Tao: An Architecturally Balanced Reconfigurable Hardware Processor

by

Andrew S. Huang

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 23, 1997

Author _____
Department of Electrical Engineering and Computer Science
May 23, 1997

Certified by_____
Gill A. Pratt
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Tao: An Architecturally Balanced

Reconfigurable Hardware Processor

by

Andrew S. Huang

Submitted to the

Department of Electrical Engineering and Computer Science

May 23, 1997

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Tao is a high performance platform for implementing reconfigurable hardware designs. The architecture features a high aggregate throughput and a modular processor core consisting of four reconfigurable macrofunction units embedded in a toroidal interconnect matrix. The modular core allows the platform to be upgraded as reconfigurable hardware technology progresses. A high aggregate throughput is achieved by striking a balance of bandwidth among all the datapath elements, from the PCI bus interface to the intelligent buffers called reformatting engines, to the processor core itself. The platform also features an embedded microcontroller for the support of dynamic reconfiguration schemes. Tao was implemented on a double height PCI card using Xilinx XC4013E-3 FPGAs as the base reconfigurable hardware technology.

Thesis Supervisor: Gill A. Pratt

Title: Assistant Professor, MIT Electrical Engineering and Computer Science

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ALE: Address Latch Enable (GPCB Signal)
BLR: Board Level Routing
BSFF: Bidirectional Stall Flip Flop
CE: Configuration Engine
DA: Distributed Arithmetic
DC: Direct Current
DSP: Digital Signal Processing
DTIP: Data Transfer In Progress (GPCB signal)
EEPROM: Electrically Erasable Programmable Read Only Memory
EFF: Enabled Flip Flop
FIFO: First In First Out
FPGA: Field Programmable Gate Array
GPCB: General Purpose Configuration Bus
GPP: General Purpose Processor
I/O: Input / Output
JPEG: Joint Picture Experts Group
LED: Light Emitting Diode
LUT: Look Up Table
PCI: Peripheral Connect Interface
PLL: Phase Locked Loop
QBOS: Quad Binary Operator Sum
RE: Reformatting Engine
RHP: Reconfigurable Hardware Processor
RISC: Reduced Instruction Set Computer
RMU: Reconfigurable Macrofunction Unit
SA: Serial Arithmetic
SLUT: SSRAM Look Up Table
SSRAM: Synchronous Static Random Access Memory
USFF: Unidirectional Stall Flip Flop

ACKNOWLEDGMENTS

The author would like to thank his parents for all their love and support over the years, and for making the sacrifices to put him through M.I.T. I love you, mom and dad!

The author would also like to extend a warm thanks to Rob Gilmore of QUALCOMM, Inc. for his enthusiastic support and sponsorship of this thesis work. Without his backing and the backing of Dr. Andrew J. Viterbi of QUALCOMM, Inc., this thesis would not have happened. The author is also indebted to Dr. Donald Pian, who served as his company thesis advisor at QUALCOMM, Inc., for all his invaluable advice on signal processing and algorithms. The author also thanks the engineers at QUALCOMM, Inc. who helped make Tao a reality, including but not limited to Sherman Hepworth, Chip Girardot, Kay Lemonovich, Quy Nguyen & the Q-Proto Team, Freddy Huston, Steve Rogers, Bill Stein, Kendrick Yuen, David E. Wang, Dexter Chun, and Steve Sirotzky. The author would also like to thank Dr. Chong U. Lee of Solana Technology Development Corporation for his advice and input on video signal processing architectures and for all the free lunches.

The author thanks his warm and supportive academic thesis advisor Prof. Gill A. Pratt for his moral support, encouragement, and advice.

Finally, the author would like to thank all his brothers and friends for their moral support, friendship, and patience which sustained him through the trials and tribulations of M.I.T., and helped make the past five years the best and most fun five years of his life.

1. Introduction

The conventional paradigm of computation, embodied by the ideas of Church and Turing and embedded in the stored-program (von Neumann) architecture, formulates the dichotomy of hardware and software. [Cha96] Hardware is the *structure* of the computer; it is immutable and tangible, like the Turing tape reader. Software is the *purpose* of the computer; it is volatile and intangible, like the bits of data on the infinite Turing tape. Excluded from this paradigm is the possibility of modifying the structure to suit the purpose. As a result, contemporary microprocessor architectures exhibit optimizations such as multiple bus standards for different purposes, branch prediction, register renaming, and speculative execution. However, if hardware could be modified (reconfigured) to suit the purpose, one can imagine a computer with a single all-in-one expansion socket, and processors would not need such complex optimizations. Indeed, thanks to programmable logic technology, this traditional paradigm can be replaced with something new. With programmable logic, structure can be modified for a specific purpose, resulting in increased performance. A new genre of computer must be defined: a computer without a stored-program architecture that can be customized to any algorithm, including a universal Turing Machine. For this computer, the architecture becomes the program. Others who have studied such machines have referred to them as either virtual computers [Cas93a], custom computers, programmable active memories [Vui96], functional memories [Hal94], or transformable computers. This work shall refer to them by the common term, "reconfigurable computer".

How is it that the same piece of hardware can be reconfigured to perform multiple tasks? The enabling technology of programmable logic is based on the premise that the functionality of hardware lies in the way primitive elements are connected. Thus, given a sufficiently large set of primitive sequential and combinational elements and a

reconfigurable interconnect, a vast number of functions can be implemented with the same hardware.[Vui94] [VuiA]

Researchers have been quick to implement reconfigurable computers, lured by the promise of fast computation. However, describing an algorithm in hardware is an arduous task, and performance is often times disappointing because the underlying structure of the programmable logic contains inherent weaknesses. For reasons involving cost, performance, and engineering difficulty, a high-end workstation is often preferable to a reconfigurable computer. Although programmable logic is capable of generalized computation, it is not capable of efficiently implementing all useful functions due to implementation-specific architectural weaknesses. These limitations can be overcome, in part, if proper hardware support is provided. In addition, it is easier to create applications for reconfigurable computer designs if certain architectural tradeoffs are applied. This thesis presents an architecture for implementing practical reconfigurable hardware designs.

1.1 The Postulation of Reconfigurable Computing

The purpose of reconfigurable computing is to provide a faster and perhaps cheaper solution to a range of key computational problems when compared to general purpose processors (GPPs) such as the DEC Alpha or TI's C80 MVP. This section will investigate the worth of reconfigurable computing from the standpoints of future growth, economics, and theoretical advantages.

1.1.1 Growth of GPP machines

GPP machines are ubiquitous because of their flexibility, relative ease of programming, and satisfactory performance. Mainstream GPPs qualify as universal computers with a stored-program (von Neumann) architecture. [Pat96] Because of their popularity, a great amount of effort and capital has been invested in developing GPPs; as a result, the performance of GPPs has been growing exponentially. Gordon Moore noticed this trend in 1965 and this phenomenon has since been dubbed "Moore's Law". [Moo96]

An important consideration when designing a reconfigurable computer is if the design can remain competitive with the aggressive performance growth curve of GPPs. A

general observation about previous work on reconfigurable computers is that none of the approaches utilized dedicated arithmetic hardware to accelerate difficult computations such as multiplies. One would presume that a dedicated multiplier connected to an FPGA would speed up operations since multiplies are inefficient in fine-grained architectures. One architecture, Neil Bergmann's CC-DSP [Ber94], seemed to agree with this philosophy; however, upon further inquiry regarding his research, Dr. Bergmann informed me that

> Arithmetic chips are not on a technology speed up curve, but most use 10 year old technology. We anticipate FPGA's will exceed their performance and continue to do so even more in the future. [Ber96]

Dr. Bergmann has since researched machines based solely on FPGAs. Although discrete arithmetic chips are not on a technology speed up curve, integrated arithmetic functions in the form of embedded cores are by definition on the same technology curve as FPGAs. It turns out that there could be a substantial advantage to having embedded cores on the same silicon die as an FPGA.

Some peak performance figures of GPPs are listed in Table 1-1 for reference.

Processor	MIPS/chip
DEC Alpha 21064, 200 MHz	400
DEC Alpha 21164, 600 MHz	2400
TMS-PPDS, 40 MHz	40
TMS320C80, 50 MHz	2000

Table 1-1: Performance figures of GPPs. MMAC/s stands for Millions of Multiply Accumulates / second. TMS-PPDS consists of 4 TMS320C40s.

1.1.2 Growth of FPGAs

GPP technology is not alone in its aggressive growth curve. High demand for programmable logic helps drive FPGA technology. Beyond economics, memory-based FPGA technology has the following factors in its favor:

- Memory technology, especially SRAM technology, is growing in density, speed and cost-effectiveness at a rate of $1/\alpha = 1.25$ each year (Noyce's thesis). Since SRAM-based FPGAs use this technology, they also share its growth curve. [Vui]
- New and refined techniques, such as distributed arithmetic and improved Electronic Design Automation (EDA) tools, will help designers squeeze every last drop of performance out of reconfigurable hardware. Steve Casselman suggests that reconfigurable hardware technology exhibits hyperscaleability—a performance to logic area scaling factor better than 1:1. In other words, for a given problem, twice the logic area will yield better than twice the performance. [Cas93]
- New architectural features, such as fast carry chains and embedded function blocks, will provide application specific performance growth.

Because reconfigurable hardware performance growth stems from significant enhancements in both the silicon and the architecture, one could speculate that the overall short-term reconfigurable hardware performance curve is steeper than that of the GPP. In fact, recent performance statistics released by Altera and Xilinx suggest that their FPGA technology outclasses GPP performance by over an order of magnitude.

Processor	Relative Performance
133 MHz CISC CPU	0.24
50 MHz DSP Processor	1
Four 50 MHz DSP Processors	4
Altera EPF8820A-2 (75% utilization)	32
Altera EPF81500A-2 (60% utilization)	67.2

Table 1-2: GPP versus FPGA performance comparison in selected DSP application. [Alt23]

Thus, reconfigurable computer designs which rely on prime-growth reconfigurable hardware technology and other actively researched technologies such as fast SRAM have a reasonable chance at keeping apace with GPP performance.

1.1.3 Programmability of Reconfigurable Hardware

Perhaps the greatest stumbling block for the mass acceptance of reconfigurable hardware is its high engineering cost per application. For example, efficient FPGA implementations require an intimate knowledge of the underlying hardware and the development tools. Unfortunately, neither hardware nor development tools are standardized, making most designs grossly unportable and difficult to upgrade. Also, debugging a hardware design is extremely difficult. It is a slow and arduous process, especially if multiple FPGAs are involved. In the end, most programmers and algorithm researchers would prefer to avoid learning EDA tools and mucking about with hardware—they would rather code in familiar languages like C/C++ even if it means that their applications run orders of magnitude slower. [Cas94]

Although the efficient translation of high-level concepts to hardware is beyond the scope of this thesis, it is important to acknowledge the futility of reconfigurable hardware without useful application configurations. When creating a reconfigurable hardware architecture, it is important for the designer to keep in mind the end users and to make tradeoffs which increase the usability and testability of the system.

1.1.4 Significance of RHP Technology

1.1.4.1 Essential Concept

Perhaps the role of the RHP can be better understood if one considers its place in an analogy using the Turing machine. To add strength to this analogy, real numbers from real processors will be presented in the next section. Recall that a Turing machine $\mathbf{T_i[X]}$ consists of some finite state machine $\mathbf{FSM_i}$ and some tape configuration **X**, and computes some result **Y**. In this example, it may be helpful to think of the tape as a sufficiently large bank of generalized memory. In the case of both GPPs and ASICs, $\mathbf{FSM_i}$ is fixed and the only variant between applications is the configuration of **X**.

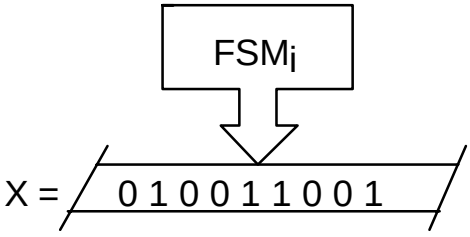


Figure 1-1: Turing Machine

GPPs achieve generality by choosing an $\mathbf{FSM_{GPP}}$ that is capable of emulating any other Turing machine—in other words, the GPP is a universal Turing machine. Church's thesis states that every discrete function computable by any realizable machine is computable by some Turing machine, so the ability to emulate any practical Turing machine is equivalent to the ability to compute any discrete function. [Pra97]

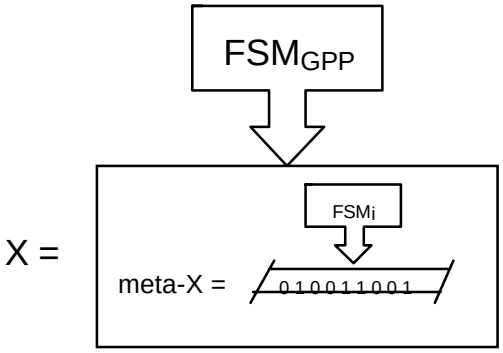


Figure 1-2: Turing machine analogy of the GPP.

This process of interpretation, or using coded representations of machines as input data to another machine, is extremely powerful, but it has the disadvantage that it is slower than direct implementations. Each emulated operation of $\mathbf{FSM_i}$ would require several operations of $\mathbf{FSM_{GPP}}$ unless the architecture of the targeted $\mathbf{FSM_i}$ happens to closely match the given $\mathbf{FSM_{GPP}}$. Also, because the $\mathbf{FSM_{GPP}}$ has to be able to emulate any number of typical $\mathbf{FSM_i}$, not every $\mathbf{FSM_i}$ application will take advantage of all the hardware embodied in $\mathbf{FSM_{GPP}}$.

ASICs are examples of direct implementations of computable functions. The Turing machine analogy of an ASIC is some $\mathbf{T_{ASIC}}$ which uses an $\mathbf{FSM_{ASIC}}$ that is optimized for a particular application. This typically implies a loss of generality, often to

the point where $\mathbf{T_{ASIC}}$ is no longer a universal Turing machine; however, this loss of generality is traded off for better performance and/or less hardware, depending on the target application.

The Turing analogy of an RHP is some $\mathbf{T_R}$ where **R** is the set of realizable FSMs using the available reconfigurable hardware resources. The key difference between $\mathbf{T_R}$ and $\mathbf{T_{GPP}}$ and $\mathbf{T_{ASIC}}$ is that $\mathbf{FSM_{GPP}}$ and $\mathbf{FSM_{ASIC}}$ are fixed while $\mathbf{FSM_R}$ is flexible within certain bounds. This introduces a new set of performance tradeoffs that spans the gap between the GPP and the ASIC. In terms of practical advantages, $\mathbf{T_R}$ offers a performance advantage over $\mathbf{T_{GPP}}$ because $\mathbf{T_R}$ can be configured to more closely match the target $\mathbf{T_i}$. This eliminates some of the interpretive overhead, without a loss of generality. $\mathbf{T_R}$ also has practical advantages over $\mathbf{T_{ASIC}}$ in many situations. By either exclusively using RHP technology or a hybrid between ASIC and RHP technologies, one can expand the scope of applications that a particular piece of hardware can perform with little loss in performance. This translates to savings in reengineering costs and a greater market penetration to more customers.

1.1.4.2 Evidence of Essential Concept

Although the Turing Machine analogy is useful for trying to introduce some fundamental concepts about the distinctions between RHPs, GPPs, and ASICs, it has little bearing on what can be accomplished in practice. The Turing Machine is a thought experiment in computation—no engineer would implement such a machine because its purpose is to prove computability with no consideration of performance. In order to add substance to the Turing Machine analogy, a comparison of contemporary GPPs, RHPs, and ASICs is provided in this section.

A comparison of GPPs, RHPs, and ASICs can be made on the basis of performance versus cost of generality. The cost of generality is defined as the amount of effort, measured in dollars, required to create a particular application in a given technology. Effort is primarily human labor costs and in the case of ASICs, the cost of a fabrication run. Since startup costs could be large compared to the cost of developing a single application, the startup cost is amortized over twenty applications in this analysis.

Measuring performance can also be very tricky, since neither RHPs nor ASICs break down operations into quantizable "instructions" as GPPs do. A more general metric, results per second, is used instead. In this case, a simple signal processing (16-tap FIR) application is used to benchmark performance in terms of samples per second.

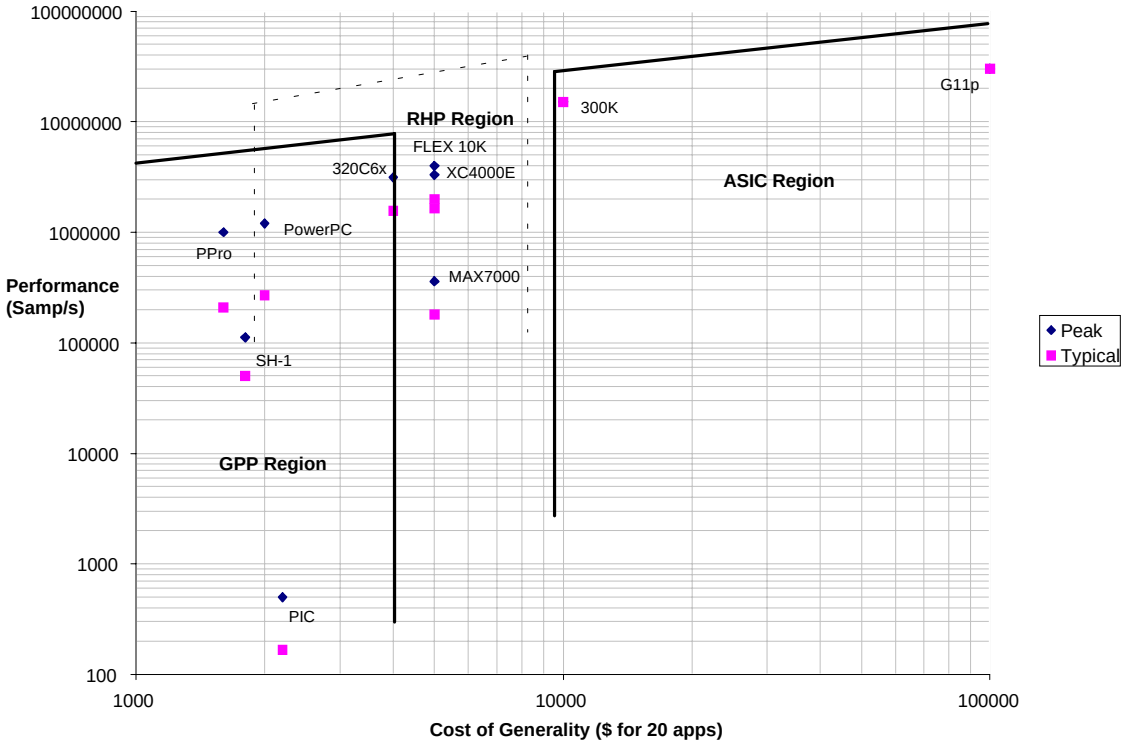


Figure 1-3: Cost of Generality versus Performance

Figure 1-3 summarizes the relationship between GPPs, RHPs, and ASICs in terms of performance versus cost of generality per twenty applications. One can see that the space is divided into three overlapping regions, one for each technology. These regions denote the set of performance-cost points one may encounter using each technology. The GPP region covers the lower performance, low cost space, while the ASIC region covers the highest performance, high cost space. However, there is quite a gap between the GPP and ASIC regions in both cost and performance. The RHP spans this gap as a compromise between performance and cost. As GPPs continue to be driven toward higher performance and ASICs toward lower cost, the RHP rises as the solution of choice in

terms of cost of generality per unit performance. Thus, this graph adds some credibility to the Turing Machine analogy presented in the previous section.

Some data points have been assigned to the regions demarcated in Figure 1-3 to add credibility to these claims. The points in the ASIC region are the G11p and the 300K processes. The G11p is LSI Logic's high end 0.18 μm process which provides up to 6.1 million gates per die at an f_t of 1.2 GHz and a system clock of up to 300 MHz. The 300K is LSI Logic's low end 0.6 μm process which provides up to 500,000 gates at an f_t of 350 MHz and a system clock of up to 150 MHz. [LSI97]

The points in the GPP region are based on Intel's Pentium Pro processor, the Power PC603e, TI's TMS320C6x VLIW DSP processor, Hitachi's SH-1 RISC processor, and the Microchip PIC 16cXX microcontroller. The points were chosen to run the gamut of performance versus cost. At the high end, TI's TMS320C6x processor performs well because it is optimized for DSP applications. At the low end, the Microchip PIC 16cXX performs very poorly and at a high development cost because the 16-tap FIR application is extremely difficult to implement in such a small processor. The performance numbers for all processors are estimates derived by coding up the 16-tap FIR application in C, compiling to assembly, and examining the number of instructions in the main loop. This number is scaled by each manufacturer's advertised performance rates. In one case, the TMS320C6x, an optimizing compiler was not available, and thus the performance information was extrapolated from manufacturer data. The author refers interested readers to the various manufacturer's web pages for the most up-to-date performance figures.

The points in the RHP region are based on Xilinx's 4000EX architecture, Altera's 10K architecture, and Altera's MAX7000 architecture. The performance numbers were pulled from [New94], [Alt23], and in the case of the MAX7000, the author's best estimate of this low-end RHP's performance.

1.2 Reconfigurable Hardware Primer

1.2.1 Commercially Available Reconfigurable Hardware Architectures

Today's market offers a wide variety of reconfigurable hardware architectures targeted at a diverse range of applications. At the low end, there are in-circuit programmable logic arrays such as the Altera MAX7000S series EPLD. These devices are designed for "system glue" functions; they have a high system performance at a relatively low density and cost. At the high end, SRAM-based Field Programmable Gate Arrays (FPGAs) such as the Xilinx XC4000E series dominate the market. These FPGAs are targeted at implementing entire computational or control subsystems in a single chip. Recently, Actel has introduced a family of FPGAs which can include standard cores, such as DSPs and PCI interfaces, right on the die. [Wil97] This higher level of integration boosts the maximum performance level by allowing greater bandwidth and accessibility between on-chip components while reducing system costs.

Regardless of the target application or performance point, all commercially available reconfigurable hardware fall into the class of architecture known as regular structures. Regular structures employ one or two core elements referred to as logic blocks or macrocells that consist of some combinational and some sequential logic. These logic blocks are then typically repeated in an array form, and surrounded by inter-block routing resources. Reconfigurability of a regular structure implies that the logic function computed by the combinational resources is programmable, and that the routing resources can be programmed to support different interconnect patterns. The primary subclasses of regular structures are the *coarse-grained* and the *fine-grained* architectures.

1.2.1.1 Coarse-Grained Architectures

Coarse-grained architectures feature relatively small arrays of wide logic blocks embedded in some hierarchical routing structure. "Wide logic" refers to logic of many (on the order of 20) input variables. Each piece of logic can feed one flip flop; hence, coarse-grained architectures favor combinational logic-intensive applications. Programmable Logic

Devices (PLDs) such as Altera's MAX 5000/7000/9000 series or Cypress' UltraLogic FPGAs, and Xilinx's 9500 CPLDs are examples of a coarse-grained philosophy.

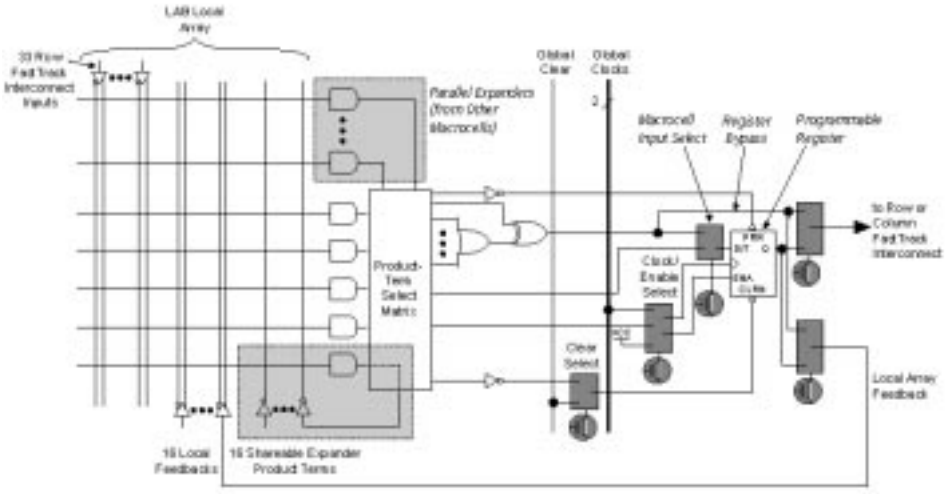


Figure 1-4: Structure of Altera's MAX9000 Macrocell and Local Array [Alt126]

The Altera MAX 5/7/9000 series employ a hierarchical structure, with the macrocell being the smallest unit (see Figure 1-4 and Figure 1-5 for block diagrams). Macrocells consist of a PAL-like (programmable AND, fixed OR) product term array (with 32+ product terms) feeding a single flip flop; macrocells are organized into logic array blocks (LABs) via a local interconnect, and LABs are globally interconnected via a Programmable Interconnect Array (PIA). [Alt95] Cypress' borderline coarse-grained UltraLogic FPGA uses an element similar to the macrocell but without the hierarchical structure; instead, the FPGA is organized as a uniform array of macrocells with global routing resources distributed throughout. [Cyp96]

As mentioned before, coarse-grained architectures are designed for system glue, control, and timing generation applications. The wide logic implemented in the logic cells allow wide address decoders to be implemented in a single cell, which is key in system-level applications. It also allows complex, high performance FSMs to be implemented with one cell per state bit—an important feature for control and timing generation applications. Most coarse-grained architectures also have a routing scheme which features predictable delays between logic cells. This allows designers to perform pre-compilation timing analyses when doing speed-critical designs. The key disadvantage of coarse-

grained architectures is that the performance roll-off with increasing complexity is very steep. For example, arithmetic operations tend to require more product terms than a single logic cell is capable of supporting; thus, logic cells must be cascaded and the computation time for the function is the depth of the cascade times the delay of a single cell plus the routing delay. Also, each bit of a multibit bus function will require at least one logic cell, since each logic cell yields a single bit of output. Thus, trivial arithmetic operations such as **y**[31:0] = (**a**[31:0] $\wedge$ **b**[31:0]) will be area-inefficient because most of the product terms in each logic cell will be unused. Finally, because of the high ratio of combinational logic to registers, pipelining computations is not practical; each pipeline stage will tie up not only a single register, but also the entire block of combinational logic which feeds the register.

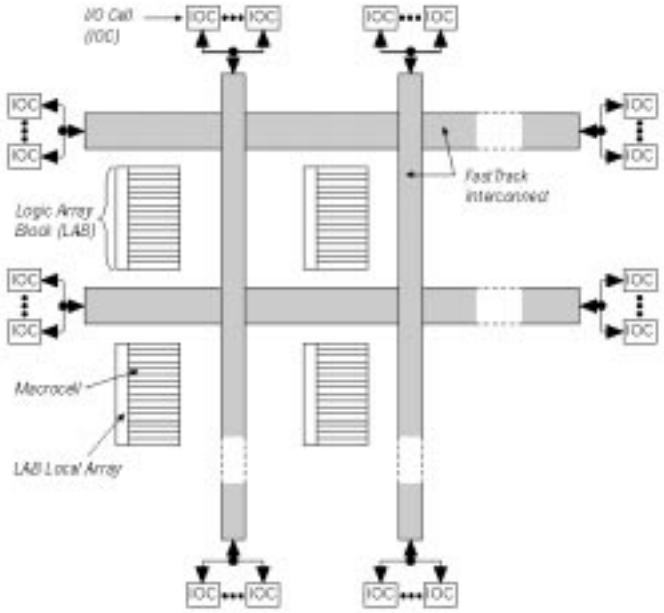


Figure 1-5: Altera's MAX9000 device block diagram [Alt123]

1.2.1.2 Fine-Grained Architectures

The answer to implementing complex computational functions in a relatively efficient manner is the fine-grained architecture. In this case, each device consists of a large array of narrow (roughly 5 input) logic cells in an orthogonal routing matrix. Because the size of the combinational logic in each logic cell is so much smaller than in the coarse-grained

case, fine-grained architectures can cram more logic cells into the same area. As a result, the ratio of combinational logic to registers and routing resources tends to be better balanced. While fine-grained logic suffers greater combinational delay penalties when implementing very wide logic functions, this loss is outweighed by the gain in area efficiency and the ability to create efficient pipelines. Area efficiency is important because spanning a single computation between multiple chips is too slow to be a viable option. Also, many fine-grained architectures offer special carry chains that can be used to speed up operations such as additions, with the tradeoff of strict logic placement constraints. This in turn limits the routability of the design.

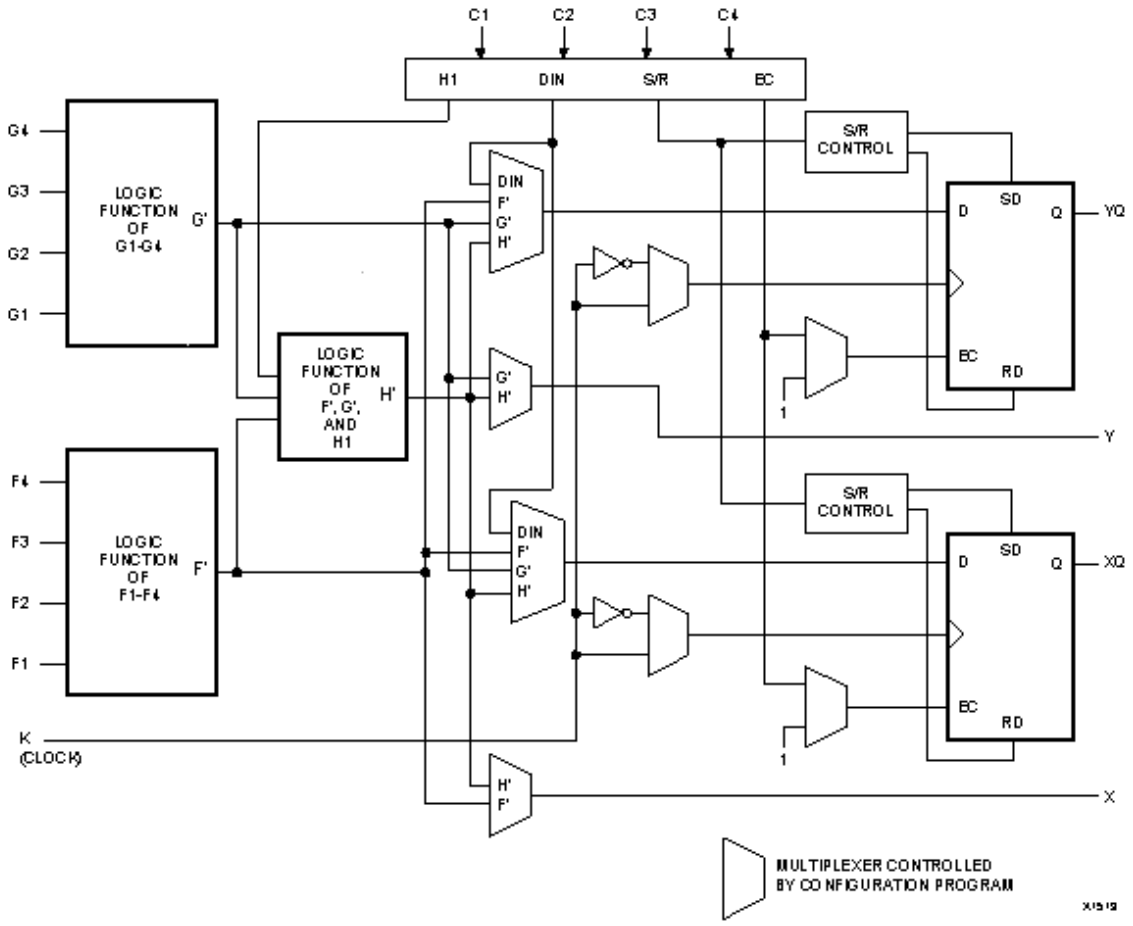


Figure 1-6: Simplified structure of Xilinx 4000 series CLB. [Xil2-10]

Some examples of fine-grained reconfigurable hardware architectures are the Xilinx 4000E series and the Lucent Orca series FPGAs. The structure of a Xilinx 4000E series

logic cell (called a Configurable Logic Block (CLB) in Xilinx jargon) is shown in Figure 1-6. There are around a thousand of these CLBs in each of their mid-range devices. Not shown in Figure 1-6 are the carry chain logic and I/O. The function generators and configuration muxes, as well as the routing resources, all have their configuration state stored in SRAM cells (as opposed to EPROM cells); this allows the devices to be reconfigured in-circuit without a special erase cycle. Figure 1-7 illustrates the orthogonal routing structure of the XC4000E FPGA.

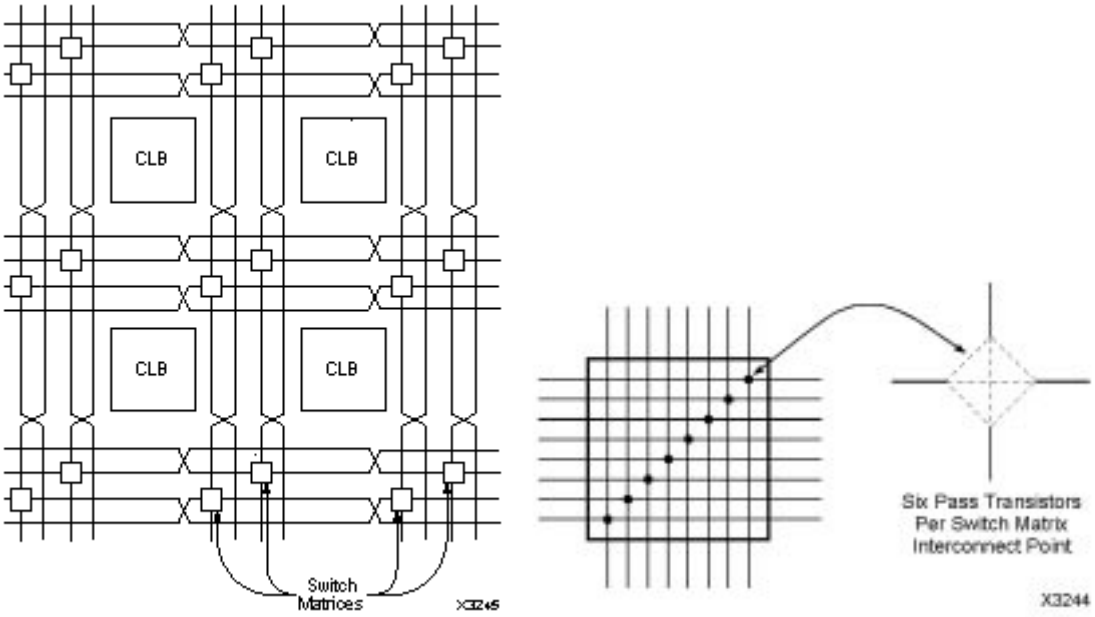


Figure 1-7: Double-length routing resources in the Xilinx 4000 Series [Xil2-15]

Because the function generators are n-bit lookup tables (LUTs), they can compute any boolean function ***h*** of n bits. Thus, in this case, each flip flop could have as an input any boolean function of the form

$$\mathbf{d} = h(f(F_1, F_2, F_3, F_4), g(G_1, G_2, G_3, G_4), H_1) \qquad (1\text{-}1)$$

Note that the ***f, g,*** and ***h*** functions are referred to as FMAP, GMAP, and HMAP generators in Xilinx jargon.

Because the function generator LUTs are SRAM based, they can also serve as distributed memory elements. In other words, each CLB can be configured as either a

16x1, 16x2, or 32 x1 single or dual-ported, edge-triggered or level-sensitive RAM. Table 1-3 summarizes the availability of the different modes.

	16 x 1	16 x 2	32 x 1	Edge-timing	Level-timing
Single-Port	X	X	X	X	X
Dual-Port	X			X	

Table 1-3: Supported RAM Modes [Xil4-14]

This feature gives this particular fine-grained architecture a distinct advantage in distributed arithmetic computations.

The Lucent Technologies Orca architecture also has special features directed at the implementation of arithmetic functions. Their ORCA 2C FPGA is based on a Programmable Function Unit (PFU) which can be configured as a 4x1 multiplier. Although the difference is subtle, the Xilinx 4000 series CLB is unable to implement a 4x1 multiplier in a single CLB due to the manner in which the LUTs are partitioned. The ORCA 2C series of FPGA also uses a hierarchical partitioning scheme for wire routing that divides the chip into four large quads, and each quad into several subquads. [ATT95]

1.2.1.3 Other Architectural Distinctions

The are some other architectural distinctions, in addition to granularity, that are important to this discourse. The first distinction is partial dynamic reconfigurability. Certain architectures, such as Atmel's AT6000 series FPGA and MIT's DPGA architecture, support this. [DeH95] Atmel's jargon for partial dynamic reconfigurability is "Cache Logic", which hints at the concept of storing logic configurations in an external RAM cache and loading them in when needed. This characteristic is desirable when implementing an RHP because it gives the user a broad range of flexibility in high-level design, as well as the ability to use portions of the hardware while concurrently reconfiguring other portions of the hardware. This helps mitigate the objectionably long configuration time (tens of milliseconds) typical of most reconfigurable hardware. [Ros9-11]

1.2.2 Research Reconfigurable Hardware Architectures

A number of innovative reconfigurable hardware architectures have been explored by research teams in universities and in industry. Perhaps one of the greatest drawbacks of all the commercially available reconfigurable hardware products is the difficulty associated with creating high clock-rate applications for them. Conventional architectures focus on providing users with an extremely large array of one or two kinds of primitive logic cells. Although this makes the architecture very general, certain applications have inefficient mappings to these architectures. System integration is also more difficult because there is no built-in structured I/O such as a PCI or localbus interface. A predominant theme in research architectures is providing users with more silicon infrastructure to make RHPs easier to use and provide higher performance.

The Dynamic Instruction Set Computer (DISC) is an example of an RHP with enhanced infrastructure for structured computation. The DISC models the GPP paradigm in the sense that it provides users with an instruction set and a programming model that can be targeted by a software compiler. This allows users to create applications for the DISC using high-level languages instead of hardware description languages and schematic capture tools. Where the DISC differs from the GPP is that the instruction set can be configured using instruction modules. As the machine executes programs, custom instructions performing complex operations can be dynamically swapped in. These instructions have to be designed prior to run time, and the compiler must create object code that utilizes these new instructions. [Hut95]

Another class of architecture which provides an enhanced level of performance and ease of use is the embedded core architecture. Embedded core architectures have been pioneered by various groups and has recently made an appearance in commercial FPGAs. These architectures tightly integrate some processing function, implemented at the transistor level, with a reconfigurable hardware array. By integrating commonly used functions into high performance embedded cores, the design effort per application could be reduced while increasing the application performance. The disadvantages of embedded

cores are that not all applications will use the embedded core, and that embedded core technology could be more expensive because it is targeted at a lower volume, higher performance market than the traditional vanilla RHP. [Act97]

2. The Tao of Design

2.1 Introduction to Tao

Because most of the research involving reconfigurable computing has been targeted at niche applications, a general purpose platform for reconfigurable hardware experimentation has yet to be developed. A platform is developed and justified in this section which fulfills that role. Current RHP platforms suffer from low bandwidth and high configuration overhead; the platform presented in this section addresses these issues. [DeH94] Because a key goal of this platform is a balanced architecture for high aggregate data throughput, it has been named Tao. Holistic balance is an important part of the Taoist philosophy.

A high aggregate data throughput is the grail of computer architects; caches, pipelines, interleaving, wide datapaths, FIFOs, and multiport memories—they are all elements of a well balanced system. Without such features, the processor core will starve and system performance will never reach the theoretical peak performance of the processor. Reconfigurable processors promise an even greater performance than traditional processors, but this is a two-edged promise; as peak performance increases, so does the demands on the overall system bandwidth.

The following sections discuss the design goals and engineering tradeoffs in creating the Tao platform.

2.2 Design Goals of Tao

Simply stated, these are the design goals of the Tao platform:

- Provide a platform for experimentation with various types of reconfigurable hardware.
- Balanced throughput from end to end. Ends include video cameras, video displays, host RAM, and hard drives
- Scaleable architecture.
- Aggressive growth curve—implementation technologies must have growth curves similar to that of GPPs.
- Worst case sustained processing at a 60 MB/s data rate.
- Sufficient flexibility to efficiently implement most signal processing algorithms.
- Easy to use—simple low level interfaces and protocols for processing hardware; reusable design elements for fast library-based design.
- Simple—a simple architecture is easier to implement and better for practical reasons.

The architecture in Figure 2-1 was devised to meet these goals; justifications and tradeoffs for each subsystem will be discussed in subsequent sections.

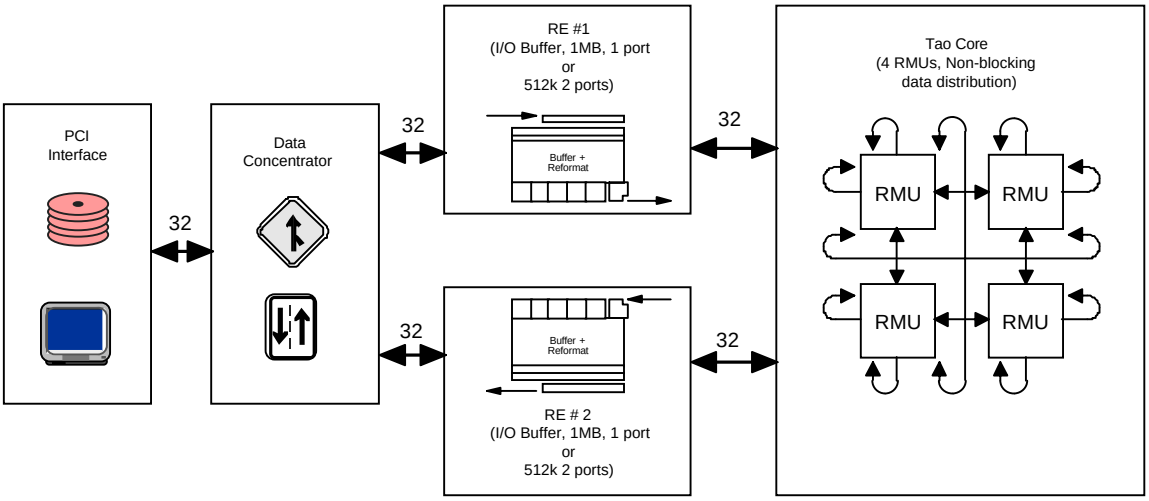


Figure 2-1: Tao Architectural Overview Block Diagram.

The Tao platform sits on a PCI card. The PCI bus was chosen for its ubiquitousness, high sustained performance (132 MB/s peak, 120 MB/s sustained unidirectional), and sophisticated bus features. This bandwidth is split into two channels by the data

concentrator (contained in the GLU FPGA) and buffered by the Reformatting Engines (REs). The REs are pivotal in maintaining a high system throughput as they act as intelligent buffers between the high bandwidth but bursty transactions of the PCI bus and the sustained bandwidth required by the Tao processor core. The REs are capable of performing implementing two-dimensional addressing schemes for the purpose of converting raster order data to block order data, and vice versa. The two REs feed their buffered contents into the processor core, which in this implementation consists of a 2x2 array of Reconfigurable Macrofunction Units (RMUs) embedded in a 2-D toroidal interconnect called the Board Level Routing (BLR). Each RMU contains some reconfigurable hardware along with any support devices one may desire. The RMUs are socketed so that the platform can be upgraded as reconfigurable hardware technology advances. The toroidal topology was chosen for the interconnect between RMUs because it allows adequate scaleability, but more importantly, it keeps the design orthogonal—because there are no interconnection "edges", an RMU design can be plugged into any of the slots without modification or re-wiring.

2.3 Board Level Routing

The heart of the Tao platform is the processor core, which consists of four RMUs and an interconnect matrix called the Board Level Routing (BLR). This section provides an in-depth discussion about the BLR.

2.3.1 Problem Statement and Givens

The BLR must provide a routing matrix which provides sufficient RMU interconnectivity and bandwidth for all common applications. The routing matrix must be efficient, i.e., common applications will use most of the routing capability. It must also be orthogonal such that it provides good scaleability to larger systems. The routing matrix must also fit within the prescribed 180 mm x 180 mm area for the Tao core.

In summary, the routing matrix must be

- extensive: capable of implementing most desired routing schemes

- comprehensive: the routing matrix must be able to connect all combinations of RMUs and REs
- efficient: the routing matrix cannot provide capacity which is rarely used in typical applications
- orthogonal and scaleable: the routing matrix must look topologically identical from each RMU's vantage point, and it must have a direct scaling scheme. In other words, there should be few or no special cases.
- practical: the routing scheme must be implementable within a realistic budget and time frame. The routing scheme must also include adequate support for handshaking and flow control.
- dynamically reconfigurable: the user must be able to modify the routing and RMU configurations while the processor is running. Reconfiguration of each RMU must also be fast, on the order of milliseconds.
- limited global operations: the routing matrix must also have provisions for a limited number of key global operations, such as resets and interrupts.

The fixed factors (givens) which BLR has to be designed around are:

- must fit in 180 mm x 180 mm area
- signals count to RMU is limited; depending on the connector scheme chosen, anywhere between 120 and 180 raw signals are available
- must incur minimal propagation delay to allow nominal board speeds of 33 MHz

2.3.2 Routing Architecture Evaluation Methodology

An iterative refinement process was used in designing a routing architecture. A large number of base architectures were investigated and were subject to projected performance evaluations in certain representative applications. The architecture which best fit the above criteria was then refined and re-evaluated until satisfactory results were achieved. The representative applications chosen for the evaluation were an alpha-blender, a simple video scaler, a 4-element inner-product computer, a quadrature modulator for streaming data, and a simple JPEG-type compressor. These applications were chosen because they

represent a broad range of possible routing and dataflow requirements. Computational intensity had no bearing on application selection. The applications are more precisely defined in the following sections.

2.3.2.1 α-blender

Alpha blending is the pixel by pixel linear interpolation of two images. Given an m x n image $\mathbf{I}_{mn}$, an m x n image $\mathbf{J}_{mn}$ and some blending coefficient α, alpha blending produces an m x n image $\mathbf{K}_{mn}$ such that

$$\mathbf{K}_{mn} = \alpha\, \mathbf{I}_{mn} + (1 - \alpha)\, \mathbf{J}_{mn}$$

The alpha blender implementation considered for BLR evaluation takes two 8-bit data streams loaded as either interleaved or consecutive blocks in one of the two RE buffers. The data is pumped into the BLR and routed to one RMU where the α-blending computation is performed with the option of dynamically changing α-coefficients. The resulting data stream is truncated to 8-bit quantization before being routed back to the remaining RE buffer.

2.3.2.2 image rescaler (decimate and interpolate by rational fractions)

The image rescaler takes in a single 8-bit video stream and outputs a single 8-bit video stream. The algorithm performed is the basic L/M rational scaler (no polyphase form):

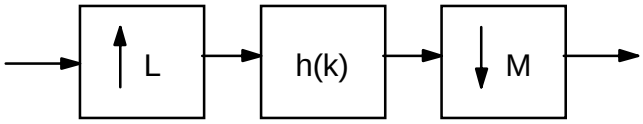


Figure 2-2: Rational L/M image scaling algorithm implementation

The interpolation by L is performed by the source RE, and the decimation by M is performed by the destination RE. The ability to distribute and collate data at multiple rates in the REs increases the flexibility of the platform as a signal processing engine. **h(k)** is some 8x8 FIR filter, and it is implemented using all 4 RMUs. Two RMUs are dedicated to computing the horizontal component of the filter, and the other two RMUs are dedicated to computing the vertical component of the filter. At a BLR rate of 33 MHz,

the system will provide 30 fps performance on 512 x 512 x 8 images when L ≤ 2. The reason this algorithm is considered over the polyphase form is because this division of labor between RMUs resembles the division of labor required for a more general set of signal processing algorithms.

2.3.2.3 4 x 1 inner product with arbitrary vectors and constants, or Quad Binary Operator Sum (QBOS)

This application is a generalized form of computing vector inner products. Recall that a 4x1 inner product has the form

$$\boldsymbol{v} = \mathbf{a_1 b_1} + \mathbf{a_2 b_2} + \mathbf{a_3 b_3} + \mathbf{a_4 b_4}$$

The generalized inner-product form allows one to substitute any binary operator in the place of the multiplies and adds, including non-linear functions. The implementation considered for BLR evaluation uses input samples with 8 to 16 bits of resolution. Once again, this algorithm is not particularly efficient, but it is challenging to implement routing-wise, as each binary operator has two inputs and one output that must be routed via the BLR. Note that Figure 2-3 includes a provision for feedback paths.

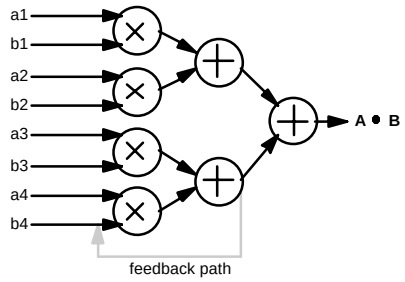


Figure 2-3: QBOS implementation. This variant has a provision for feedback paths.

Note that the input rate is 8 times the output rate.

2.3.2.4 Quadrature modulator

This application demonstrates the capability to perform modulation and demodulation tasks if the platform were used as part of say, a channel simulator in digital wireless communications. This application takes in a stream of 8-bit samples, divides it into I and Q components, and multiplies each stream by either a locally synthesized cosine or sine

function. The modulated streams are added together and sent off to an RE for buffering. The point of this application is to demonstrate the flexibility to perform in non-video applications.

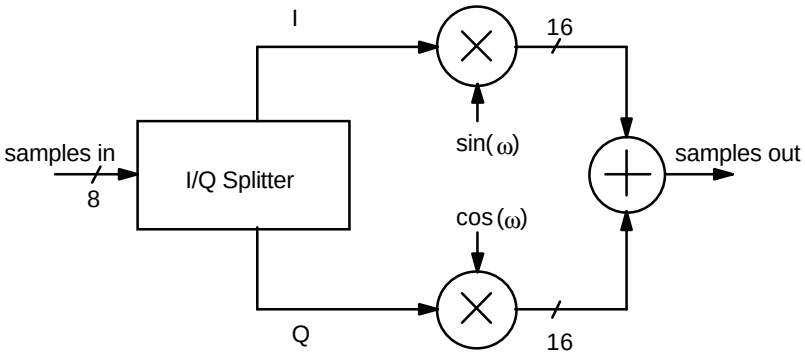


Figure 2-4: Modulator implementation.

2.3.2.5 simple JPEG compression system

This is the stock JPEG compression application. There are many references available describing this application, so a lengthy description of the algorithm is not provided in this text. The key point is that the algorithm can be broken into computational blocks that can be fit within a single RMU. What each computational block does is not important in the current context.

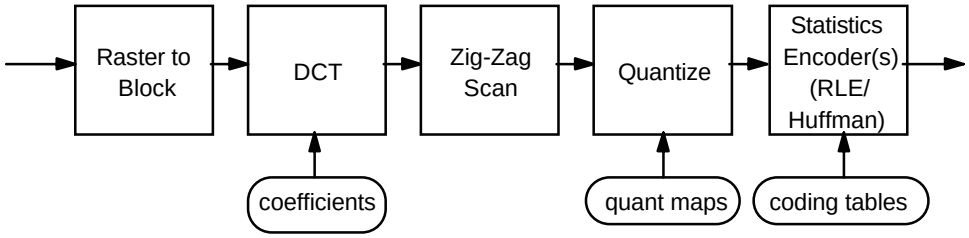


Figure 2-5: JPEG compression implementation [Deb96]

The DCT coefficients, quantization maps, and coding tables can all be reconfigured, which makes this an attractive tool for tweaking aspects of the JPEG algorithm for different applications. Once again, the REs serve as a buffer for the unencoded and encoded data.

2.3.3 Initial Guess—Architecture 1

Figure 2-6 illustrates the details of the first-generation routing architecture.

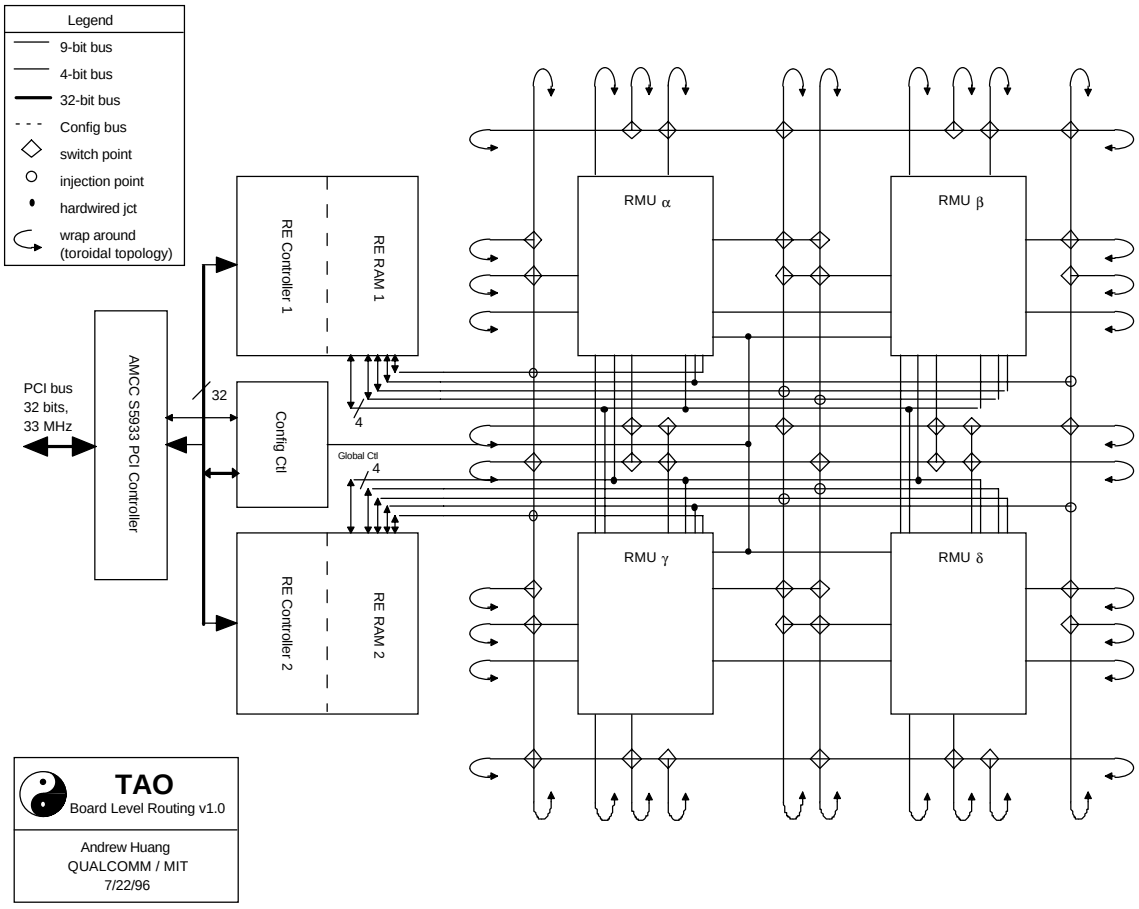


Figure 2-6: First pass architecture

This architecture uses a toroidal routing scheme to eliminate "edges"—thus meeting the orthogonality goal. The switching scheme employed at the intersections of interconnect wires is based on the scheme used inside most FPGAs. Each diamond at the intersection of two wires represents a non-blocking crossbar interconnection. The routing scheme scales linearly with respect to the number of processor nodes in the array. Since the size of the crossbar interconnects is a function of the number of wires crossing at an intersection and not a function nodes, the crossbars stay a constant size regardless of the number of nodes. The routing scheme is sufficient for medium-sized arrays (10 x 10), but it loses its effectiveness as its size approaches infinity (100 x 100 or bigger). At this point it may be

useful to use a hierarchical routing scheme. Although this architecture is orthogonal and scaleable, it has the obvious problem of using too many switch boxes, thus making it impractical to implement within the prescribed 180 mm x 180 mm area reserved for the processor core.

2.3.3.1 Architecture 1 Case Studies

The following diagrams were used to help analyze the architecture for efficiency, comprehensiveness and extensiveness.

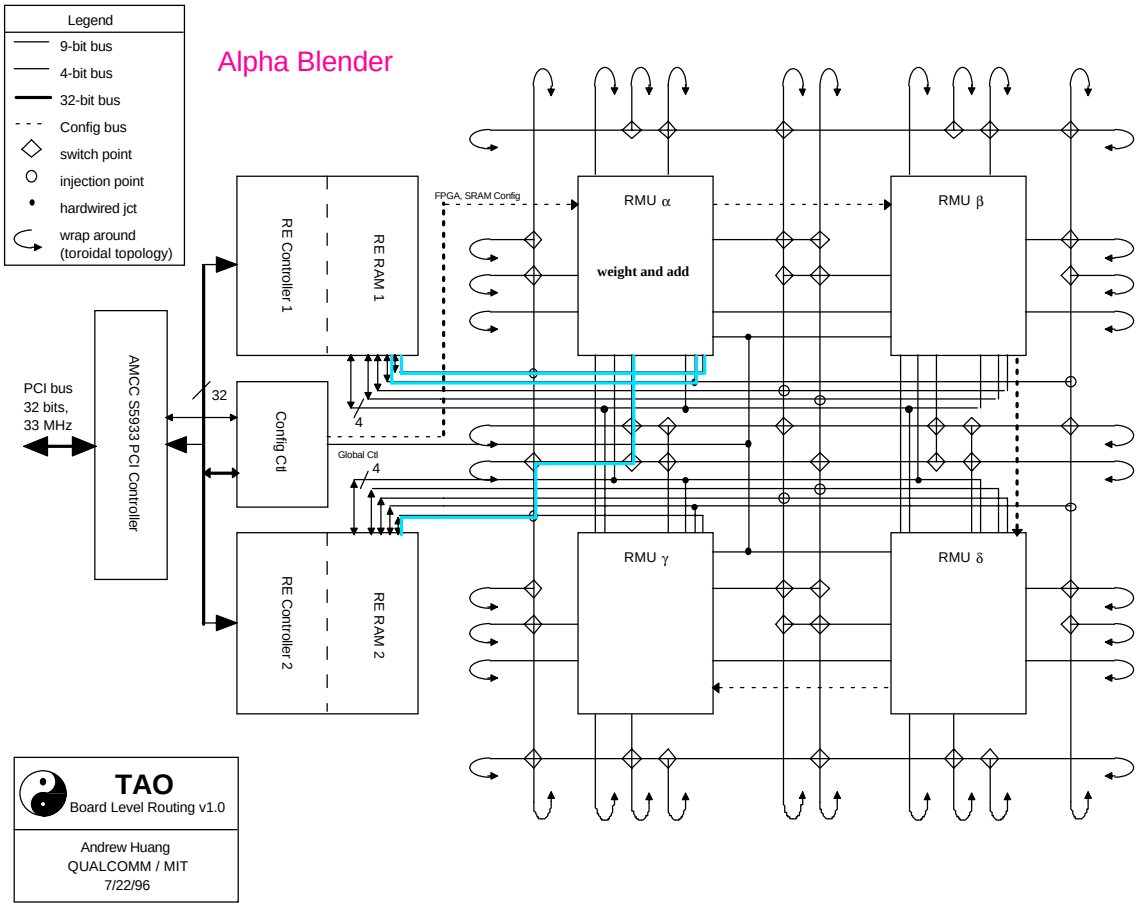


Figure 2-7: Architecture 1 with alpha blender

The alpha blender application is perhaps the most trivial, utilizing a single RMU and minimal BLR resources. The primary challenge in the alpha blender is in the RE, where two video streams must be merged and interleaved.

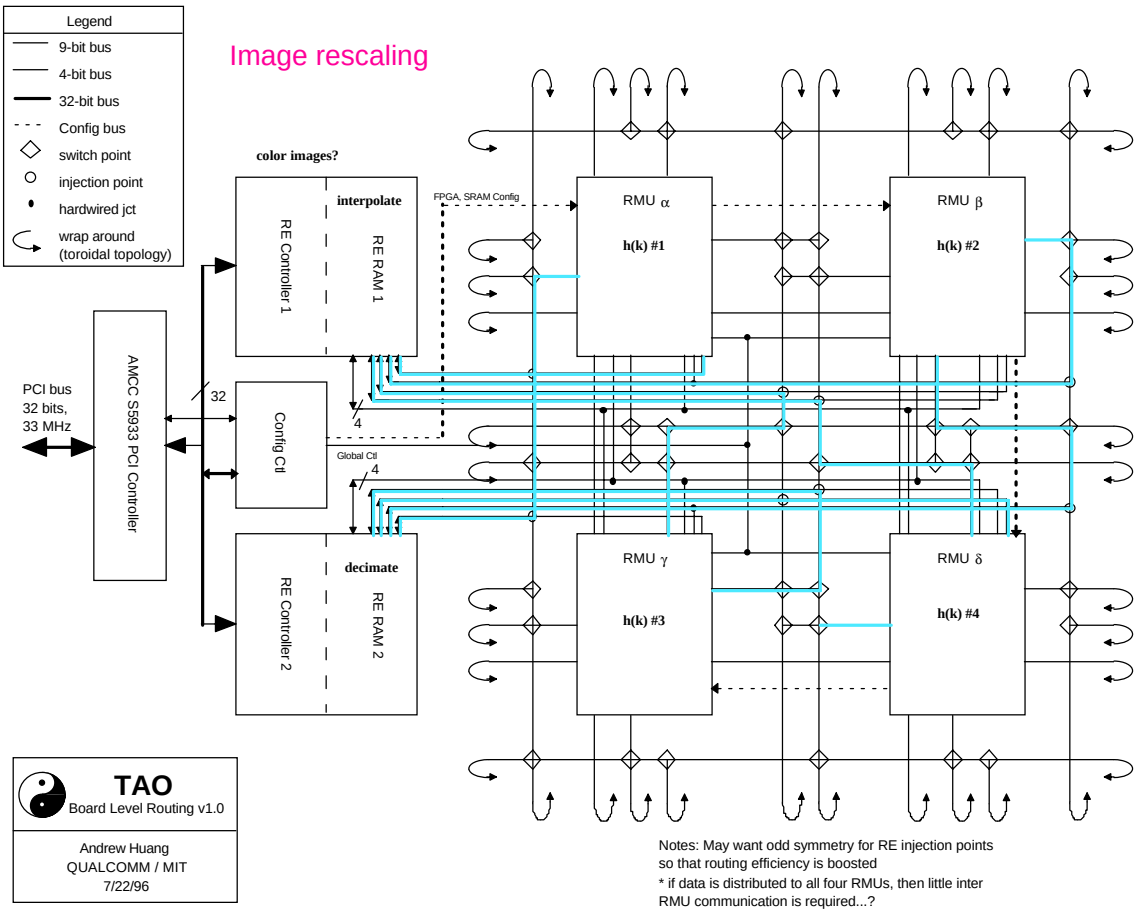


Figure 2-8: Architecture 1 with image rescaler

In the image rescaler application, RE 1 must equally distribute data to all four RMUs. The first generation BLR architecture is biased toward applications requiring two 16-bit streams feeding into two RMUs. Because of this, some BLR resources must be utilized in the distribution of data. The same goes for the collation of data into RE 2.

A significant amount of data shuffling occurs within the REs in this implementation. The REs must simultaneously convert raster data to block data and upsample by padding with zeros or downsample by skipping samples. In addition, the REs may need to provide multiple simultaneous disjoint streams of data, or perhaps time-multiplexed disjoint streams. This implies that the address generation circuitry may have to be replicated fourfold, and that multiple independently addressable SRAM buffer banks must be available within the RE. These rigorous requirements are reflected in section 2.5 which presents the RE architecture.

There is no inter-RMU communication in this example, but there seems to be ample resources available to implement inter-RMU communication if required.

The image rescaler example is a good example of a problem which requires distributed computational elements. Many other problems (general filtering, convolutions, transforms, and motion estimation) can be implemented with a similar topology.

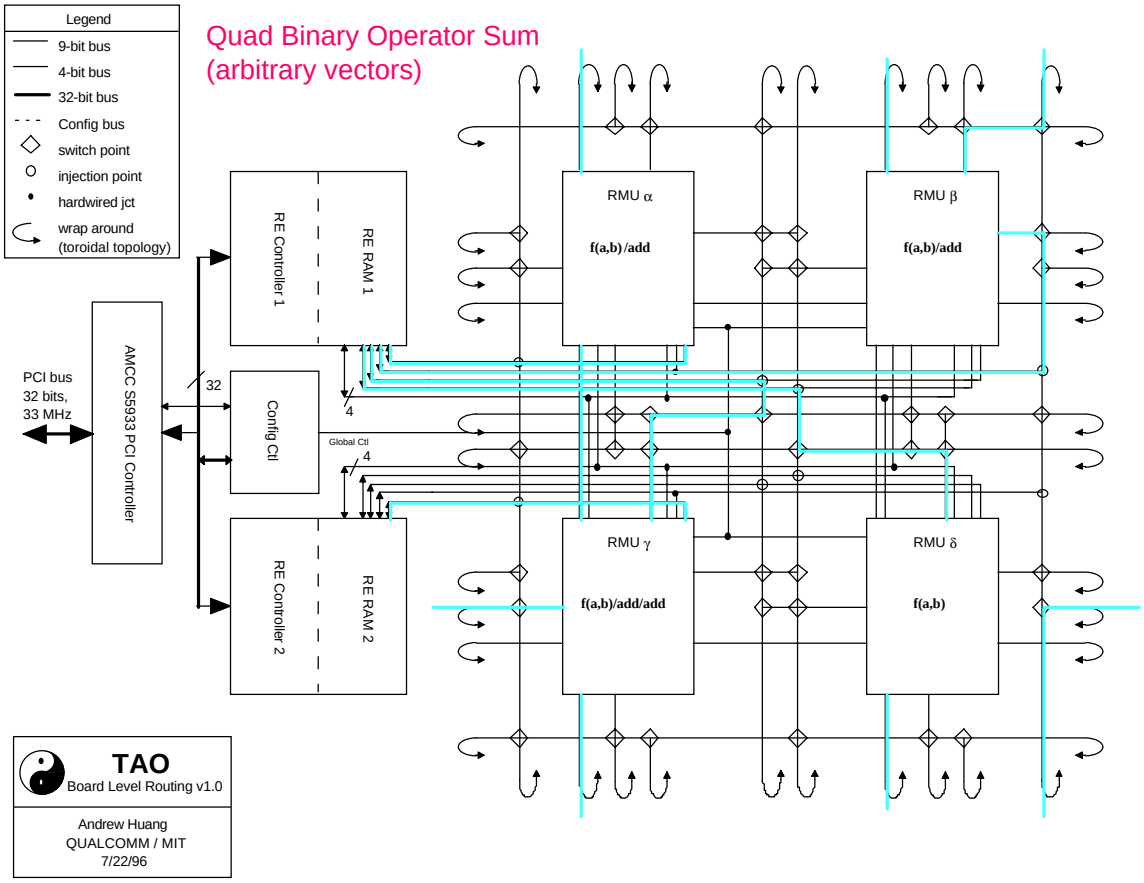


Figure 2-9: Architecture 1 with quad binary operator sum

The QBOS example is perhaps the most BLR-intensive example. QBOS fits comfortably into the current BLR scheme with plenty of room to spare for more inter-RMU communications. Note that this application implementation assumes that the QBOS operates on one constant vector and one variable input vector.

Although QBOS itself is fictitious, many operators resemble the QBOS, including inner products, four-way video mixing/fading, and nonlinear signal processing requiring heavy use of transcendental functions implemented in lookup tables.

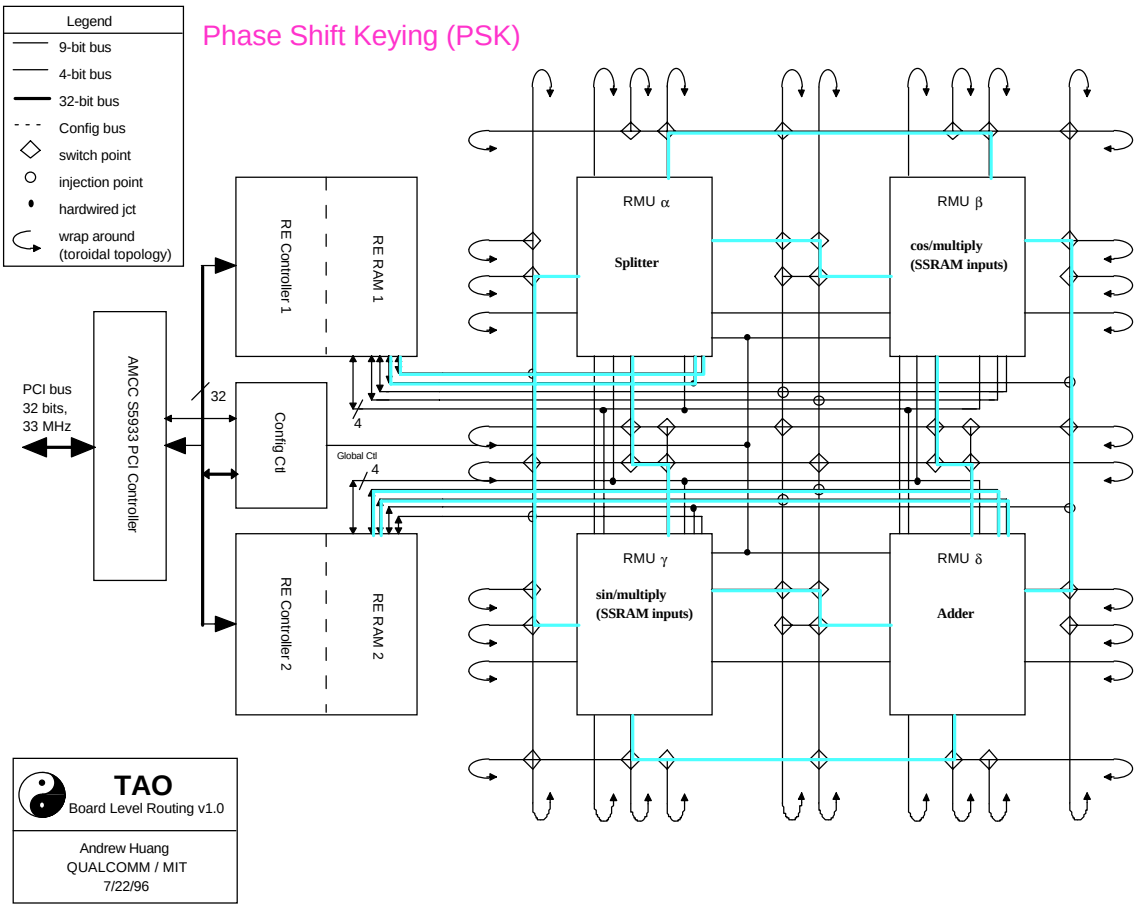


Figure 2-10: Architecture 1 with modulator

The quadrature modulator example is included as a demonstration of Tao's ability to perform operations other than video DSP. Samples can be modulated at a rate of 33 megasamples/second or better. The quadrature modulator example requires fairly intensive inter-RMU communication because of its split-and-combine datapath.

The JPEG encoder example is included as a canonical image processing system demonstration. The example may be slightly unrealistic in its allocation of a single RMU to the Huffman/RLE encoder problem. However, the allocation of RMUs to the DCT problem seems to be fairly real and backed with sufficient evidence [Ber94].

Note that the JPEG encoder example allows dynamic reloading of the quantization tables, so as to give researchers an opportunity to interactively explore the dynamics of perceptual coding schemes. Unlike the previous examples, the limiting reagent in this equation is the availability of computational resources, instead of routing resources. This

is because the basic JPEG algorithm resembles a very deep pipeline with no bifurcations ("string of pearls" algorithm—each computational block has one input and one output, and the blocks are strung together like pearls on a necklace).

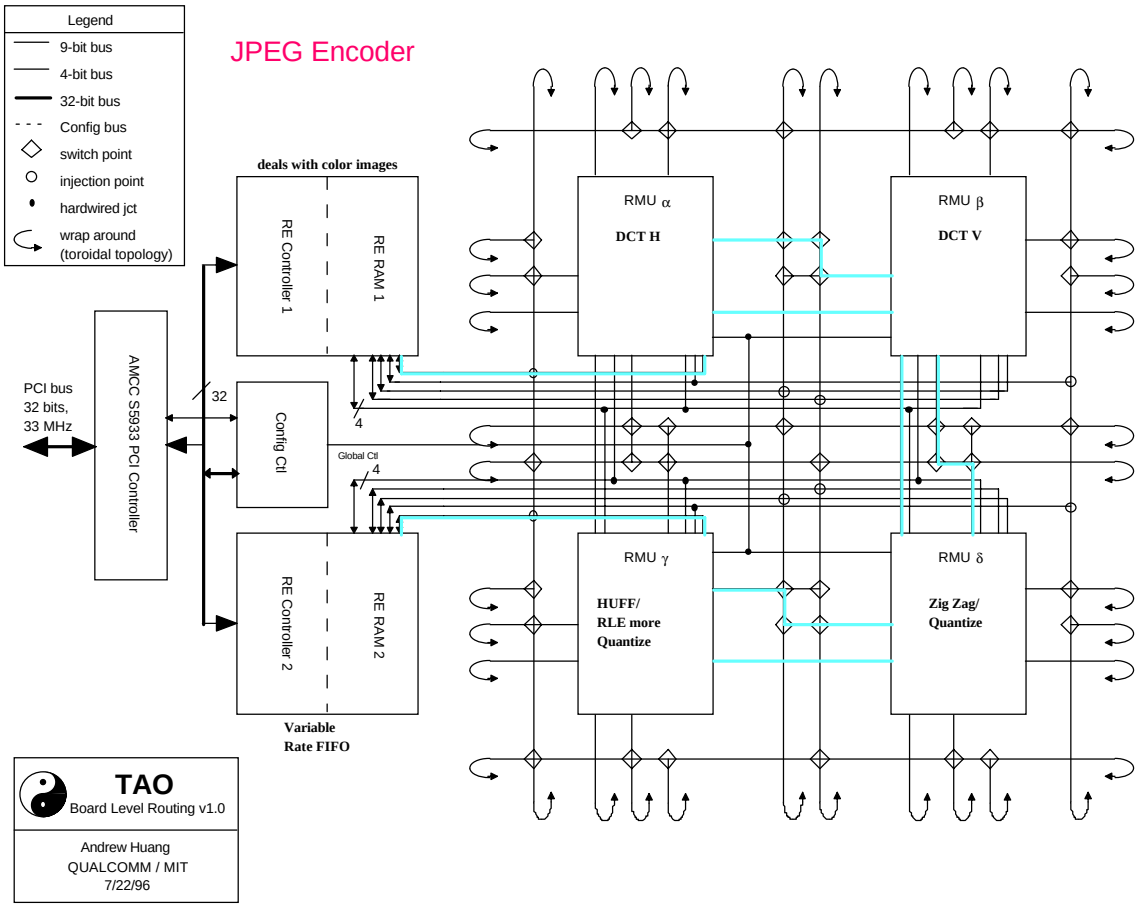


Figure 2-11: Architecture 1 with JPEG encoder

2.3.3.2 Architecture 1 in Review

Architecture 1 satisfies most of the primary objectives of the routing architecture, namely the orthogonality, scaleability, efficiency, and comprehensiveness criteria. The architecture is orthogonal in the sense that it has no edges, and in the sense that from the perspective of each RMU, the BLR looks the same. It is scaleable in the sense that the growth of wires with respect to number of processing nodes is order N. Efficiency and comprehensiveness were demonstrated in the case-study evaluations. However,

architecture 1 is lacking in the practicality criteria. As previously noted, the primary objection about architecture 1 is its unrealistic use of switches.

An analysis of switch utilization in the case-study evaluations has led to a more efficient switching architecture. It turns out that the current switch architecture has too many redundancies and connection pairs that were never used in the case-study applications. The types of switching networks employed in architecture 1 can be broken down into two types. The architecture of the primary (type 1) switchboxes is depicted in Figure 2-12.

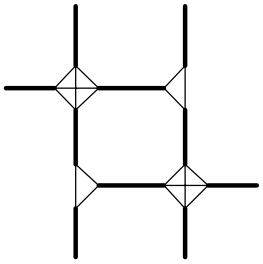


Figure 2-12: Architecture 1 switchbox type 1 architecture. Thin lines are pass gates and each thick line represents a 9 bit bus. 22 switches are required for this scheme.

Type 1 switchboxes are located between RMUs and are used to connect RMUs to the BLR network. Type 2 (diagonal) switchboxes are located on the diagonals between RMUs and are used to connect wires to wires. The current scheme places a degenerate crossbar switching network at each bus intersection.

A new switchbox based on a partial crossbar topology involving more busses is proposed in Figure 2-13.

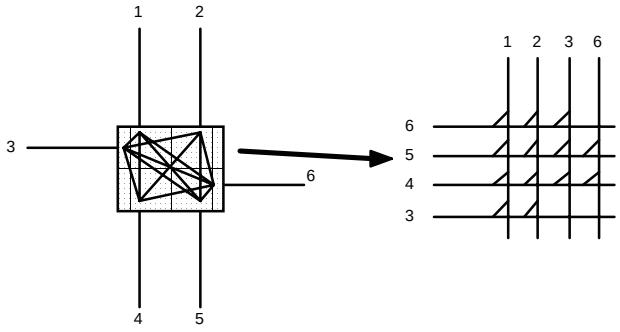


Figure 2-13: Switchbox configuration. Each line represents an 9-bit bus. 13 switches are required to implement this scheme.

The switchbox in Figure 2-13 turns out to be equivalent to the switchbox in Figure 2-12. The following analysis breaks down the possible interconnection pairs and evaluates their purpose in the routing architecture.

- inputs 3 and 6 are RMU injection points—these are wires which connect the RMU to the BLR matrix
- inputs 1, 2, 4, and 5 are BLR routing lines—they connect between switchboxes
- some routing pairs will never occur
    - 1 will never route to 2: this is not useful, as that will loop a signal right back to the sending switchbox
    - 4 will never route to 5, same reasoning
- some routing pairs are marginally useful
    - 1 may route to 5 or 4, and 2 may route to 5 or 4, for the purposes of routing "long" signal runs (farther than one switch-box hop)
    - diagonal routing idea (1-5, 2-4) has dubious value, since the RMU injection points are fully associative (connected to all possible inputs)—when would one use such a junction? The only reason might be to route around a pre-allocated line which is "in the way" of a long signal run. However, since the FPGA injection points are fully associative, an intelligent router could compensate for this by moving the shorter run out of the way.
    - straight routes (1-4, 2-5) are frequently used

By trimming route pairs of dubious value, crossbar switch count can be reduced from 13 switches down to 11 switches. Since the switches come in packages of four, this gain in area efficiency is worth the loss of dubiously useful switches, as the package count per switchbox will be reduced from four to three. So as not to waste a switch, one diagonal will be preserved, bringing the number of switches up to 12. Thus, the revised switching matrix has 12 switches, implemented with three QS34X245 Quad 8-bit bus switches and three QS3125 Quad single switches (the QS3125 is needed for the ninth flow control bit).

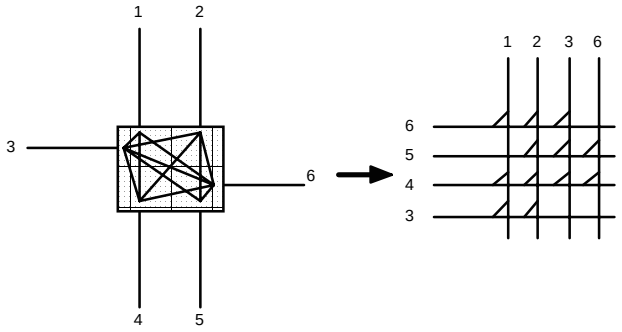


Figure 2-14: Simplified switchbox with one connection pair (1-5) removed to bring number of switches down to 12.

Analysis of the type 2 (diagonal) switchboxes reveals that they are redundant in the case of the 2x2 RMU array. Diagonal switchboxes are used to provide right angles on routes which span distances longer than two RMUs. Their location in the BLR is illustrated in Figure 2-15. In the 2x2 case, all routes are single hops between Manhattan neighbors. These routes will never require the diagonal switchbox because adjacent Manhattan neighbors are always accessible with a straight wire route. In the degenerate 2 x 2 case, a diagonal neighbor is also accessible without a diagonal switchbox.

Diagonal switchboxes are somewhat useful in the 3x3 case, and are critical in the 4x4 case. Thus, even though they will be eliminated for practical reasons in this implementation, they are required for larger designs. As an additional note, a larger design will also require more routing resources. Double-length wires may become necessary, as they are in the internal Xilinx 4000 series architecture. Wide wiring channels with more degrees of freedom will also be desired to accommodate long-run wires. Although this doesn't sound scaleable O(n), it is—in the asymptotic limit, which is reached in the 4x4 or 5x5 case. By the time a 4x4 case is implemented, full diagonal routing boxes and double-width routing with 32 bit (quad 8-bit) channels (as opposed to the current 16 bit (dual 8-bit) channels) will be required. Scaling the wiring density any further than this will provide diminishing returns for larger designs. The primary reasons for cutting back on the wiring and switches in this implementation are 1) Tao is a proof-of-concept design and 2) Tao must fit in the form factor of a double-height PCI card. With more space, a full routing matrix is feasible.

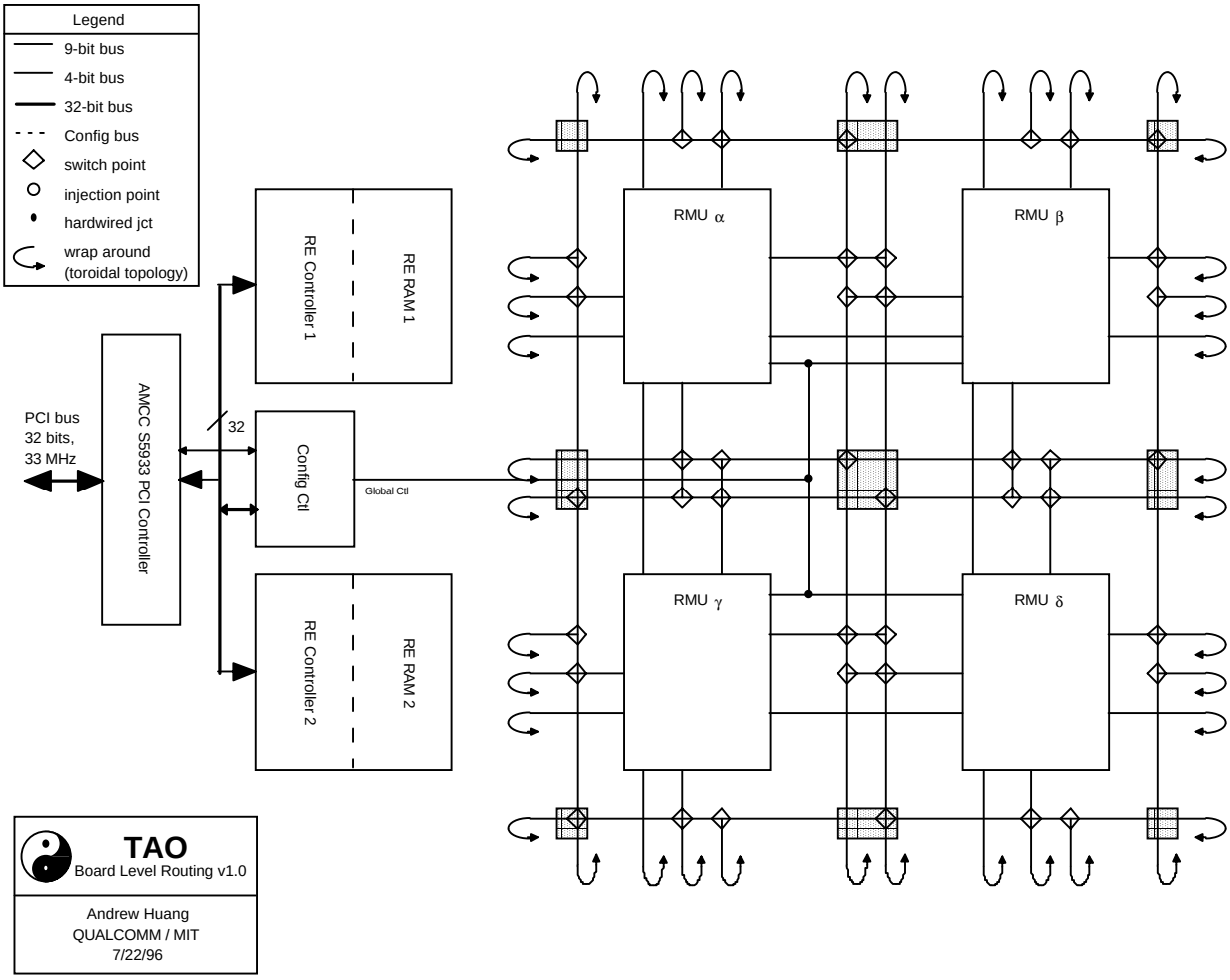


Figure 2-15: Diagonal switchboxes are highlighted with gray boxes.

Removing the diagonal switchboxes has the side effect of requiring the data distribution scheme from REs to RMUs to be augmented. Instead of distributing data solely through direct and vertical-wire channels, a horizontal distribution path shall be added. This will completely eliminate the need for diagonal switchboxes with only a marginal increase in the data distribution complexity.

Finally, in evaluating the BLR scheme, one finds that the SRAM of the REs will have to be segmented into several banks. Access times will need to be lower than 28 ns for a clock period of 33 ns with 5 ns of margin. SSRAM is optimal for this purpose, as it is easy to design with and pipeline depth is not as important as throughput. The RE

SSRAM section has provisions for expansion in the form of daughtercard connectors and physical mounts.

2.3.4 Architecture 2

Figure 2-16 illustrates the improved architecture. Notice that the switchboxes have been revised, the diagonal switchboxes deleted, and the data distribution injection points reorganized.

Although architecture 2 preserves the routing density of architecture 1, the parts count has been reduced. Table 2-1 summarizes these results.

Category	Architecture 1	Architecture 2
Number of bus T-Gates	1600	896
Number of bus T-Gate Packages	50	28
Number of injection points	8	16

Table 2-1: Architecture resource usage comparison.

The designs affected by the optimizations employed in Architecture 2 are the QBOS and the image rescaler. Figure 2-17 and Figure 2-18 shows these designs implemented with Architecture 2.

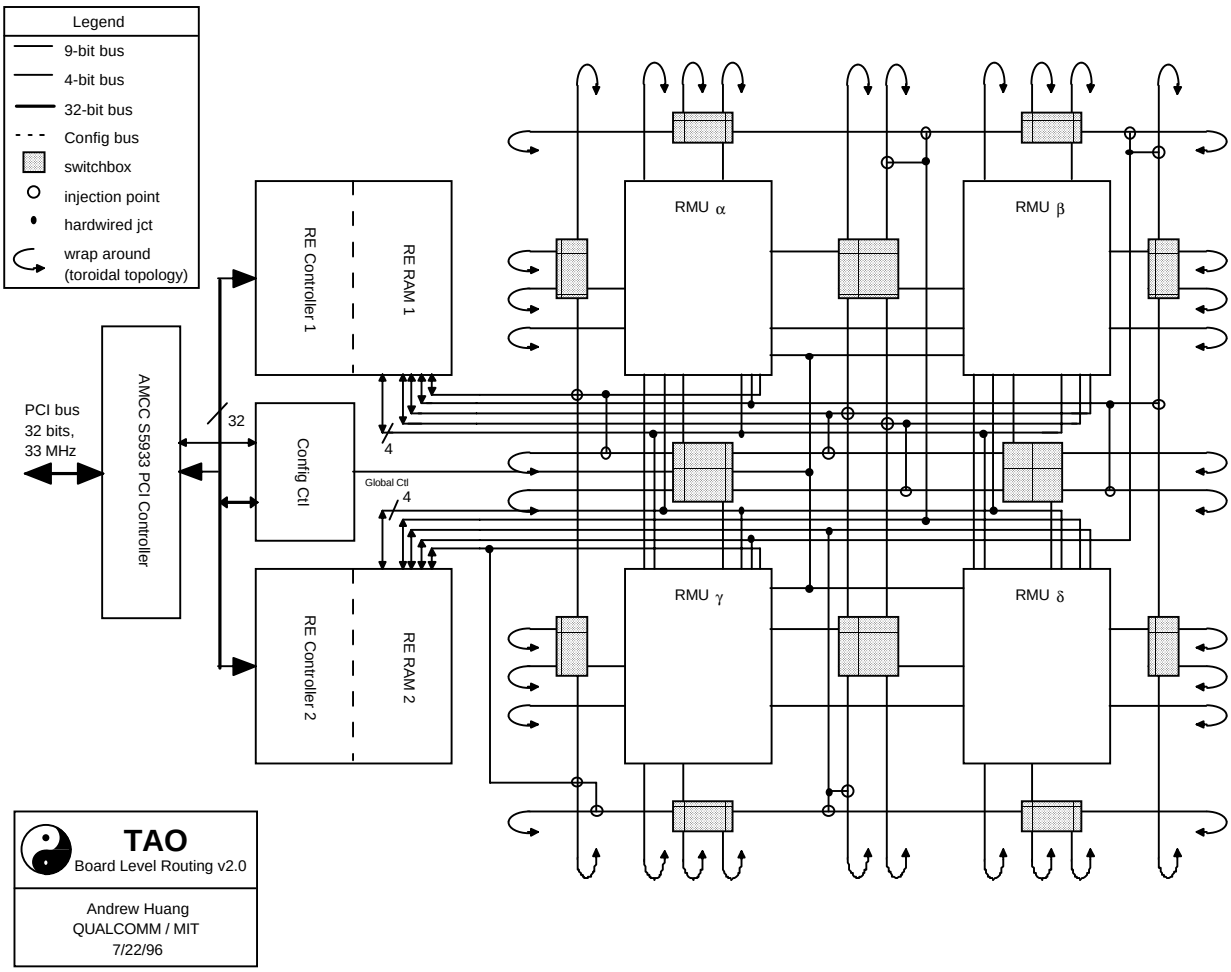


Figure 2-16: Architecture 2

Architecture 2 represents an improvement over Architecture 1 in terms of implementation feasibility and T-gate allocation efficiency.

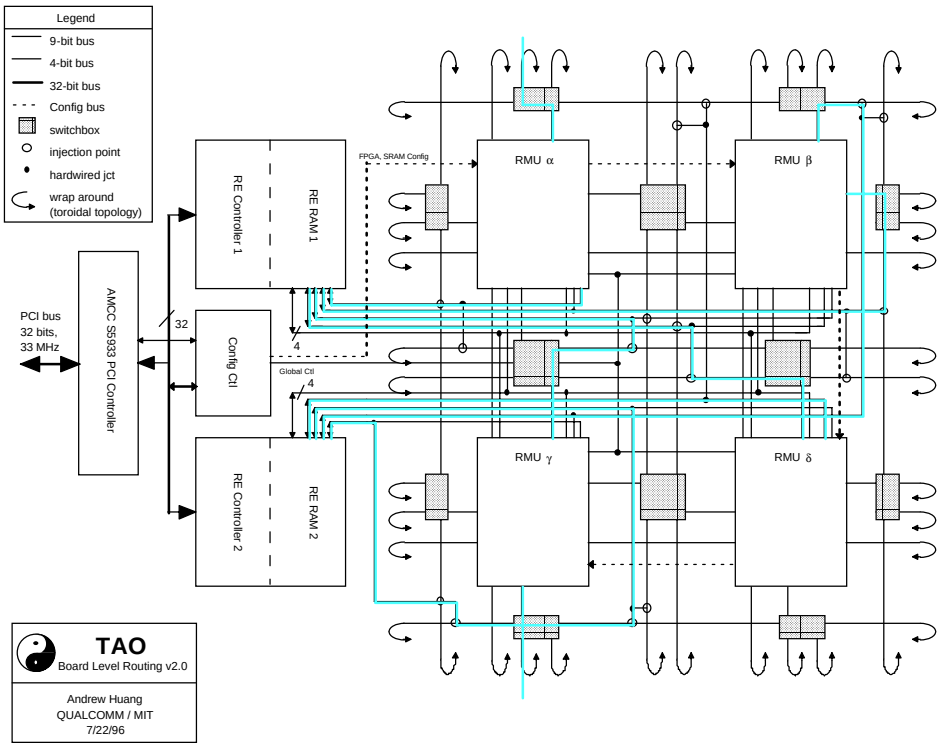


Figure 2-17: Image Rescaler with Architecture 2

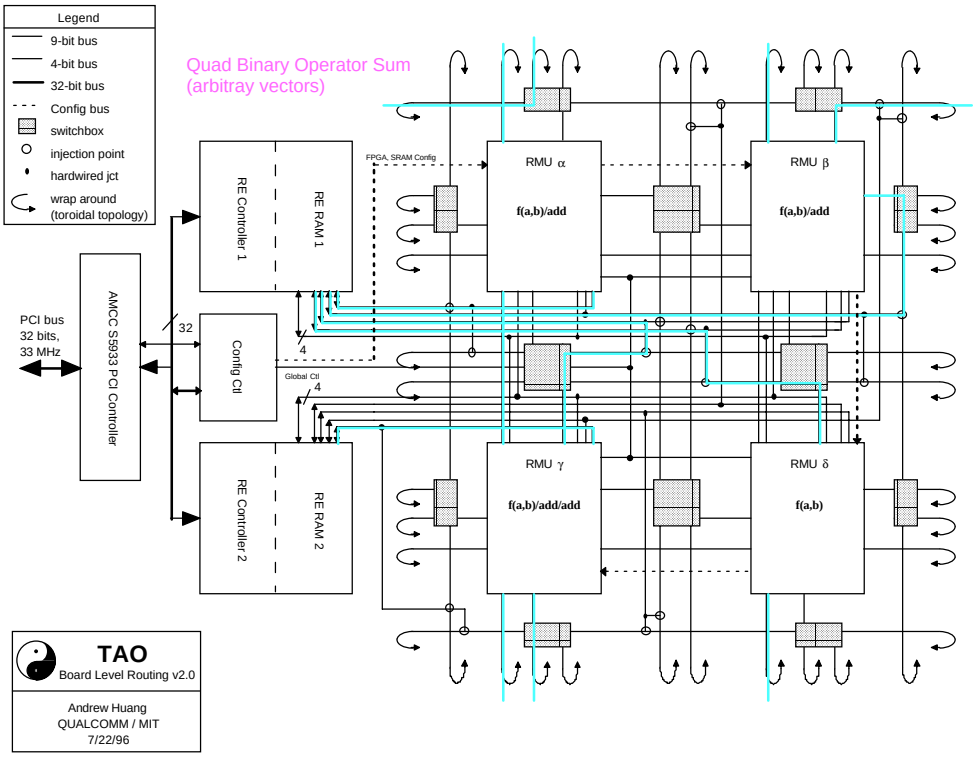


Figure 2-18: QBOS with Architecture 2

2.3.5 Architecture 2 for Implementation

A careful examination of architecture 2 reveals that there are a number of topological equivalents which can yield a significant improvement in board layout and parts utilization. All the optimizations presented in this section rely on the fact that the 2x2 case is a degenerate case. In order to scale the matrix up, it would be necessary to remove these topological optimizations.

The following three figures illustrate a topological evolution of the BLR matrix.

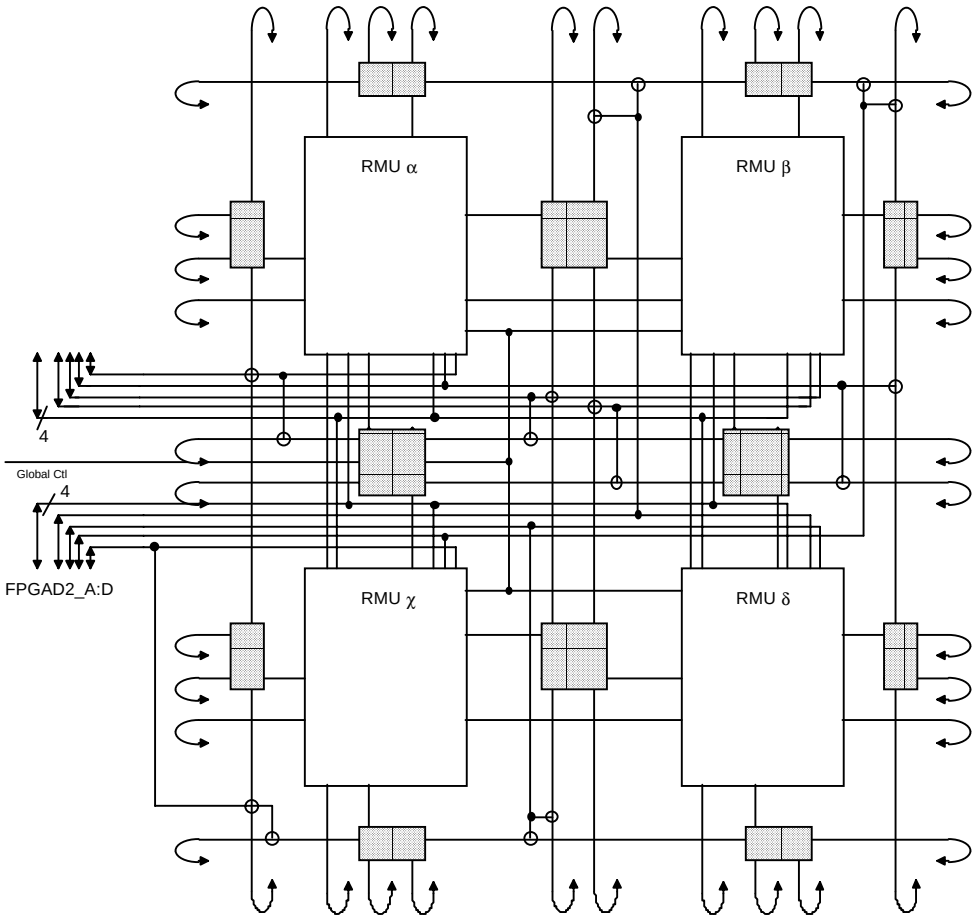


Figure 2-19: BLR at step one of the evolution (original Architecture 2)

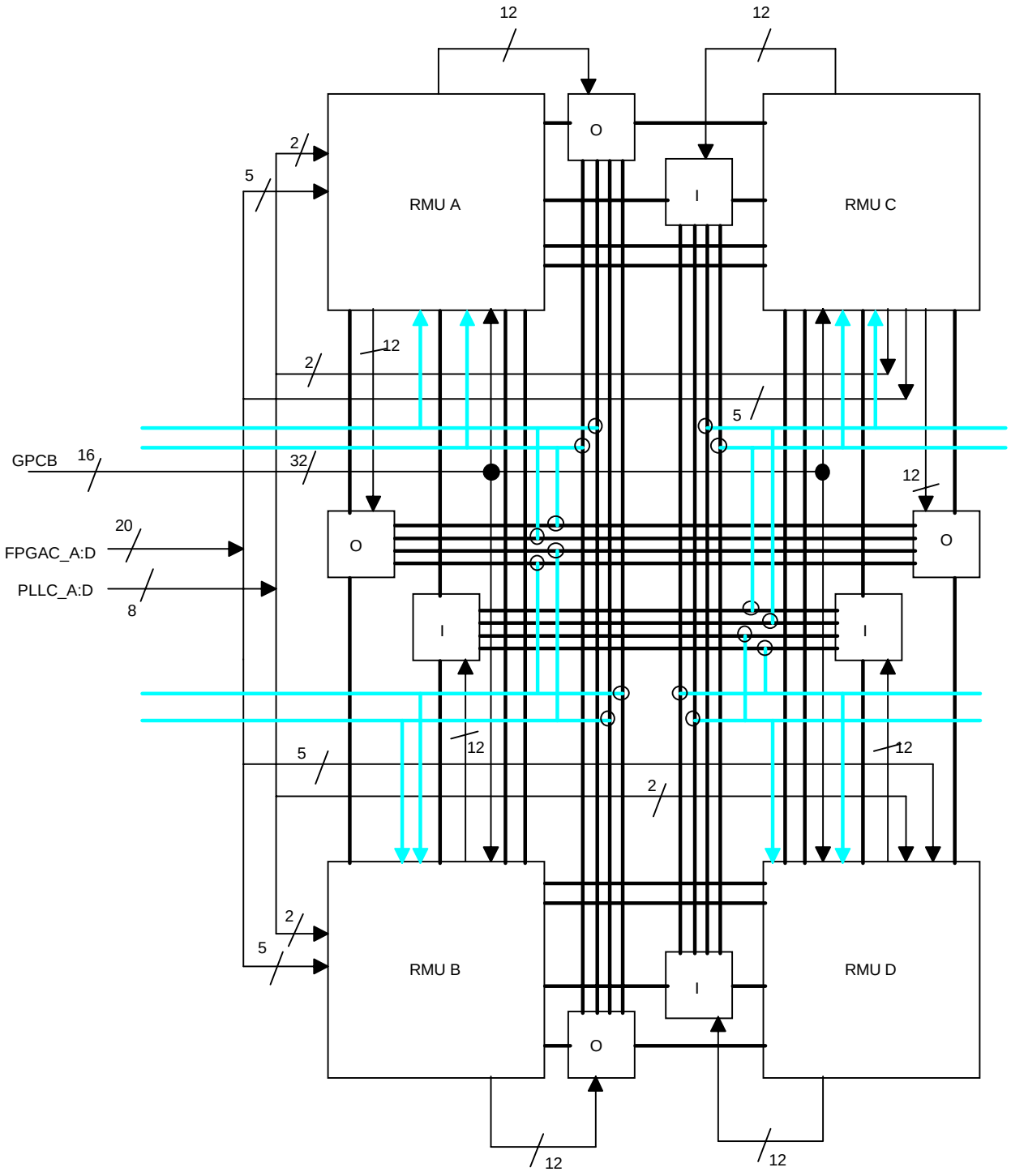


Figure 2-20: BLR at step two of the evolution. Wrap around signals have been folded inside. Gray lines are 9 bits wide. Data enters from both sides of diagrams as (FPGAD{1,2}_A:D).

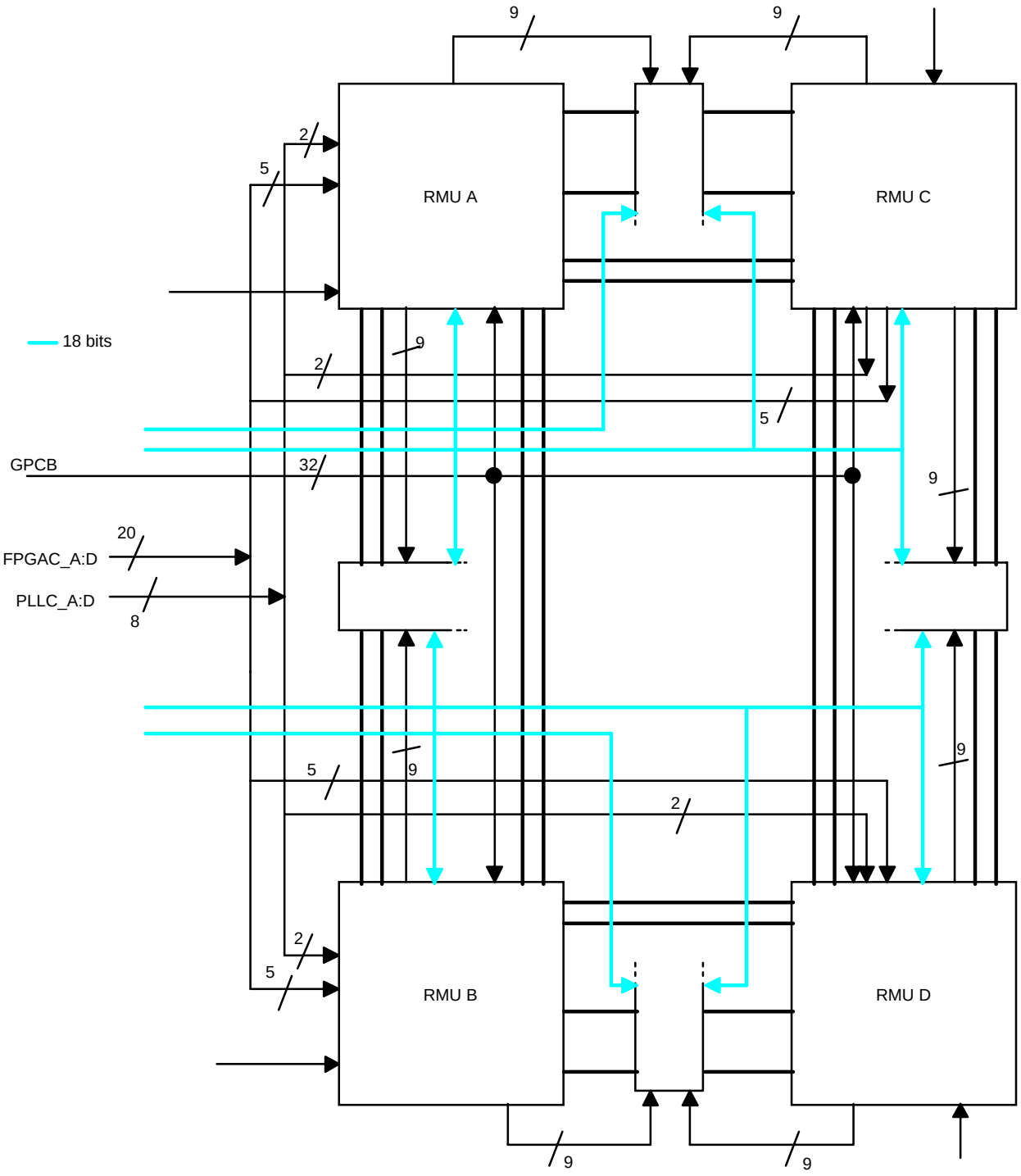


Figure 2-21: Step three of the BLR evolution. Switchboxes have been condensed. Gray lines are 18 bits wide.

Figure 2-19 depicts architecture 2 with the original topology, featuring a true toroidal wrap-around topology. Figure 2-20 depicts the BLR with the wrap-around signals flipped inward. It is the logical equivalent of the original architecture with a different physical layout. Figure 2-21 depicts the BLR with a final optimization: the switchboxes have been condensed. The savings on the required number of switches by applying these topological optimizations is summarized in Table 2-2. The condensed switchboxes have a topology depicted in Figure 2-22.

Category	Architecture 1	Architecture 2	Evolutionary
Number of bus T-Gates	1600	896	320
Number of bus T-Gate Packages	50	28	10
Number of injection points	8	16	16

Table 2-2: Architecture resource usage comparison, revised. Note that these numbers are for 8-bit buswitch packages. An equivalent number of packages will have to exist for the "9th bit" switches.

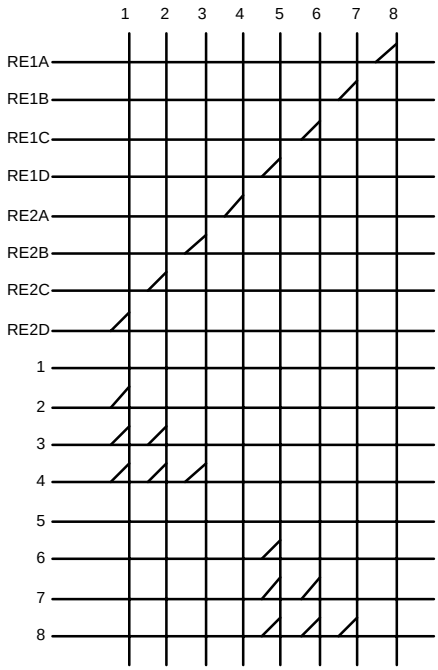


Figure 2-22: Condensed switchbox topology. There are two switchboxes total, and each RMU gets half of each.

Note that with the condensed switchbox topology, the number of controls for switches goes down, but now all RMUs must be installed for proper operation of the entire matrix. This will be a problem when debugging; a possible approach is to provide a manual override to switchbox configuration signals via hard jumpers in case an RMU is not installed or operational.

2.3.6 Globally Available Signal Resources

2.3.6.1 Introduction

The Tao architecture features, in addition to routable resources, globally available resources. These resources are chiefly used for control purposes—reset signals, backward stalls, global framing, exceptions and interrupts, coefficient distribution, and operating mode signals. Global resources can also be used to augment local routing resources when routing is very difficult.

2.3.6.2 Resource Allocation

There are 16 signals allocated to general purpose global communications. 16 more signals (the General Purpose Configuration Bus (GPCB)) are allocated to handling the "warm-start" configuration of RMUs. A warm-started RMU is one in which the bootstrap FPGA has been configured. There are a total of 20 lines (5 lines per RMU) provided for bootstrapping the RMUs.

2.3.7 Handshaking and Pipeline Protocol

2.3.7.1 Introduction and Definitions

The Tao hardware supports a forward and backward stallable pipeline protocol. Forward stalls occur when an upstream element fails to produce data on time. Forward stalls could be caused by input starvation, variable length coding schemes (e.g., statistical redundancy compression), source decimation, and computational overload. Backward stalls occur when a downstream element cannot accept data. Backward stalls could be caused by full

FIFOs, variable length coding schemes (e.g., statistical redundancy decompression), destination interpolation, and computational overload.

Forward stall logic is always required since the pipeline must always be synchronized to the availability of input data. Backward stall logic is optional because a data bucket (FIFO) is always available to buffer output data. Because of the performance hit associated with them, backward stalls should be avoided if possible.

2.3.7.2 Implementation and Example

The pipeline protocol implemented in Tao uses the following three classes of signals:

- *Global reset*—a signal which causes all computational elements to enter a known starting state. All pipeline registers are considered to contain invalid data after a global reset. The schematic name for global reset is `G_R`.
- *Forward flow*—a set of signals, one for each 8-bit bus of data, which goes high to indicate the presence of valid data on the bus. Note that the forward flow signal is local to each 8-bit bus run, except for the runs connected to the REs. The REs generate a globally available set of forward flow signals. The schematic name for forward flow signals is `ST_F#`, where # is any letter A-Z, ranked by distance from the pipeline entrance. All equidistant pipe stages share the same rank suffix. Pipe registers of one stage generates the next rank's data signal, e.g. flip flop A generates output A (`Q_A[#:#]`), which turns into data signal B (`D_B[#:#]`) after being processed by combinational logic.
- *Backward flow*—an optional signal or set of signals which applies backpressure to the pipe. Backward flow signals are implemented using the global routing resources. This is because pipe data is incompressible and one cannot insert bubbles as done during forward stalls—hence, back stalls must cause the *entire* pipe to freeze simultaneously, forward stall signals included. The back-stall signal needs to have a high drive capacity since it may have to drive several pipeline stages' worth of registers.

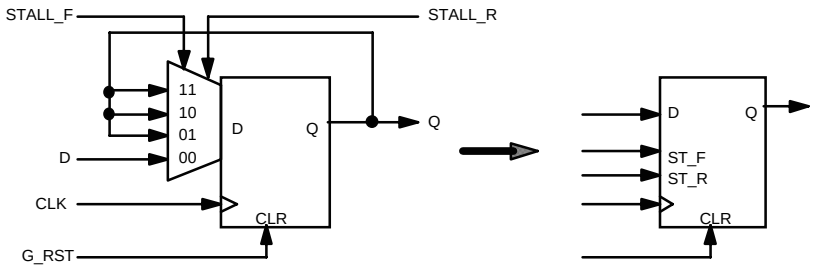


Figure 2-23: Bidirectional stall flip flop with clear (BSFF). Unidirectional stall flip flop (USFF) has a similar configuration.

The core element of the pipeline protocol is the bidirectional stall flip flop with asynchronous clear (BSFF). Please see Figure 2-23 for a diagram. It is essentially an Enabled Flip Flop (EFF) with dual enable signals. The unidirectional stall flip flop (USFF) is an EFF with asynchronous clear. The BSFF is efficiently implemented in the Xilinx 4000 series architecture with a half CLB—the BSFF requires one register and one function generator to implement the mux.

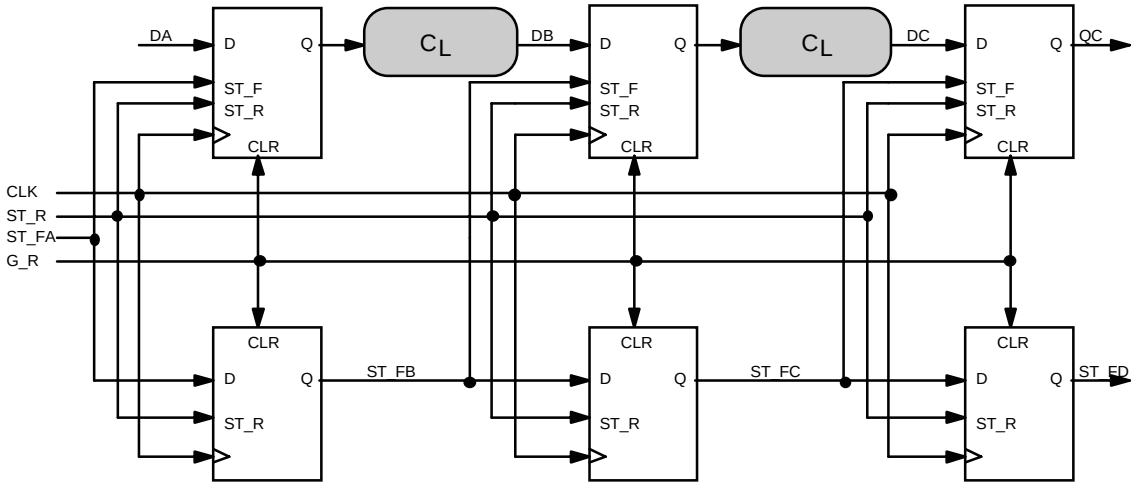


Figure 2-24: Example pipeline using BSFF

Figure 2-24 and Figure 2-25 illustrate a pipeline using the protocol described in this section. It is a three stage pipeline surrounding two clouds of combinational logic. The timing diagram illustrates a moderately hairy situation—when forward and reverse stall signals collide. Note that a forward stall will never be issued if a reverse stall is in

progress, because reverse stalls will affect the issuer of the forward stall—any forward stall issue in progress at the time of a reverse stall cannot terminate until the reverse stall has ended.

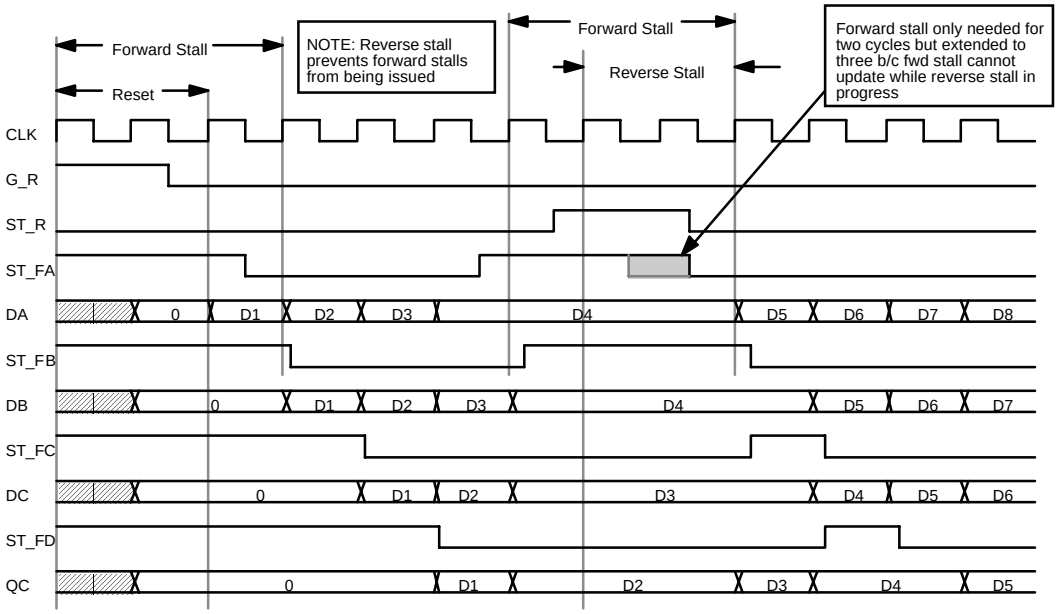


Figure 2-25: Timing of example pipeline using BSFF.

2.4 Reconfigurable Macrofunction Unit (RMU)

2.4.1 Introduction

A reconfigurable macrofunction unit is the computational core of the Tao. The constraints applied by the BLR are the primary design considerations. Aside from satisfying those constraints, an RMU is free to use any kind of computational element. This helps give the Tao platform a longer life span and greater flexibility. By implementing the core processing elements on removable cards, one can update the processors as commercially available reconfigurable hardware technology progresses. It also allows one to mix and match various types of hardware; for example, the use of ASICs or GPPs within an RMU is not ruled out. This section develops the RMU as an abstraction and also a specific hardware implementation of an RMU. Thus, an RMU need only comply with the I/O port

specification, the configuration scheme, and the clock management scheme. More information on the configuration scheme can be found in section 2.6 and information on the clock management scheme can be found in section 2.7.

2.4.2 Interface Specifications

2.4.2.1 I/O Ports

I/O port organization is driven by the requirements of the BLR; hence, the RMU must comply to these external specifications. For a justification of the I/O port organization, please consult section 2.3, *Board Level Routing*.

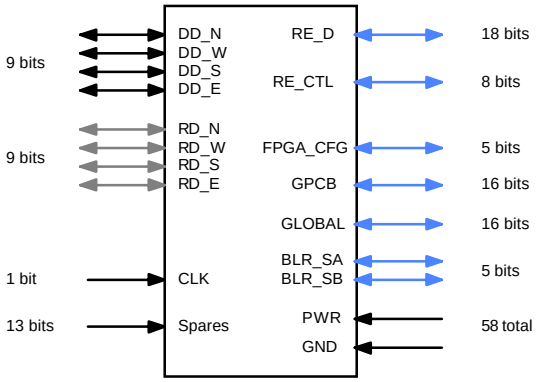


Figure 2-26: I/O port organization for an RMU. Pin count = 210.

Figure 2-26 details the I/O port organization specification for an RMU. Signal names have the following meaning:

DD_[NWSE]	Direct data from nearest Manhattan neighbor FPGAs
RD_[NWSE]	Routed data from nearest Manhattan neighbor switchboxes. All RD signals are 3.3V compliant.
RE_D	Reformatting engine data, D = Direct
RE_CTL	Reformatting engine flow control information, distributed globally
FPGA_CFG	FPGA configuration signals (CCLK, DIN, PROG, INIT, DONE)
GPCB	General Purpose Configuration Bus interface (for SSRAMs, etc.)

GLOBAL	Global communication bus interface (reset signals, hacks, etc.)
BLR_S[AB]	BLR switchbox control signals.
CLK	Clock signal, 33 MHz
PWR, GND	Power signals (5V, 3.3V, current return GND)
Spares	Spare pins (NC)

The RMU connector on the motherboard uses two AMPMODU® .050 grid 80 pin count male surface mount connectors and one AMPMODU® .050 grid 50 pin count male surface mount connector.

Protocols for various busses are described in section 2.3.7, *Handshaking and Pipeline Protocol*, section 2.6, and section 2.5.

2.4.3 Configuration scheme

The RMU must obey the following configuration rules:

- Upon global configuration reset, the RMU must tristate all outputs and prepare for total reconfiguration
- Data will first be provided on FPGA_CFG; this will configure the primary bootstrap FPGA on the RMU. The primary FPGA should take care to initialize its PLL configuration outputs at this time.
- Data will then be provided on the GPCB for configuration of secondary FPGAs, SSRAM modules, and routing switches.
- Fully configured, the RMU may enter run mode when instructed, either via command on the GPCB or a GLOBAL signal.

Configuration protocol details are covered in section 2.6.2.2, *GPCB Protocol*.

2.4.4 Clock Management Scheme

Each RMU is required to internally regenerate the clock via a local PLL. This helps reduce clock skew. It also provides a key feature: the local PLL can generate design-specific clocks. Since the operation speed of an FPGA relies heavily on the outcome of

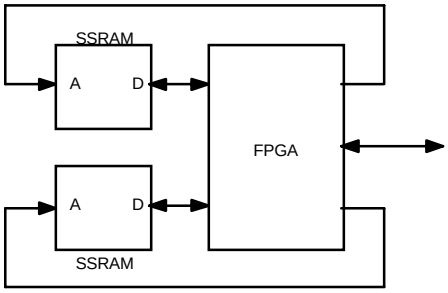
the place and route hardware compiler, the local PLL can generate a clock signal tailored to the design. If a design routes well, the user may choose to double the clock speed and gain a factor of two in performance. If the design calls for long combinational delays, the clock speed can be halved. System synchronization issues are left to the end user.

2.4.5 RMU Internal Architecture

A single general purpose RMU was designed for this thesis work. This RMU features two SSRAM LUTs (SLUT, pronounced "ess-lut"), one FPGA, and a PLL clock generator.

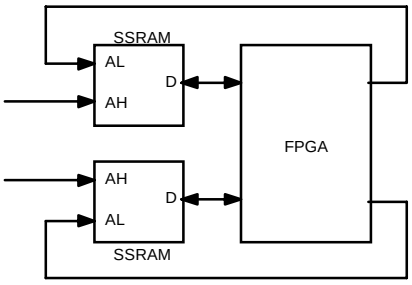
The SLUTs can function as either local data storage, local coefficient storage (e.g., quantization tables, DCT coefficients, etc.), or as a computational element. The SLUT has a computational advantage over the FPGA in situations where highly nonlinear functions need to be represented. Such non-linear functions include logarithms, transcendental relationships, gamma correction curves, square roots, number recoding, and saturating rounds. The SLUT can also compute very fast, low-latency 8-bit binary operations, such as multiplies and adds, if necessary.

The following diagrams illustrate possible configurations of the SLUTs and the FPGA and some of their potential applications.



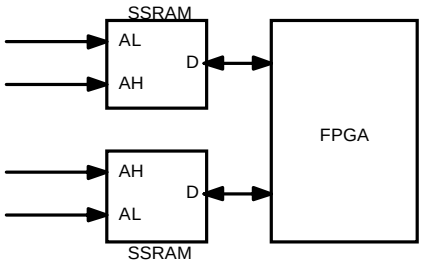
All SLUT addresses supplied locally.

- SLUT as a buffer for reformatting
- SLUT as a local LUT
- SLUT as a FIFO



Half of the SLUT addresses supplied locally, half from external I/O.

- SLUT as a binary operator



All SLUT addresses supplied externally, either from single or multiple sources

- SLUT as a binary operator (mixer operation)
- SLUT as a flow-through computational LUT

Figure 2-27 is a complete block diagram of the internal architecture of the RMU.

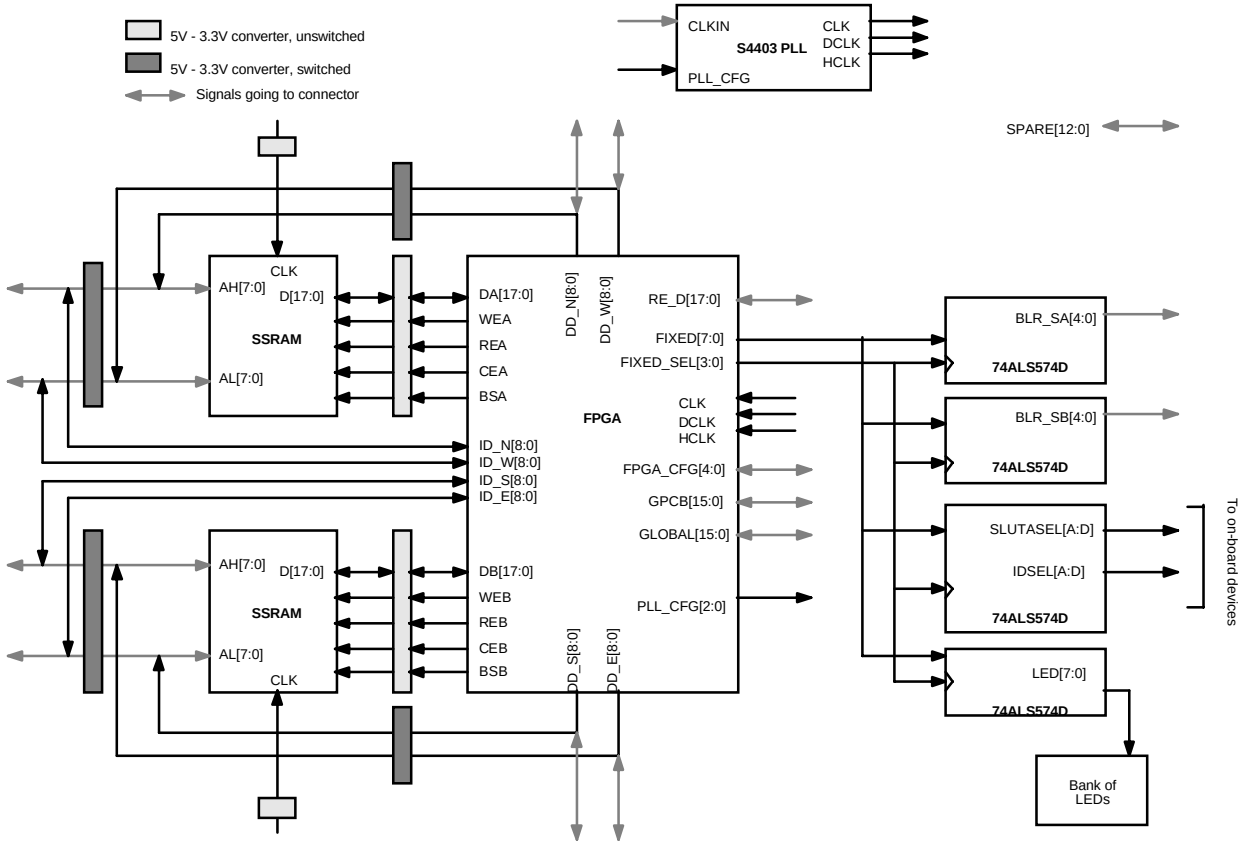


Figure 2-27: Complete block diagram of the proposed RMU. FPGA I/O count=181.

Because the FPGA is I/O limited, the direct data pins (DD[NWSE]) are multiplexed between functioning as direct data pins and as local address for the SSRAM. This is an acceptable compromise since the use of any of these functions is usually mutually

exclusive. Also, all of the signals which need to be configured once or infrequently throughout operation are demultiplexed and latched with addressable registers. The FIXED[7:0] and FIXED_SEL[3:0] busses form the interface for these signals. The only exception to this rule are the PLL_CFG[2:0] signals. These signals need to be fixed upon FPGA configuration in order for the RMU to be functional; hence, they cannot be demuxed by the FIXED[7:0] auxiliary bus.

2.5 Reformatting Engine (RE)

2.5.1 Introduction

The reformatting engines (REs) play a crucial role in achieving a high sustained system performance. The Tao platform is in a situation where incoming and outgoing data on the PCI bus may flow in several short, high bandwidth bursts, while the internal dataflow must be maintained at a constant rate. Hence, each RE must have sufficient capacity to insure that the Tao core does not starve or overload given typical I/O conditions. Also, the incoming and outgoing data streams will usually be in raster format, while most algorithms which run on Tao will expect data in block format. The requirements of an RE are

- buffering capacity for one frame of data
- sufficient bandwidth to keep up with maximum output and input rates
- raster-to-block and block-to-raster conversion capability
- flexible data distribution to Tao core
- decimation by sample deletion and interpolation by zero-padding capabilities
- easy upgrade path to larger buffers
- easy upgrade path to include direct video input and output support

A pair of symmetric REs are provided on the Tao platform, one to handle incoming data to the core, and one to handle outgoing data from the core to the PCI bus.

2.5.2 2D Addressing

All video processing systems must deal with the task of converting a 1D stream of data into a 2D format to which most signal processing algorithms are native. To complicate matters, many algorithms require that the 2D image data be subdivided into smaller blocks, sometimes with complex addressing requirements such as zig-zag scanning. The Tao processor provides a native 2D addressing environment to simplify the task of implementing many algorithms.

Figure 2-28 summarizes the 2D addressing nomenclature. The variables **x** and **y** refer to major indices. Boldfaced variables may not take on the full set of natural numbers, **Z**, as valid values. Also, a variable-naught, such as $\mathbf{x_o}$, are constants with the maximum value of that index. For example, if an image were 128 x 32, then $\mathbf{x_o}$ equals 128. The variables i and j refer to minor indices within sub-blocks. Thus, to convert between linear (1D) addressing and 2D addressing,

$$\begin{aligned} \text{linear address} &= ((\mathbf{x} + i) + (\mathbf{y} + j) \cdot \mathbf{x_o}), \\ \mathbf{x} \in \{n \cdot i_o\},\ \mathbf{y} &\in \{m \cdot j_o\} \text{where } n, m \in \mathbf{Z} \\ \mathbf{Z} &\in \{\ 0, 1, 2, \ldots\ \} \end{aligned} \tag{2-1}$$

The canonical view of memory as a linear space drives hardware and software constructs, such as pointers, to be designed for linear addressing. There are numerous methods of translating 2D addresses to linear addresses, with a typical tradeoff of complexity versus flexibility. The following pseudo-code describes a method for reading out 2D blocks from a flattened raster using four hardware counters, i, j, x, and y.

```
if (i == i_o) => j++, i = 0
if (j == j_o) => x += i_o, j = 0
if (x == x_o) => y += j_o, x = 0
if (y == y_o) => end
linear address = i + x + (y + j)·x_o
```

It is a design goal to break with the linear addressing tradition and use multidimensional addressing. Thus, all memory address spaces will be specified in terms of coordinates: (x, y), or (x, y, i, j), depending on the number of dimensions required by the application.

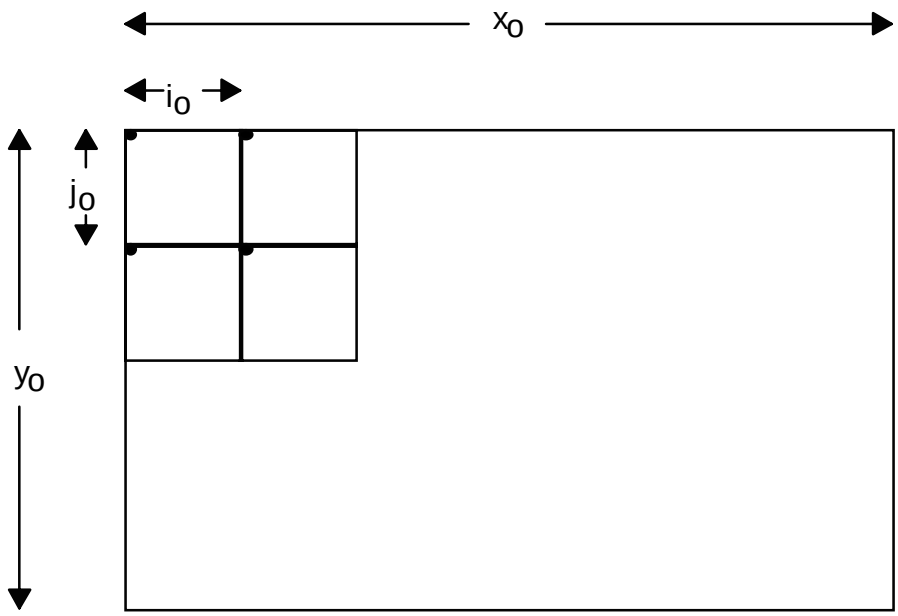


Figure 2-28: 2D addressing variable nomenclature. Black dots indicate valid points for x_o, y_o.

Translating these multidimensional coordinates to device addresses is straightforward. An address word will be broken into several parts, with each coordinate receiving the dimensional length it requires. Thus, for the case of (x, y) addressing, one possible address breakdown would be (assuming memory is organized as 256k x 16):

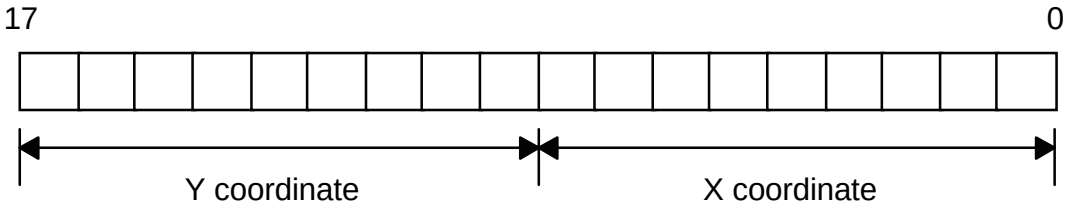


Figure 2-29: Physical address breakdown for 2D scheme.

This addressing scheme allows images up to 1024 x 512 x 8 bits to be addressed and stored, with a granularity down to 2x1 blocks. Four dimensional addressing, as in the case of (x, y, i, j), is similar:

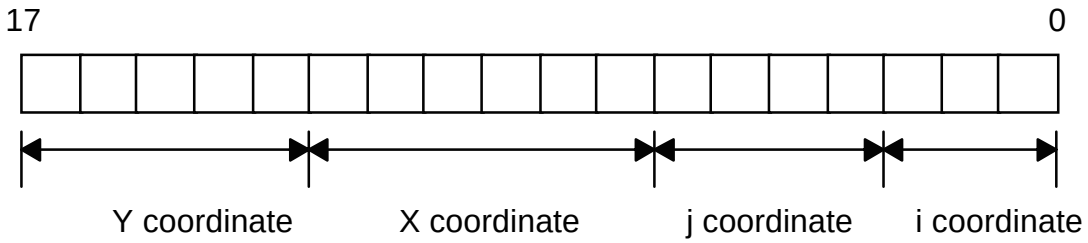


Figure 2-30: Physical address breakdown for 4D scheme.

This allows an image of up to 64x32 16 x 16 x 8 bit blocks (1024 x 512 x 8 total resolution) to be stored and addressed on an individual block level. By storing images in this format, random access of blocks is straightforward, and if zig-zag scanning is desired, the lower seven bits can be run through a 256-bit look-up table ($2^7 \cdot 2$, to compensate for the 16-bit organization of the memory) to generate a zig-zag pattern. As a note, 256-bit look-up tables can be implemented with just 4 CLBs in a Xilinx 4000 series FPGA.

2.5.3 RE Architecture

On the physical level, an RE consists of an FPGA and a mezzanine memory card. The FPGA provides the control interface, and the mezzanine memory card allows for an easy upgrade of the SSRAM buffer. The RE has a 33 MHz clock buffered from the PCI bus clock, and a 66 MHz clock generated by a local PLL clock synthesizer.

2.5.3.1 SSRAM Buffer

The SSRAM buffer can be configured for any of three operation modes: double buffered half-rate, double buffered full-rate, and single buffer full-rate. Figure 2-31 illustrates buffer operation in the three modes. The double buffer modes allow simultaneous reads and writes, and it is useful for operations on real-time datastreams, such as those generated by video cameras. Real-time datastreams tend to “trickle” at a constant rate, so the ability to read and write simultaneously is important. The half-rate double buffered mode exists because some FPGAs have slow I/O drivers and cannot keep up with the 66 MHz bus rate. The single buffer mode allows only exclusive read or write access per bus cycle; this mode exists because it is the easiest to implement, and because it is well-suited for stored data streams. Stored data streams come from host RAM or hard disk, where the dataflow

pattern tends to be infrequent, short, full-bandwidth bursts, grouped by read or write. In other words, the data must come and go on the PCI bus, so the maximum throughput is reduced to 66 MB/s because the bidirectional data streams have to be merged into a single unidirectional pipe. Instead of trickling in a frame, a frame write or read must finish to completion before the buffer changes direction. Although the performance of a single buffer is limited to 66 MB/s, it is sufficient for most purposes.

Also, the 32-bit data bus for the single-buffer mode is partitioned into two 16-bit busses with independent address generation. This allows a single image to be split into halves so parallel units can process simultaneously.

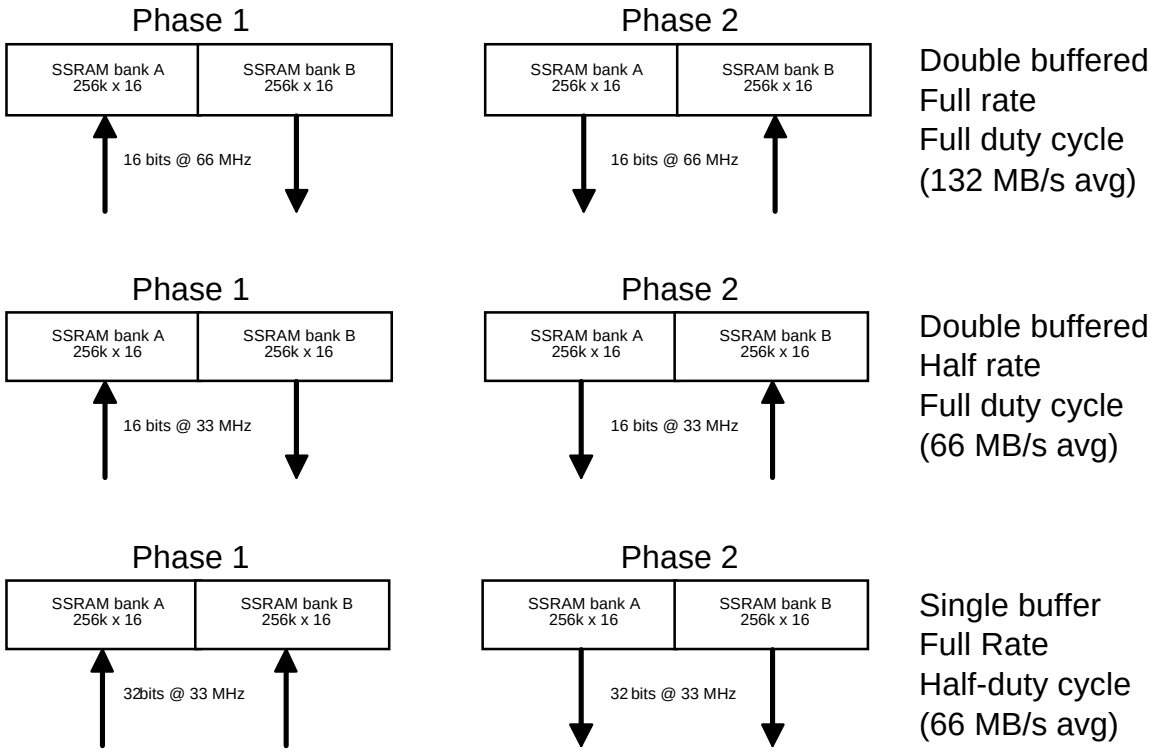


Figure 2-31: Buffer configurations.

2.5.3.2 Flow Control; Interpolation and Decimation

The RE uses the standard flow control interface described in section 2.3.7, *Handshaking and Pipeline Protocol*. In the case of a single-buffer mode RE used in the input path, data

will flow into the RE from the Glue FPGA until the RE detects that a full frame has been written. While this is happening, the Tao core is stalled. After this point, the RE switches to read mode and pushes data into the Tao core at a constant rate, until the data is exhausted. The RE signals completion, switches back to write mode, and the cycle begins anew. The RE is sensitive to global stall signals only during RE read mode. During RE write mode, global stall signals do not affect buffer filling since it should be able to hold an entire frame worth of data.

Interpolation of data is performed by inserting zeroes between samples using a multiplexer. Zero insertion control can rely on cues provided by the parity bits of the buffer SSRAMs, since 2-D interpolation can be tricky. Decimation of data is performed by either stalling the pipe or by skipping addresses in the address generator.

2.5.3.3 Video I/O Support

The RE can also be upgraded to handle direct video input and output. This feature could alleviate some or all of the load on the PCI bus. Direct video input works by sharing buffer SSRAM in dual-port mode between a digitizer or an IEEE 1394 Firewire™ interface and the RE. Direct video output occurs in a similar fashion. The datapath infrastructure for supporting direct video exists as part of the base design, and only the control signals are provided for direct video support. A total of 5 lines per RE are provided for direct handshaking, and an additional 5 configuration lines connected to the CE are provided for any additional FPGAs which may be on the mezzanine.

2.5.3.4 Summary Diagram

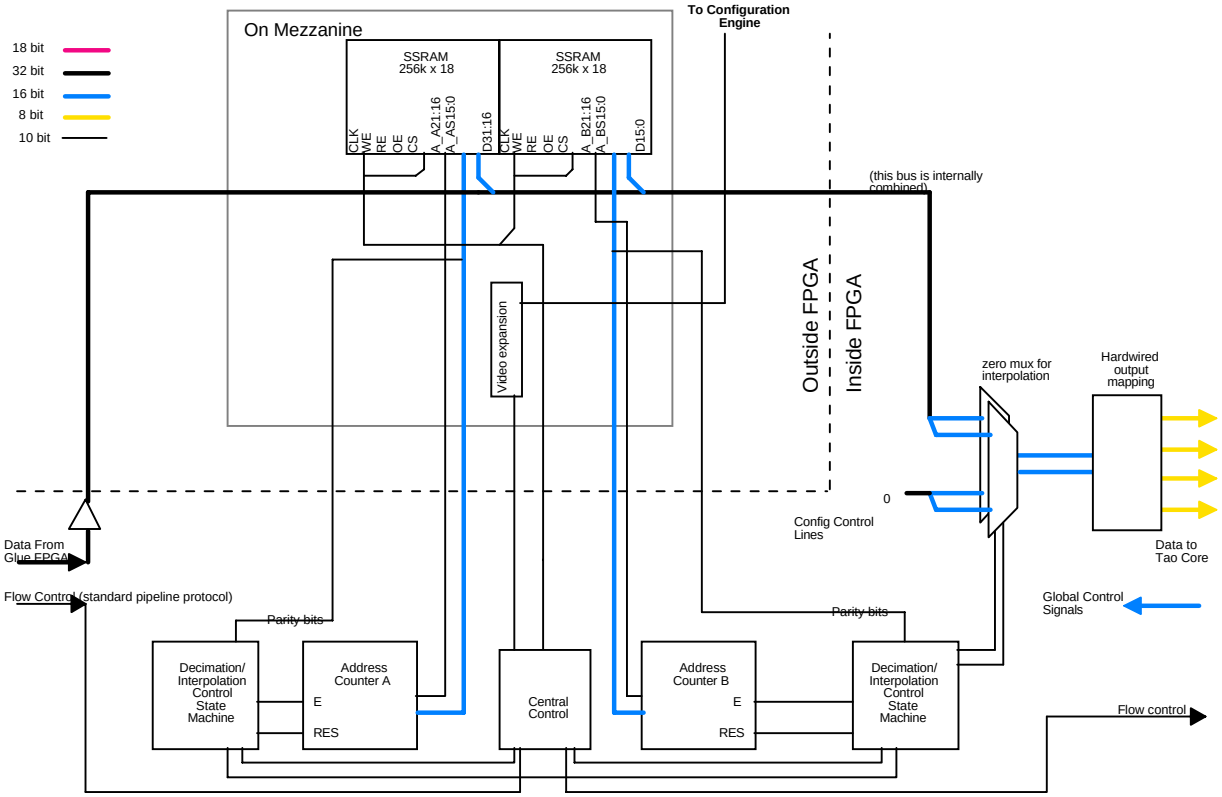


Figure 2-32: Block diagram of RE for single buffer mode. Mezzanine signal count: 266 for two REs. RE FPGA I/O count: 184. Diagram does not include 5V to 3.3V conversion logic.

The bottom line on RE design philosophy is to get maximum functionality for minimum design and debug effort. Because the interface between the buffer mezzanine and the RE FPGA is so flexible, dataflow balancing between the PCI bus and the Tao core is possible for a number of scenarios.

Note that 5V to 3.3V conversion logic is performed with Quickswitches as described in Quality Semiconductor's AN-11A application note. [QSIAN-11A]

2.5.4 Pin Count Budget Summary

Mezzanine card

Buffer Data	32
FPGA Config Data	5
Parity Bank A	2
Parity Bank B	2
High Address A	6
High Address B	6
Low Address A	16
Low Address B	16
Buffer Control	10
Vide expansion control	5
Total Signals	100
pwr/gnd percentage	33%
Total Pins for one RE	133
pwr/gnd pins	33
Total Pins for two RE	266

RE FPGA

Data from Glue	32
Control from Glue	4
Data to Buffer	32
Control to Buffer	10
Parity Bank A	2
Parity Bank B	2
High Address A	6
High Address B	6
Low Address A	16
Low Address B	16
Data to Tao Core	32
Control to Tao Core	4
Global Control Sigs	16
Video expansion	5
Control from CE	1
Total I/O count	184
Max I/O count (PQ240)	192
Percent margin	4.17%
Free I/Os	8

Table 2-3: Pin count budget for mezzanine and RE FPGA

Note that the actual number of I/Os available for most FPGA designs is around 180—the balance of 12 pins is usually required for providing clocks, configuration information, etc. In the case that an application requires more I/O pins, the High Address lines and the Parity lines can be reconfigured since not all applications or hardware implementations will require them.

2.6 Configuration Engine (CE) and Glue FPGA (Glue)

The Configuration Engine (CE) and the Glue FPGA (Glue) link the Tao processor to the host computer. As its name implies, the CE is responsible for configuring all of the computational FPGAs, initializing all LUTs, and setting up the BLR switchboxes. The Glue FPGA is responsible for translating S5933 PCI controller protocol to Tao protocols, and for directing high-level dataflow. The CE must be able to perform the following tasks:

- configuring RMU FPGAs
- configuring RMU PLLs, if necessary
- configuring RMU SLUTs
- configuring RE FPGAs
- configuring BLR switchboxes, if necessary
- partial and dynamic reconfiguration capability
- fast configuration
- cached configurations—local storage of configurations for fast recall

The requirements of the Glue are:

- Bus mastered DMA protocol support
    - FIFO management
    - PCI protocol management
    - Bus mastered DMA transfers have been clocked in at 120 MB/s
- Data concentration
    - Combining bidirectional data streams from two REs
- Protocol conversion
    - PCI to Tao pipeline
    - PCI to CE interface
    - PCI to GLOBAL bus
- Mailbox management

The Glue FPGA and the CE are intimately linked, and are thus discussed together in this section.

2.6.1 Glue Architecture

The Glue consists of a single high I/O count FPGA. Internally, the Glue FPGA consists of the following major components: concentrator datapath, PCI interface control, and CE interface/support logic. Figure 2-33 is a block diagram of the Glue architecture.

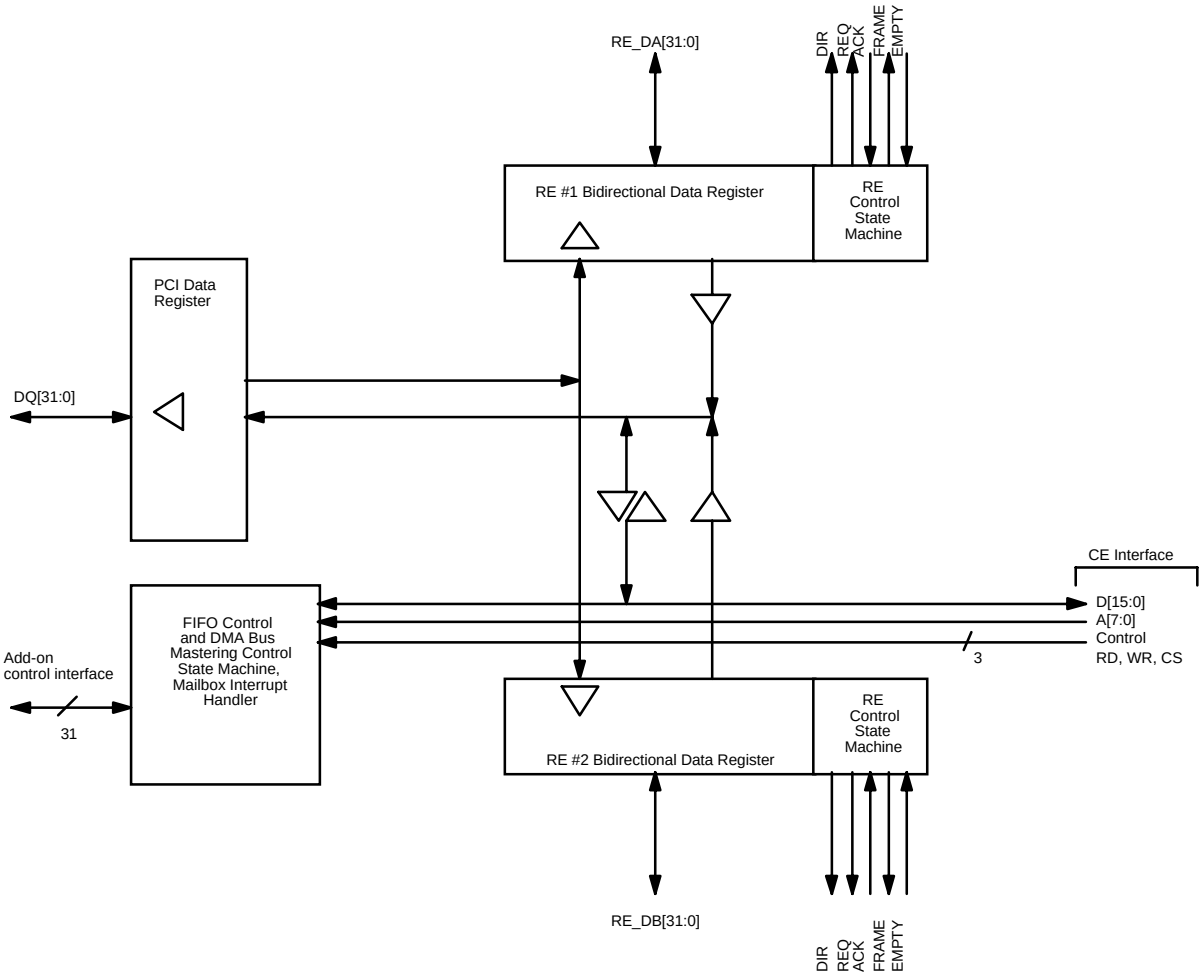


Figure 2-33: Glue FPGA internal block diagram. I/O count = 164.

A large part of the PCI interface control tasks occur via the CE control interface. This is because the PCI interface looks like a bank of addressable registers—mailboxes, control, and data all have addressable register assignments. The remaining signals—FIFO control and other fast-response signals—are controlled by internal state machines.

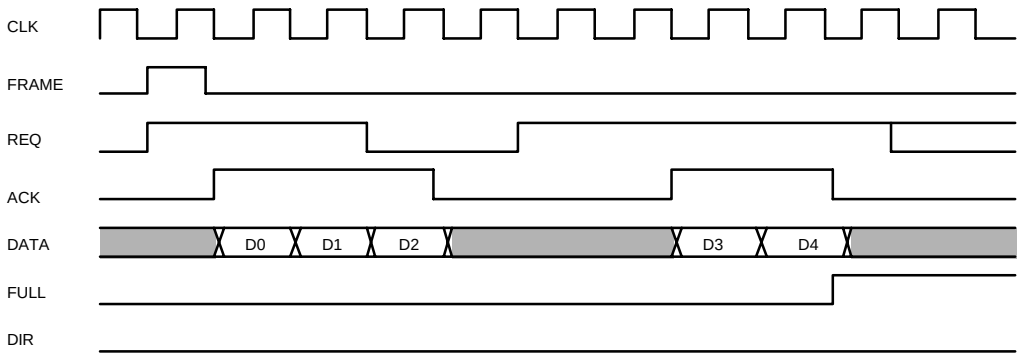


Figure 2-34: Glue to RE protocol. In this example, the Glue is requesting data from the RE, and the RE has only 5 bytes of data to give.

Figure 2-34 and Figure 2-35 illustrate the protocol between the RE and the Glue. This protocol is different from the standard Tao pipeline protocol because it has to support frequent stalls. In the RE-Glue protocol, the Glue is always the master, regardless of transaction direction, because the PCI Controller's shallow FIFO of 8 double-words depth is too small to allow sufficient time for arbitration.

FRAME is a framing signal; its assertion forces the RE into a "zero" state (usually all address counters pointing to the upper left hand corner of memory). DIR specifies the direction of the transaction. If DIR is low, then data is flowing from the RE to the Glue. If DIR is high, then data is flowing from the Glue to the RE.

When DIR is low, REQ indicates that the Glue is ready for data. The RE responds with ACK while simultaneously providing valid data. The RE is free to stall as long as ACK reflects a stall. When the RE runs out of data, it asserts FULL while deasserting ACK. The RE will stay in this state until FRAME is asserted. REQ is ignored while FULL is asserted.

When DIR is high, REQ indicates that the Glue is ready to send data. The RE responds with ACK, and on the next clock cycle, the Glue presents valid data to the RE. If the RE must deassert ACK for any reason, the Glue has one cycle to stop valid data flow. Hence, the RE must compensate for this single-cycle lag. When the RE is full, it asserts FULL while deasserting ACK, and REQ and data are ignored. They remain in this state until FRAME is asserted. Glue deasserts REQ one clock cycle before data becomes invalid.

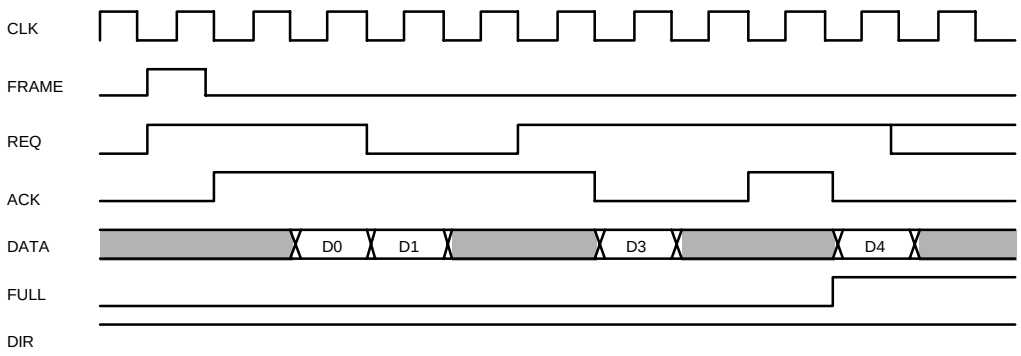


Figure 2-35: Glue to RE protocol. In this example, the Glue is sending data to the RE, and the RE can only accept 5 bytes.

2.6.2 CE Architecture

2.6.2.1 Overview

The CE uses an embedded RISC microcontroller to handle most of the configuration and control functions. The microcontroller, an SH7032 by Hitachi, provides sufficient speed and functionality to handle most operations independently, including PCI interface control and most of the FPGA configuration control. The presence of the microcontroller also opens the door to advanced FPGA configuration caching schemes. In addition, the microcontroller is a potent tool for debugging the PCI interface as well as the processor core since the microcontroller is accessed through a serial port and has access to all major signals.

The CE also has an FPGA which performs translation between the SH7032 protocol and the GPCB protocol. It also can perform up to six simultaneous parallel to serial conversions for the fast configuration of FPGAs. The GPCB and FPGA configuration busses take advantage of the SH7032's DMA feature to provide fast reconfiguration. Since the SH7032 provides a no-glue logic DRAM interface that supports burst-mode access, DRAM could be used for storage, but SRAM was chosen because of hairy ROM monitor interface issues.

Note that the SH7032 is directly responsible for programming the CE FPGA and the Glue FPGA.

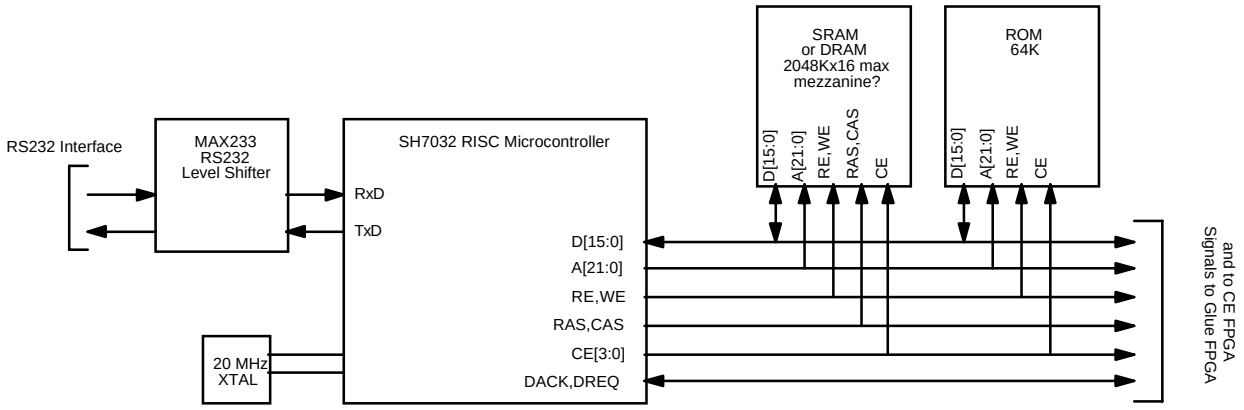


Figure 2-36: CE Microcontroller subsystem. The SH7032 is also responsible for programming the CE FPGA and the Glue FPGA.

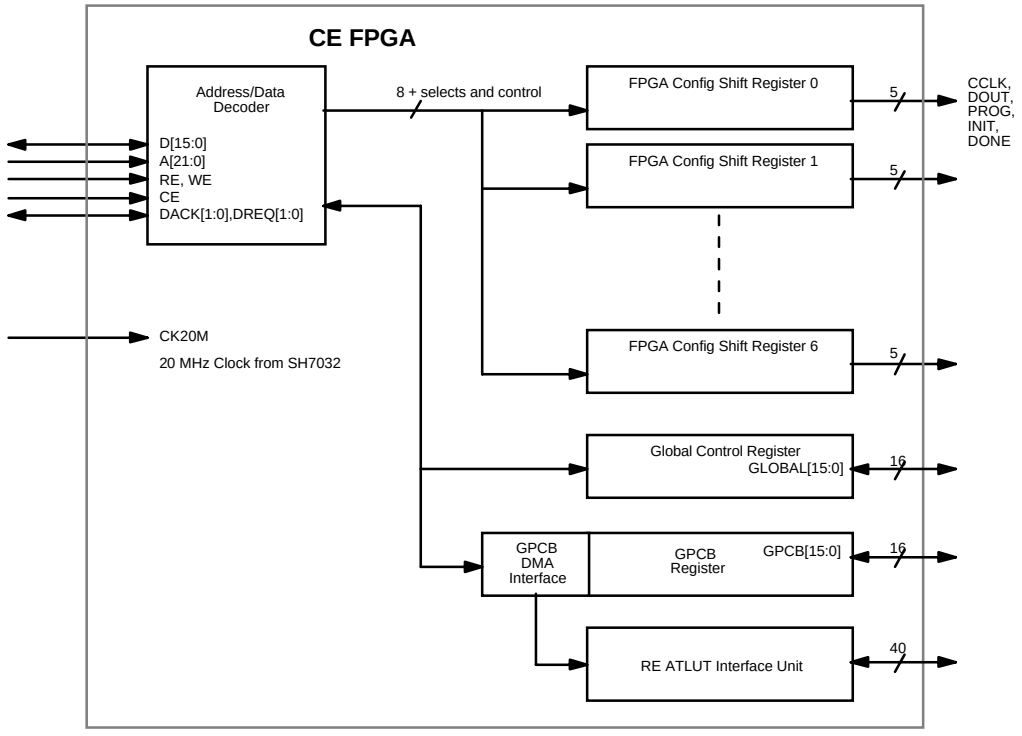


Figure 2-37: CE FPGA subsystem. I/O count = 147.

Because of the I/O count required by the CE FPGA, the CE FPGA is an XC4013E PQ240 package device. The CE FPGA needs to run at 20 MHz.

2.6.2.2 GPCB Protocol

Figure 2-38 illustrates the GPCB protocol timing.

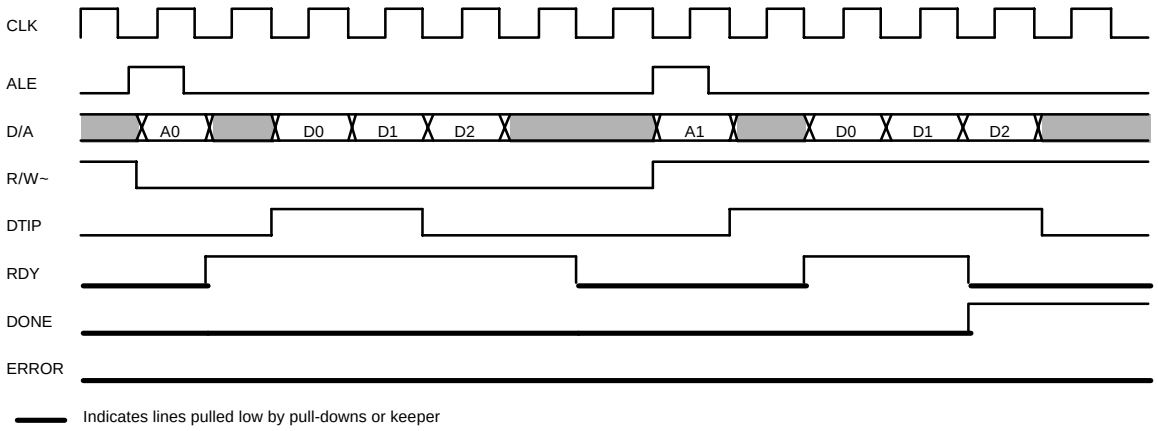


Figure 2-38: GPCB protocol timing. Some signals, such as RESET, are not depicted.

The GPCB bus protocol is strictly burst-based, with the CE being the only master. The following list defines the signals:

- ALE: Address latch enable
- D/A: Multiplexed data and address. 8-bit bus
- R/W~: Read/write signal. Read active high, write active low.
- DTIP: Data transfer in progress. Indicates CE is sending data or is ready to receive data.
- RDY: Tristate bus with keeper or weak pulldown. Indicates target device is ready to send or receive data.
- DONE: Tristate bus with keeper or weak pulldown. Indicates target device has exhausted all data, or is full.
- ERROR: Tristate bus with keeper or weak pulldown. Indicates an error has occurred in the target and that transaction should be restarted.
- RESET: Stops all transactions and brings all devices to a known zero state.

- C/R~: Config/run signal. Sets the mode of operation of the core. When in the config state, all data operations must cease and pipelines freeze.

After resetting the bus, all transactions begin with an ALE cycle. ALE indicates that a device address is available on the bus. Devices must latch the address and decode it. R/W~ is also stable at the time of ALE, and remains stable throughout the transaction.

If the transaction is a write transaction, the target device indicates readiness by asserting RDY. The CE will respond by asserting DTIP and commence the transfer of data. Once data has begun transfer, it cannot be stopped until either an error occurs, the device is full, or the CE wishes to abort the transfer.

If the transaction is a read transaction, the CE immediately asserts DTIP to indicate readiness to accept data. When the target device is ready, the target device asserts RDY and commences data transfer. Once again, the data transfer cannot be stopped until an error occurs, the device is empty, or the CE wishes to abort the transfer by deasserting DTIP.

Note that target devices are responsible for address generation. This incurs some internal overhead in the RMUs. Also note that since the address space is limited to 256 devices, each device, such as a SLUT on board an RMU, or a switchbox configuration register, has only one selector address, and all subsequent burst data is implicitly addressed.

2.7 Clocks

The master clock for the Tao platform is derived from the PCI bus 33 MHz clock. The AMCC S5933QB PCI controller chip provides a buffered version of the clock, called BPCLK. This signal is used as a reference clock for a Cypress CY7B991 “Robo-Clock” buffer. The CY7B991 is capable of frequency and phase locking to a reference clock, and provides doubled and half versions of the clock as well. The CY7B991 is capable of delivering tunable skew outputs, and provides 8 outputs.

Because the CY7B991 provides 8 clock outputs, each FPGA and/or RMU get its own dedicated clock line from the CY7B991. This provides some flexibility in clock termination schemes. Since the output duty cycle of the CY7B991 is guaranteed to be

50% ± 5%, AC end-termination can be used. [CypCY7B991] AC end-terminations are used on clock lines with daisy-chained loads, and series terminations are used for clock lines with a single load. An advantage of this clock-per-device distribution method is that rise-time variations due to loading is kept to a minimum. The allocation of the eight outputs of the CY7B991 are as follows:

- two outputs for the far RMUs
- two outputs for the near RMUs
- two outputs for the RE FPGAs' 33 MHz clock input
- two outputs for the RE FPGAs' 66 MHz clock input

Because the REs require a 66 MHz clock for double-rate half-width (66 MHz at 16 bits) bus support, and because there are potentially a large number of SSRAMs in the RE's buffer array, an additional CY7B991 is used to provide two switchable 33 MHz/66 MHz clock drives per RE mezzanine.

Onboard each RMU is yet another CY7B991 PLL clock generator. This local PLL is used to de-skew clocks and provide local half and double clocks if necessary.

Thus, the philosophy is to keep clock skew down to a minimum by using programmable skew buffers and local PLLs for clock management, and providing a clock-per-device distribution method. The total contribution to skew by device mismatch is kept below 1 ns; all remaining skew component is due to wire propagation delays, and most of that can be tuned away using the CY7B991's skew tuning feature.

2.8 Power Management

2.8.1 5V to 3.3V Conversion

Because the Tao uses 3.3V parts but is only guaranteed 5V off the card edge connector, it must provide its own 3.3V converter. A Power Trends PT6501 DC-DC integrated switching converter is used to accomplish this. This converter has a 14-pin SIP form factor, provides 3.3V @ 8A at 83% efficiency, and requires a single 330 μF external capacitor. Note that the Tao requires 6A at 3.3V.

2.8.2 Power Consumption Estimate

Table 2-4 summarizes Tao worst-case power consumption, which is roughly 47W. This is a problem because the PCI bus standard places the maximum power consumption specification of any PCI card at 25W, and footnotes it with a message indicating that most motherboards will probably provide only 10W. Note that 35W is probably a more likely average for the power consumption of a Tao card.

In order to circumvent a potential power problem, a pair of header connectors are provided on the Tao board for supplemental 5V and 3.3V power.

Description	Device	Package	Amps	Volts	Qty	Power (W)
PCI Controller	AMCC S5933QB	PQ160	0.200	5	1	1.00
FPGA	XC4013E-3PQ240C	PQ240	0.250	5	8	10.00
RISC Microcontroller	SH7032F20	FP112	0.130	5	1	0.65
RS232 Interface	MAX233CPP	DIP20	0.010	5	1	0.05
Serial EEPROM	XC24C16P	DIP8	0.005	5	1	0.03
SSRAM,128Kx18	MT58LC128K18D8LG-10	TQ100	0.250	3.3	24	19.80
32-bit quickswitch	QS32X245Q3	QS80	0.010	5	24	1.20
4-bit quickswitch	QS3125Q	QSOP16	0.010	5	24	1.20
roboclock	CY7B991-5JC	PLCC32	0.100	5	6	3.00
Power converter	PT6501	SIP14	83%	N/A	1	3.37
512Kx8 SRAM	TC518512FI-80	SOP32	0.300	5	4	6.00
64Kx8 FLASH	N28F512-120	PLCC32	0.100	5	1	0.50
				Total		46.80

Table 2-4: Power consumption estimate.

2.9 Debug

The Tao prototype is designed for easy debugging so as to provide a swift and painless board bring-up. This section discusses some of the features included on the prototype to aid debugging. Although it may seem odd to include a section on debugging features in a thesis, testability issues are extremely important from a practical standpoint and are too often overlooked and paid for dearly.

The Tao has a liberal helping of ground test points, roughly 1 per 2 sq. in. In addition, all clock lines have a test point near its termination. All mezzanine connectors

are male to promote easy probing. Since all the RMUs and the RE memory are on mezzanines, this makes a large number of signals readily available.

All key control signals, such as BLR switchbox configuration signals, pipeline flow control signals, GPCB, and GLOBAL signals, are made available on standard .1" spacing headers, in a format supported by HP logic analyzer pods. Some key SH7032 control signals are also be made available to help debug.

BLR switchbox signals can also be routed to LEDs for fast visual feedback on the status of the routing matrix. Four LEDs are available on each RMU for general purpose debug feedback. Power LEDs are also be provided, and each FPGA has an LED to indicate successful configuration.

The 3.3V power rail is routed to an SH7032 analog port so a 3.3V power failure can be automatically detected. Additional analog ports are brought to headers so that temperature sensors can be easily added to the board.

3. Discussion

3.1 Implementation

The Tao architecture discussed in section 2 was implemented on a double-height PCI card that can plug into any PC which supports the PCI bus. Appendix A contains the schematics for the Tao motherboard.

As it stands, the hardware is ready to host designs for real applications. Limited burst transfers over the PCI bus are currently supported, and full functionality of the on-board RISC microprocessor has been achieved. It is not the purpose of this thesis to discuss hardware details, nor is it to discuss user applications. Hence, the focus will remain on architecture issues and tradeoffs and the reader is referred to the appendix for more details on implementation issues.

3.2 Design Summary

The architecture described in this thesis fills the role of a general purpose, high performance platform for reconfigurable hardware experimentation. Figure 3-1 is a summary diagram of the devised architecture.

The processor core consists of four RMUs connected in a routing matrix that is topologically equivalent to a toroidal interconnect scheme. The interconnect switches are implemented using pass-gates. Pass-gates incur a 0.5 ns propagation delay due to the capacitance within the gates themselves; hence, the interconnect scheme is capable of distributing fast signals with low skew. Each RMU is a mezzanine daughtercard which can hold any kind of computational element. In this case, a single Xilinx 4013E FPGA with local SSRAM was implemented on each RMU. The inter-RMU signaling rate of the

processor core is 33 MHz, and each RMU has an on-board PLL that can provide a doubled (66 MHz) clock to the FPGA. Because the toroidal topology of the routing scheme has no edges, it looks the same from any RMU's point of view. This orthogonality allows users to design a single RMU that can be placed in any of the RMU slots. The toroidal topology is also extendible to larger RMU arrays with few additional wires. Hence, the Tao processor core has the infrastructure to support a high performance application and the flexibility to adapt as reconfigurable hardware technology progresses.

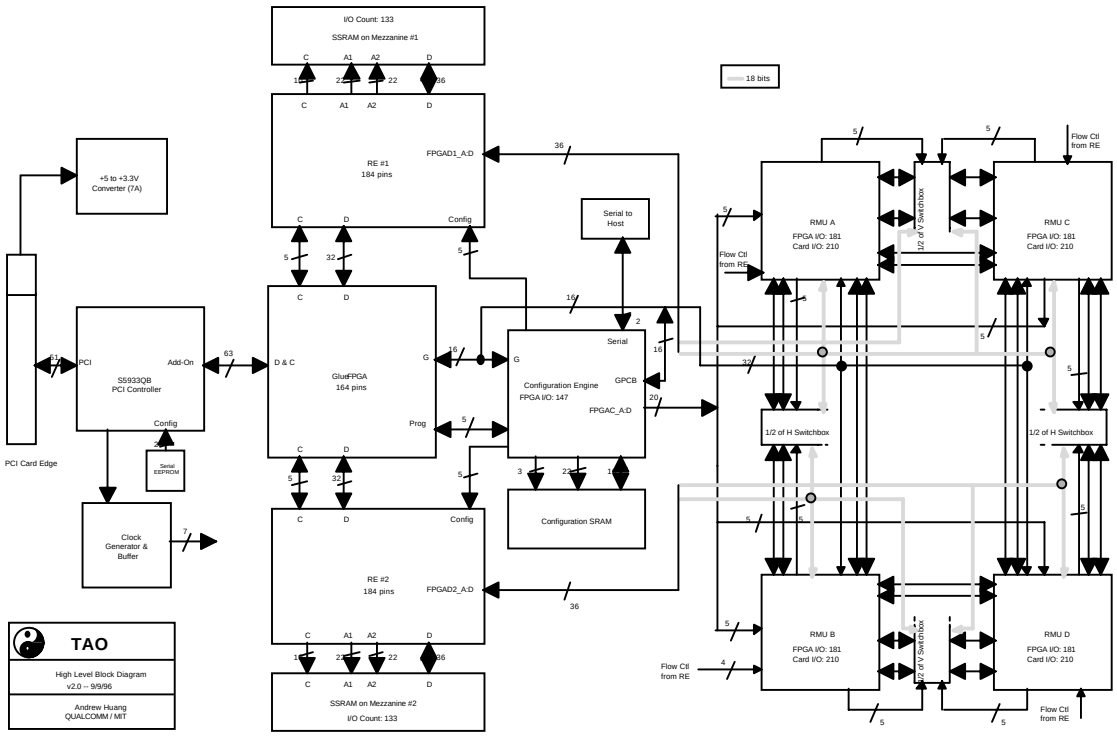


Figure 3-1: Architectural Summary of the Tao Platform

As previously noted, a high performance core is useless if it is starved for input data or if it is stalled writing out the results. High sustained bandwidth is guaranteed through the processor core by the two RE buffers that sit between the core and the peripheral interface. The peripheral interface is the PCI bus because of its relatively high peak bandwidth. The RE buffers are large enough to double-buffer least one quantum of data--in this case, a 2D image of 1024 x 512 x 8 bits--so that the processor core can continue processing even if it has to rearbitrate for PCI bus access. In addition to serving

as buffers, the REs play a critical role in formatting the data for the processor core; reconfigurable address generators in the control FPGAs within the REs can implement functions such as block-to-raster conversion, deinterlacing, interpolation, and decimation. Thus, a high aggregate throughput is possible with this architecture.

In addition to these performance features, the Tao platform sports an embedded microcontroller for managing system configuration. Since the architecture supports on-the-fly BLR reconfiguration and FPGA reconfiguration, one can cache RMU designs in the configuration engine SRAM and swap them in when necessary. Since RMU configuration can happen in a matter of milliseconds, users can implement real-time resource management schemes for implementing designs too complex to be loaded in all at once or even time-sharing schemes between multiple processes.

3.3 Future Directions

In retrospect, there are some architectural features that would be very interesting to try in future implementations of generalized reconfigurable hardware platforms. A large amount of effort went into developing a fast, thorough, and practical routing scheme. The task was difficult because there were so many wires—many routing topologies with desirable properties are too expensive or impractical to implement. To help ease the wiring requirements, it might be a good idea to use fewer, faster wires. In other words, instead of relying on a 9-bit bus running at 33 MHz, a 1-bit bus at 297 MHz would work just as well and require less space and fewer switches. There has recently been an explosion of "hot wire" technologies such as LVDS (Low Voltage Differential Signaling), fiberchannel, and SSA. All of these technologies are capable of achieving data rates in excess of 300 MBit/s. [Chi96] For example, Texas Instruments has the Flatlink™ series of LVDS data transmission products which transmit at bit rates of 455 MBits/s. Integrated PLL clock multipliers and shift registers make system design easier and more practical to implement. [TI96] By using serial LVDS technology, the number of wires running between RMUs could be cut down by an order of magnitude, thus making larger RMU arrays easier to implement despite the increased demands on wire and switch performance.

Another architectural feature that could enhance system performance is the incorporation of multiple high bandwidth I/O ports. The current architecture channels all I/O through the PCI bus. This represents a bottleneck since the core can support peak bidirectional stream rates in excess of 132 MB/s while the PCI bus supports peak *unidirectional* burst rates of 132 MB/s. Perhaps the incorporation of a high bandwidth HIPPI interface or a direct video I/O port via IEEE 1394 Firewire™, SSA or fiberchannel in addition to the PCI bus interface would alleviate this potential bottleneck.

3.4 Conclusion

The Tao reconfigurable hardware processor platform provides the necessary bandwidth and features to enable the implementation of demanding real-world applications with reconfigurable hardware. It accomplishes this goal through the use of a high bandwidth, low latency toroidal interconnection scheme between reconfigurable macrofunction units and two large, intelligent buffers between the processor core and the high bandwidth PCI I/O bus. The Tao platform has a modular architecture so that as reconfigurable hardware technology progresses, new modules can be fabricated and plugged into the current system. The platform also has an embedded microcontroller to enable sophisticated dynamic reconfiguration schemes.

This platform could be a valuable tool in many research areas, including but not limited to computer architecture and signal processing. The Tao platform is a good choice for architectural studies and benchmarking in high bandwidth applications, since that is what it was designed for. It is also capable of implementing video signal processing algorithms in real-time, thus offering signal processing experts a method of testing and tweaking algorithms against a large set of real-time video sources. This has great significance when testing algorithms for subjective performance in motion compensation because without a processor like Tao, researchers are limited to off-line simulations computed on GPPs. These simulations often take hours to compute even for relatively short video clips.

4. Appendix A — Schematics

5. References

[Act97] Actel web site. “Integrating ASIC Cores”. Web page last accessed May 17, 1997.
`http://www.actel.com/whatsnew/html/integrating_asic_cores.html`

[Alt23] “Digital Signal Processing in FLEX Devices”. Altera Product Information Bulletin #23. January 1996, ver. 1.
`ftp://ftp.altera.com/pub/ab_an/document/pib023.pdf`

[Alt95] “MAX 7000 Programmable Logic Device Family, March 1995, ver. 3”. Altera 1995 Data Book. Pp. 155-218.

[Alt123] “MAX 9000 Programmable Logic Device Family, March 1995, ver. 2”. On-line data sheet. Index 123.

[Alt126] “MAX 9000 Programmable Logic Device Family, March 1995, ver. 2”. On-line data sheet:
`ftp://ftp.altera.com/pub/dsheet/max9k.pdf.` Index 126.

[ATT95] “Optimized Reconfigurable Cell Array (ORCA) 2C Series Field-Programmable Gate Arrays”. AT&T Field Programmable Gate Arrays Data Book. April 1995. Pp. 2-5 : 2-310.

[Ber94] Bergmann, Neil W. Mudge, J. Craig. “An Analysis of FPGA-Based Custom Computers for DSP Applications.” Proceedings of the ICASSP 1994. IEEE Press. Vol. 2, Pp. II-513 - II-516.

[Ber96] Bergmann, Neil. Private email conversation with the author. June 28, 1996.

[Cas93] Casselman, Steve. “Virtual Computing and the Virtual Computer”. Paper presentation at the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM). April 5-7, 1993. Napa, California.

[Cas93a] Casselman, Steven. “Virtual Computing and the Virtual Computer”. IEEE Workshop on FPGAs for Custom Computing Machines, April 5-7, 1993. Napa, California.

[Cas94] Casselman, Steve. “Transformable Computers”. A paper presented at the 8th International Parallel Processing Symposium Sponsored by the IEEE Computer Society. April 26-29, 1994. Cancun, Mexico.

[Cha96] Chan, Sophia. "The Turing Machine". Web page. Link followed on July 10, 1996.
http://http1.brunel.ac.uk:8080/research/AI/alife/al-turin.htm

[Chi96] Child, Jeff. "Serial Buses Blaze Ahead of Parallel Solutions". Computer Design. August 1996, Vol. 35, No 9. Pg. 59.

[CypCYB7991] "Programmable Skew Clock Buffer (PSCB)". Cypress Databook, 1995. Pp. 10-130 : 10-140.

[Cyp96] "UltraLogic™ Very High Speed CMOS FPGAs". Programmable Logic Data Book 1996. Cypress Semiconductor Corporation. 1996. Pp. 4-1 : 4-5.

[Deb96] http://www.esat.kuleuven.ac.be/~debruyke/jpeg.html, page titled "A JPEG block diagram" found via AltaVista search.

[DeH94] DeHon, Andre. "DPGA-Coupled Microprocessors: Commodity ICs fort he Early 21st Century." Paper presented at the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM) 1994. April 10-13, Napa, CA.

[DeH95] DeHon, Andre. "A First Generation DPGA Implementation". Paper presented at the FPD '95--Third Canadian Workshop of Field-Programmable Devices. May 29-June 1, 1995, Montreal, Canada.

[Hal94] Halverson, Richard Jr. Lew, Art. "Programming with Functional Memory." Proceedings of the 1994 International Conference on Parallel Processing. Aug. 15-19, 1994. CRC Press, Ann Arbor.

[Hut95] Hutchings, Brad L. "DISC: The Dynamic Instruction Set Computer", in Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, John Schewel, Editor. Proc. SPIE2607, pp. 92-103. 1995.

[LSI97] LSI Logic Corporate Web page. Information can be found at the link to "Products".

[Moo96] "Moore's Law". Web page. Link followed on July 16, 1996.
http://www.intel.com/intel/museum/25anniv/html/hof/moore.htm

[New94] Newgard, Bruce and Goslin, Greg. "16-tap, 8-bit FIR Filter Applications Guide". Xilinx Application note. November 21, 1994. Version 1.01.

[Pat96] Patterson, David A. Hennessy, John L. Computer Architecture, A Quantitative Approach. Second Edition. Morgan Kaufmann Publishers. San Francisco. 1996. Pp. 53-54.

[Pra97] Pratt, Gill. "6.004 Spring 1997 Lecture Notes". Lecture 18 slide 11.

[QSIAN-11A] Quality Semiconductors Application Note AN-11A.

[Ros9-11] Rosenburg, Joel. "Implementing Cache Logic with FPGAs". Field Programmable Gate Array Application Note. Document from the internet. `http://www.atmel.com/atmel/acrobat/doc0461.pdf`.

[TI96] "SN65LVDS81 And SN75LVDS81 Flatlink™ Transmitters". Product preview datasheet. October 1996, revision 18.

[Vui] Vuillemin, J. Bertin, P, Roncin, D. Shand, M. Touati, H. Boucard P. "Programmable Active Memories: Reconfigurable Systems Come of Age".

[VuiA] Vuillemin, J. Bertin, P, Roncin, D. Shand, M. Touati, H. Boucard P. "Programmable Active Memories: Reconfigurable Systems Come of Age".

[Vui94] Vuillemin, Jean. "On Circuits and Numbers". Digital Paris Research Laboratory Report #25. November, 1993. France. Send email to `doc-server@prl.dec.com` with the subject line `help`.

[Vui96] Vuillemin, J. Bertin, P, Roncin, D. Shand, M. Touati, H. Boucard P. "Programmable Active Memories: Reconfigurable Systems Come of Age", IEEE Transactions on VLSI Systems, Vol. 4, No 1, 1996.

[Wil97] Wilson, Ron. OEM Magazine. "The riddle of the One-Chip System". Vol. 5, No 37. Pp. 37-46, p. 77.

[Xil2-10] "XC4000, XC4000A, XC4000H Logic Cell Array Families". Xilinx On-line Data Book: *XACT Step Development System Software CD, Version 5.2.0/6.0.0 for IBM PC or Compatibles*. Media DOS title: `XACT_STEP_951109`. Path: `\XACT\ONLINE\ONLINEDB\XC4000AH.PDF`. 2-10.

[Xil2-15] "XC4000, XC4000A, XC4000H Logic Cell Array Families". Xilinx On-line Data Book. 2-15.

[Xil4-14] "XC4000 Series Field Programmable Gate Arrays". The Programmable Logic Databook. 7/96. Xilinx, Incorporated. Pg. 4-14.