R. Gammack, V. Germini, M.K. Gowan, P. Gronowshi, D.B. Jankson adn S. Mehta, S.V. Morton, J.D. Pickholtz, M.H. Reilly, and M.J. Smith. Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 266–267, February 2002.

[9] K. Roy, S. Mukhopadyay, and H. Mahmoodi-meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, pages 305–327, February 2003.

[10] M. Horowitz and W. Dally. How scaling will change processor architecture. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 132–133, February 2004.

[11] S. Borkar. Low power design challenges for the decade. In *Asia and South Pacific Design Automatic Conference (ASP-DAC)*, pages 293–296, 2001.

[12] J. Stinson and S. Rusu. A 1.5 GHz third generation Itanium processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 252–253, February 2003.

[13] S. Naffziger, T. Grutkowski, and B. Stackhouse. The implementation of a 2-core multi-threaded Itanium family processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 182–183, 592, February 2005.

[14] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A dual-core multi-threaded Xeon processor with 16MB L3 cache. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 102–103, February 2006.

[15] H. D. Man. Ambient intelligence: Gigascale dreams and nanoscale realities. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 29–35, February 2004.

[16] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, pages 490–504, April 2001.

[17] International Roadmap Committee. International technology roadmap for semiconductors, 2005 edition. Technical report, ITRS, 2005. `http://public.itrs.net/`.

[18] 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)*, pages 338–342, June 2003.

[19] K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S. W. Williams, and K.A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, University of California, Berkeley, 2006.

[20] S. Kaneko, K. Sawai, N. Masui, et al. A 600 MHz single-chip multiprocessor with 4.8GB/s internal shared pipelined bus and 512kB internal memory. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 254–255, February 2003.

[21] T. Takayanagi, J. Shin, B. Petrick, J. Su, and A. Leon. A dual-core 64b ultraSPARC microprocessor for dense server applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2004.

[22] T. Shiota, K. Kawasaki, Y. Kawabe, W. Shibamoto, A. Sato, T. Hashimoto, F. Hayakawa, S. Tago, H. Okano, Y. Nakamura, H. Miyake, A. Suga, and H. Takahashi. A 51.2 GOPS 1.0 Gb/s-DMA single-chip multi-processor integrating quadruple 8-way VLIW processors. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 194–195, February 2005.

[23] 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)*, pages 184–185, February 2005.

[24] J. Hart, S. Choe, L. Cheng, C. Chou, A. Dixit, K. Ho, J. Hsu, K. Lee, and J. Wu. Implementation of a $4th$-generation 1.8GHz dual-core SPARC v9 microprocessor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 186–187, February 2005.

[25] A. Bright, M. Ellavsky, A. Gara, R. Haring, G. Kopcsay, R. Lembach, J. Marcella, M. Ohmacht, and V. Salapura. Greating the BlueGene/L supercomputer from low-power SoC AISCs. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 188–189, February 2005.

[26] A. Leon, J. Shin, K. Tam, W. Bryg, F. Schumachier, P. Kongetira, D. Weisner, and A. Strong. A power-efficienct high-throughput 32-thread SPARC processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 98–99, February 2006.

[27] M. Golden, S. Arekapudi, G. Dabney, M. Haertel, S. Hale, L. Herlinger, Y. Kim, K. McGrath, V. Palisetti, and M. Singh. A 2.6GHz dual-core 64b x86 microprocessor with DDR2 memory support. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2006.

[28] E. Cohen, N. Rohrer, P. Sandon, M. Canada, C. Lichtenau, M. Ringler, P. Kartschoke adn R. Floyd, J. Heaslip, M. Ross, T. Pflueger, R. Hilgendorf, P. McCormick, G. Salem, J. Connor, S. Geissler, and D. Thygesen. A 64b cpu pair: Dual- and single-processor chips. pages 333–334, February 2006.

[29] J. Friedrich, B. McCredie, N. James, et al. Design of the POWER6^{TM} microprocessor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007.

[30] Y. Yoshida, T. Kamei, K. Hayase, et al. A 4320MPIS four-processor core SMP/AMP with individually managed clock frequency for low power consumption. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007.

[31] U.G. Nawathe, N. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, and H. Park. An 8-core 64-thread 64b power-efficient SPARC SoC. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 108–109, February 2007.

[32] Z. Chen, P. Ananthanarayanan, S. Biswas, et al. A 25W SoC with dual 2GHz Power^{TM} cores and integrated memory and I/O subsystems. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007.

[33] J. Dorsey, S. Searles, M. Ciraula, et al. An integrated quad-core Opteron^{TM} processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007.

[34] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs. Implementation of the 65nm dual-core 64b Merom processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007.

[35] 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)*, pages 170–171, February 2003.

[36] 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)*, pages 428–429, February 2006.

[37] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Lyer, A. Singh, T. Jacb, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28TFLOPS

network-on-chip in 65nm CMOS. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 98–99, February 2007.

[38] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, D. Gurman, C. Chen, J. Cheung, D. Truong, and T. Mohsenin. Hardware and application of AsAP: An asynchronous array of simple processor. In *Hotchips*, August 2006.

[39] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, and J. Cheung. AsAP: A fine-grain multi-core plarform for DSP applications. *IEEE Micro*, pages 34–45, March/April 2007.

[40] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, Stanford, CA, October 1984.

[41] Ryan W. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, and B. Baas. A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1125–1134, October 2007.

[42] M. J. Meeuwsen, Z. Yu, and B. M. Baas. A shared memory module for asynchronous arrays of processors. *Eurasip Journal on Embedded Systems*, 2007:Article ID 86273, 13 pages, 2007.

[43] M. J. Meeuwsen, O. Sattari, and B. M. Baas. A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter. In *IEEE Workshop on Signal Processing Systems (SiPS '04)*, pages 124–129, October 2004.

[44] M. Laii. Atithmetic units for a high performance digital signal processor. Master's thesis, University of California, Davis, CA, USA, September 2004.

[45] Z. Yu and Bevan Baas. Implementing tile-based chip multiprocessors with GALS clocking styles. In *IEEE International Conference on Computer Design (ICCD)*, pages 174–179, October 2006.

[46] Z. Yu and B. Baas. Performance and power analysis of glabally asynchronous locally synchronous multi-processor systems. In *IEEE Computer Society Annual Symposium on VLSI*, pages 378–384, March 2006.

[47] J. Oliver, R. Rao, M. Brown, J. Mankin, D. Franklin, F. Chong, and V. Akella. Tile size selection for low-power tile-based architecture. In *ACM Computing Frontiers*, pages 83–94, May 2006.

[48] 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 *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 29–40, February 2002.

[49] S. Y. Kung. VLSI array processors. In *IEEE ASSP Magazine*, pages 4–22, July 1985.

[50] 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)*, pages 168–169, February 2003.

[51] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *IEEE International Conference on Design Automation (DAC)*, pages 684–689, June 2001.

[52] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das. Evaluating the Imagine stream architecture. In *International Symposium on Computer Architecture (ISCA)*, pages 19–23, June 2004.

[53] Texas Instruments. DSP platforms benchmarks. Technical report. `http://www.ti.com/`.

[54] Berkeley Design Technology. *Evaluating DSP Processor Performance*. Berkeley, CA, USA, 2000.

[55] The Embedded Microprocessor Benchmark Consortium. *Data sheets*. www.eembc.org, 2006.

[56] N. Bindal, T. Kelly, N. Velastegui, and K. Wong. Scalable sub-10ps skew global clock distribution for a 90 nm multi-GHz IA microprocessor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 346–347, 498, February 2003.

[57] T. Olsson and P. Nilsson. A digitally controlled PLL for SOC applications. *IEEE Journal of Solid-State Circuits (JSSC)*, pages 751–760, May 2004.

[58] E. Beigne and P. Vivet. Design of on-chip and off-chip interfaces for a GALS Noc architecture. In *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 13–15, March 2006.

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

[60] K. K. Parhi. *VLSI Digital Signal Processing Systems*. Wiley & Sons, 1999.

[61] G.K.Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, pages xviii–xxxiv, February 1992.

[62] J. Glossner, J. Moreno, M. Moudgill, J. Derby, E. Hokenek, D. Meltzer, U. Shvadron, and M. Ware. Trends in compilable dsp architecture. In *IEEE Workshop on Signal Processing Systems*, pages 181–199, October 2000.

[63] Eric W. Work. Algorithms and software tools for mapping arbitrarily connected tasks onto an asychronous array of simple processors. Master's thesis, University of California, Davis, CA, USA, in prepreration 2007.

[64] E.A. Lee. Programmable DSP architectures: Part i. *IEEE ASSP Magazine*, pages 4–19, October 1988.

[65] I. Kuroda and T. Nishitani. Multimedia processors. *Proceedings of IEEE*, pages 1203–1221, June 1998.

[66] A. Peleg and U. Weiser. MMX technology extension to the intel architecture. *IEEE Micro*, pages 42–50, August 1996.

[67] Analog Devices. Embedded processing and DSP. Technical report. `http://www.AnalogDevices.com/`.

[68] Freescale. Digital signal processors and controllers. Technical report. `http://www.freescale.com/`.

[69] EE times. Configurable processor focuses on video sureillance systems. Technical report. Aug. 23, 2007.

[70] S. Agarwala, A. Rajagopal, A. Hill, T. Anderson, M. Joshi, S. Mullinnix, T. Anderson, R. Damodaran, L. Nardini, P. Wiley, P. Groves, J. Apostol, M. Gill, J. Flores, A. Chachad, A. Hales, K. Chirca, K. Panda, R. Venkatasubramanian, P. Eyres, R. Velamuri, A. Rajaram, M. Krishnan, J. Nelson, J. Frade, M. Rahman, N. Mahmood, U. Narasimha, J. Frade, M. Rahman, N. Mahmood, U. Narasimha, S. Sinha, S. Krishnan, W. Webster, D. Bui, S. Moharil, N. Common, R. Nair, R. Ramanujam, J. Frade, and M. Ryan. A 65nm C64x+ multi-core DSP platform for communication infrastructure. *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263, February 2007.

[71] W.A. Wulf and C.G. Bell. C.mmp – a multi-mini-processor. In *AFIPS Conference*, pages 765–777, 1972.

[72] D. Lenoshi, J. Laudon, K. Gharachorloo, W.D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M.S Lam. The stanford DASH multiprocessor. *IEEE Computer*, pages 63–79, March 1992.

[73] C.L. Seitz. The cosmic cube. *Communications of the ACM*, pages 22–33, January 1985.

[74] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, , and J. Hennessy. The stanford FLASH multiprocessor. In *International Symposium on Computer Architecture (ISCA)*, pages 302–313, April 1994.

[75] D.H. Lawrie. Access and alignment of data in an array processor. *IEEE Transaction of Computers*, pages 1145–1155, 1975.

[76] H.S. Stone. Parallel processing with the perfect shuffle. *IEEE Transaction of Computers*, pages 153–161, 1971.

[77] C. Whitby-Strevens. Transputers-past, present and future. *IEEE Micro*, pages 16–19, December 1990.

[78] H. T. Kung. Why systolic architectures? In *Computer Magazine*, January 1982.

[79] H. T. Kung. Systolic communication. In *International Conference on Systolic Arrays*, pages 695–703, May 1988.

[80] L. Snyder. Introduction to the configurable, highly parallel computer. *IEEE Computer*, pages 47–56, January 1982.

[81] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. Bhaskar Rao. Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers*, pages 1054–1066, November 1982.

[82] U. Schmidt and S. Mehrgardt. Wavefront array processor for video applications. In *IEEE International Conference on Computer Design (ICCD)*, pages 307–310, September 1990.

[83] A. Keung and J.M. Rabaey. A 2.4 GOPS data-driven reconfigurable multiprocessor IC for DSP. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 108–110, February 1995.

[84] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, pages 86–93, September 1997.

[85] M.B. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwi. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *International Symposium on Computer Architecture (ISCA)*, June 2004.

[86] S. Rixner, W.J. Dally, U.J. Kapasi, B. Khailany, A. Lopez-Laguns, P. Mattson, and J.D. Owens. A bandwidth-efficient architecture for media processing. In *IEEE international Symposium on Microarchitecture (MICRO)*, pages 3–13, November 1998.

[87] B. Khailany, W. J. Dally, A. Chang, U. J. Kapasi, J. Namkoong, and B. Towles. VLSI design and verification of the imagine processor. In *IEEE International Conference on Computer Design (ICCD)*, pages 289–294, September 2002.

[88] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W.J. Dally. A programmable 512 GOPS stream processor for signal, image, and video processing. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 272 – 273, February 2007.

[89] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, and K. Olukotun. The stanford Hydra CMP. *IEEE Micro*, pages 71–84, March 2000.

[90] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey. A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing. *IEEE Journal of Solid-State Circuits (JSSC)*, 35(11):1697–1704, November 2000.

[91] 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)*, pages 161–171, June 2000.

[92] K. Mai, R. Ho, E. Alon, D. Liu, Y. Kim, D. Patil, and M. Horowitz. Architecture and circuit techniques for a reconfigurable memory block. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 500–501, 2004.

[93] K. Sankaralingam, R. Nagarajan, H. Liu, J. Huh, C.K. Kim, D. Burger, S.W. Keckler, and C.R. Moore. Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture. In *International Symposium on Computer Architecture (ISCA)*, pages 422–433, February 2003.

[94] M. Saravana, S. Govindan, D. Burger, S. Keckler, et al. TRIPS: A distributed explicit data graph execution (EDGE) microprocessor. In *Hotchips*, August 2007.

[95] H. Schmit, D. Whelihan, M. Moe, B. Levine, and R. R. Taylor. PipeRench: A virtualized programmable datapath in 0.18 micron technology. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 63–66, May 2002.

[96] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin. Wavescalar. In *IEEE international Symposium on Microarchitecture (MICRO)*, December 2003.

[97] S. Swanson, A. Putnam, M. Mercaldi, K. Michelson, A. Petersen, A. Schwerin, M. Oskin, and S.J. Eggers. Area-performance trade-offs in tiled dataflow architectures. In *International Symposium on Computer Architecture (ISCA)*, pages 314–326, May 2006.

[98] J. Oliver, R. Rao, P. Sultana, J. Crandall, E. Czernikowski, L. W. Jones, D. Franklin, V. Akella, and F. T. Chong. Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. In *International Symposium on Computer Architecture (ISCA)*, June 2004.

[99] D.C. Cronquist, P. Franklin, C. Fisher, M. Figueroa, and C. Ebeling. Architecture design of reconfigurable pipelined datapaths. In *Conference on Advanced Research in VLSI*, pages 23–40, March 1999.

[100] R. Baines and D. Pulley. A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers. *IEEE Communication Magazine*, pages 105–113, January 2003.

[101] S. Kyo, T. Koga, S. Okazaki, R. Uchida, S. Yoshimoto, and I. Kuroda. A 51.2GOPS scalable video recognition processor for intelligent cruise contol based on a linear array of 128 4-way VLIW processing elements. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 48–49, February 2003.

[102] J. Carlstrom, G. Nordmark, J. Roos, T. Boden, L. Svensson, and P. Westlund. A 40Gb/s network processor with PISC dataflow architecture. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 60–61, February 2004.

[103] W. Eatherton. The push of network processing to the top of the pyramid. In *Symposium on Architectures for Networking and communications systems*, 2005.

[104] Intellasys. SEAforth-24B, embedded array processor. Technical report. `http://www.intellasys.net/`.

[105] Mathstar. Arrix family product brief. Technical report. `http://www.mathstar.com/`.

[106] Rapport. KC256 technical overview. Technical report. `http://www.rapportincorporated.com/`.

[107] A.M. Jones and M. Butts. TeraOPS hardware: A new massively-parallel MIMD computing fabric IC. In *Hotchips*, August 2006.

[108] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivit, P. Penard, A. Bouttier, and F. Berens. A telecom baseband circuit based on an asynchronous network-on-chip. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 258–259, February 2007.

[109] V. Yalala, D. Brasili, D. Carlson, A. Hughes, A. Jain, T. Kiszely, K. Kodandapani, A. Varadhrajan, and T. Xanthopoulos. A 16-core RISC microprocessor with network extensions. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 100–101, February 2006.

[110] H. Zhang, M. Wan, V. George, and J. Rabaey. Interconnect architecture exploration for low-energy reconfigurable single-chip dsps. In *IEEE Computer Society Workshop on VLSI*, pages 2–8, April 1999.

[111] P. P. Pande, C. Crecu, M. Jones, A. Ivanov, and R. Saleh. Effect of traffic localization on energy dissipation in noc-based interconnect. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1774–1777, May 2005.

[112] S. Vangal, A. Singh, J. Howard, S. Dighe, N. Borkar, and A. Alvandpour. A 5.1GHz 0.34mm^2 router for network-on-chip applications. In *Symposium on VLSI Circuits*, pages 42–43, June 2007.

[113] J. Hu and R. Marculescu. Application-specific buffer space allocation for network-on-chip route design. In *IEEE International Conference on Computer Aided Design (ICCAD)*, pages 354–361, 2004.

[114] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, pages 1347–1356, December 1987.

[115] W. J. Dally. Vitual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, pages 194–205, March 1992.

[116] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwi. Scalar operand networks: On-chip interconnect for ILP in partitioned architecture. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 341–353, February 2003.

[117] G. Campobello, M. Castano, C. Ciofi, and D. Mangano. GALS networks on chip: a new solution for asynchronous delay-insensitive links. In *Design, Automation and Test in Europe (DATE)*, pages 160–165, April 2006.

[118] K. Lee, S. Lee, and H. Yoo. Low-power network-on-chip for high-performance SoC design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 148–160, February 2006.

[119] M. Fukase, T. Sato, R. Egawa, and T. Nakamura. Scaling up of wave pipelines. In *International Conference on VLSI Design*, pages 439–445, January 2001.

[120] P. Cocchini. Concurrent Flip-Flop and repeater insertion for high performance integrated circuits. In *IEEE International Conference on Computer Aided Design (ICCAD)*, pages 268–273, November 2002.

[121] 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)*, pages 267–274, October 2005.

[122] A. Suga, T. Sukemura, H. Wada, H. Miyake, Y. Nakamura, Y. Takebe, K. Azegami, Y. Himura, H. Okano, T. Shiota, M. Saito, S. Wakayama, T. Ozawa, T. Satoh, A. Sakutai, T. Katayama, K. Abe, and K. Kuwano. A 4-way VLIW embedded multimedia processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 240–241, February 2000.

[123] R. Witek and J. Montanaro. StrongARM: A high-performance arm processor. In *IEEE COMPCON*, pages 188–191, February 1996.

[124] J. Sungtae, M. B. Taylor, J. Miller, and D. Wentzlaff. Energy characterization of a tiled architecture processor with on-chip network. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 424–427, 2003.

[125] C. Kozyrakis and D. Patterson. Vector vs. superscalar and vliw architectures for embedded multimedia benchmarks. In *IEEE international Symposium on Microarchitecture (MICRO)*, pages 283–293, November 2002.

[126] B. Gorjiara and D. Gajski. Custom processor design using NISC: a case-study on DCT algorithm. In *Workshop on embedded systems for real-time multimedia*, pages 55–60, September 2005.

[127] T. Lin and C. Jen. Cascade – configurable and scalable DSP environment. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 26–29, May 2002.

[128] M. Matsui, , H. Hara, Y. Uetani, L.-S. Kim, T. Nagamatsu, Y. Watanabe, A. Chiba, K. Matsuda, and T. Sakurai. A 200 MHz 13 mm^2 2-d DCT macrocell using sense-amplifying pipeline flip-flop scheme. *IEEE Journal of Solid-State Circuits (JSSC)*, pages 1482–1490, December 1994.

[129] K. Maharatna, E. Grass, and U. Jaqdhold. A 64-point fourier transform chip for high-speed wireless LAN application using OFDM. *IEEE Journal of Solid-State Circuits (JSSC)*, 39:484–493, March 2004.

[130] J. Thomson, B. Baas, E. M. Cooper, J. M. Gilbert, G. Hsieh, P. Husted, A. Lokanathan, J. S. Kuskin, D. McCracken, B. McFarland, T. H. Meng, D. Nakahira, S. Ng, M. Rattehalli, J. L. Smith, R. Subramanian, L. Thon, Y.-H. Wang, R. Yu, and X. Zhang. An Integrated 802.11a Baseband and MAC Processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 126–127, 451, February 2002.

[131] B. M. Baas. A parallel programmable energy-efficient architecture for computationally-intensive DSP systems. In *Signals, Systems and Computers, 2003. The Thirty-Seventh Asilomar Conference on*, pages 2185–2189, November 2003.

[132] S. F. Smith. Performance of a GALS single-chip multiprocessor. In *The International Conference on Parallel and Distributed Processing Techniques and Applicatoins (PDPTA)*, pages 449–454, June 2004.

[133] E. Talpes and D. Marculescu. Toward a multiple clock/voltage island design style for power-aware processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 591–603, May 2005.

[134] A. Upadhyay, S. R. Hasan, and M. Nekili. Optimal partitioning of globally asynchronous locally synchronous processor arrays. In *Great Lakes Symposium on VLSI (GLSVLSI)*, pages 26–28, April 2004.

[135] D. A. Patterson and J. L. Hennessy. *Computer Architecture – A Quantitative Approach*. Morgan Kaufmann, second edition, 1999.

[136] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. In *International Symposium on Computer Architecture (ISCA)*, pages 158–168, May 2002.

[137] E. Talpes and D. Marculescu. A critical analysis of application-adaptive multiple clock processor. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 278–281, August 2003.

[138] G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, S. G. Dropsho, and S. Dwarkadas. Hiding synchronization delays in a GALS processor microarchitecture. In *International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 159–169, April 2004.

[139] Omar Sattari. Fast fourier transforms on a distributed digital signal processor. Master's thesis, University of California, Davis, CA, USA, September 2004.

[140] C. E. Dike, N. A. Kurd, P. Patra, and J. Barkatullah. A design for digital, dynamic clock deskew. In *Symposium on VLSI Circuits*, pages 21–24, June 2003.

[141] P. Mahoney, E. Fetzer, B. Doyle, and S. Naffziger. Clock distribution on a dual-core multi-threaded Itanium-family processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 292–293, February 2005.

[142] D. C. Sekar. Clock trees: differential or single ended? In *International Symposium on Quality of Electronic Design*, pages 545–553, March 2005.

[143] M. Hashimoto, T. Yamamoto, and H. Onodera. Statistical analysis of clock skew variation in H-tree structure. In *International Symposium on Quality of Electronic Design*, pages 402–407, March 2005.

[144] T. Njolstad, O. Tjore, K. Svarstad, L. Lundheim, T. O. Vedal, J. Typpo, T. Ramstad, L. Wanhammar, E. J. Aar, and H. Danielsen. A socket interface for GALS using locally dynamic voltage scaling for rate-adaptive energy saving. In *Annual IEEE International ASIC/SOC conference*, pages 110–116, September 2001.

[145] K.J. Nowka, G. D. Carpenter, E.W. MacDonald, H.C. Ngo, B.C. Brock, K.I. Ishii, T.Y. Nguyen, and J.L. Burns. A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling. *IEEE Journal of Solid-State Circuits (JSSC)*, pages 1441–1447, November 2002.