Comparing IP Integration Approaches for FPGA Implementation

Introduction

Since the early days of computers and telephony, interconnection networks have been a critical part of electrical engineering *(1)*. This has become even more critical in the era of very large-scale integration (VLSI) circuitry because of the drive characteristics of MOS transistors *(2)* combined with the relatively high capacitance of on-chip interconnects *(3)*.

The interconnection networks used to connect functional units within a chip can have a significant—indeed, a dominating—effect on the chip's performance. Buses, although the simplest form of interconnect, are a poor choice from a density or power standpoint because the power and space required to drive them at maximum speed grow exponentially with the capacitance of the bus *(4)*. Furthermore, multi-point connection networks are a poor choice because the entire length of the bus must be driven even when only a single conversation may be going on at a time, or where the communication is between direct neighbors. A crossbar is an optimal solution, up to a maximum size determined by the underlying device and wiring technology. In general, the optimal solution to multi-party communication is a network built out of crossbars *(5)*.

Status Quo

On-chip buses as seen today are a simple and straightforward outgrowth (shrinkage, actually) of system-wide buses used in computer systems. Although we have obvious proof that such buses are functional, they are just as obviously designed for the commercial and technical limitations of their day. That is, wires were cheap, circuits were expensive, and interconnect was faster than logic.

Today, none of that is true. Modern large-scale ICs are routinely speed-limited by their interconnect, not by their logic. The evidence is everywhere in chips with multiple clock domains and/or exotic wave-propagation techniques. Logic gates are now plentiful, with the proverbial (and generally misquoted) "Moore's Law" delivering more gate density than most engineers know what to do with. This embarrassment of riches has turned circuit and system design on its head: logic and interconnect are cheap and conserving wires is counter-productive.

Buses led to the development of bus standards—again, a natural outgrowth of the contemporary commercial and technical climate. Discrete ICs from different manufacturers needed to communicate over printed-circuit boards, so standards for logic levels, current drive, and signaling polarity were desirable. Standards for individual pins and signals (e.g., transistor-to-transistor logic, or TTL) led to standards for groups of pins, or buses. Signaling standards made it easy to connect unrelated parts, and bus standards made it easy to connect unrelated microprocessors, peripherals, and memories. These, in turn, led to board-level standards (e.g., VME, S-100, Futurebus, PCI, etc.) that treated entire circuit boards as plug-in modules. In many—in fact, most—cases these board-level standards were little more than extensions of vendors' proprietary chip-level pseudo-standards.

Although pin-level, chip-level, or board-level buses ensure electrical compatibility, they don't guarantee that the parties can actually talk to one another. The current is willing, but the protocol is weak, so to speak. Bus standards don't solve the big problems of system architecture and data flow, but they do a fine job of sweeping away the trivial problems, like exactly how to signal a successive string of ones and zeros.

Like many laws, policies, and regulations, bus standards have a way of outliving their usefulness. No sooner is a bus adopted than it starts to become obsolete. The very rigor the bus enforces becomes its weakness. Bus standards are intolerant of changes in signaling, protocol, bandwidth, or usage model. Buses are by nature slow to change and quick to stifle the unexpected. They deter innovation; they impede originality. Yet buses have a tenacious staying power; they're nothing if not consistent. They remain steadfast beacons of standardization in a sea of ever-present change and progress. Without fixed specifications, how would component makers adhere to the standard?

WP-01032-1.1

Alternatives Considered

The alternatives to buses are many, and all have been used successfully in various computers, chips, boards, ASICs, and FPGAs. These alternatives are no panacea, just as buses aren't a cure-all for every interconnection ailment. Avoiding the fixed routing and timetable of a standard bus can open up new avenues for design, and restore a bit of glamour and creativity to an otherwise mundane project.

Buses and Networks

The alternative to current bus architectures is merely a different kind of bus. To be precise, it's a different interconnect topology, such as a network, switch fabric, or crossbar. These interconnect topologies have been used successfully in many chip-, board-, and system-level products, and their popularity is growing. As chip- and system-level architectures change, it's logical that interconnect strategies should change with them.

A bus topology, like that shown in Figure 1, generally favors one-to-one or one-to-many communications. These buses were, again, outgrowths of the need to standardize pin- or board-level interfaces so that various vendors' chips could interoperate. Buses may have multiple masters (participants that originate a transaction and source or sink data), but only one master can be active at a time. Inherent in the definition of a bus is its exclusive nature. Only one master can use the bus at a time; all other potential masters must wait. Bus arbitration (i.e., the sharing mechanisms) thus becomes a significant part of any bus specification.

Figure 1. Traditional Bus Topology
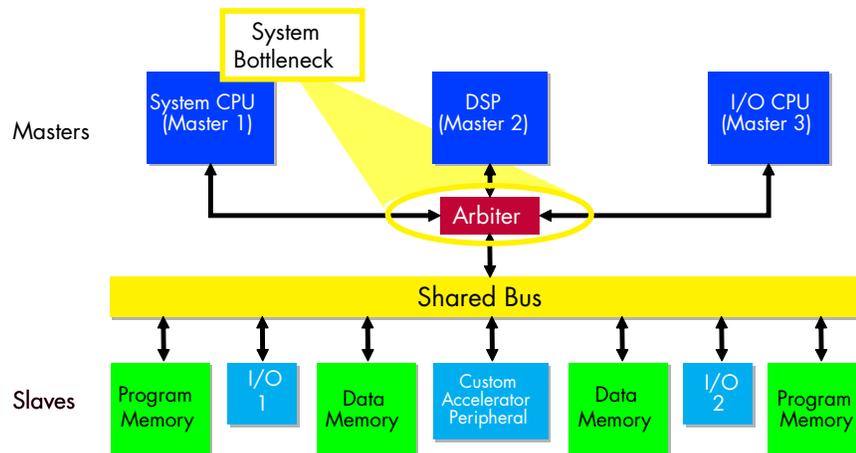
Most buses support multiple masters, although, again, only one master can be active at a time. The master competes for access to the bus, initiates a transaction, waits for the slave (or in the case of a "broadcall" transaction, multiple slaves) to respond, and then relinquishes the bus. The master may then initiate a second transaction or, through arbitration, lose control of the bus to another master. This arrangement can lead to system bottlenecks where the waiting for access slows the system performance significantly.

Somewhat more advanced buses support split transactions where the overhead of arbitration or transaction delays are mitigated by overlapping the beginning of the next transaction with the end of the previous one. Although these shave a few precious cycles from overall transaction times, they do nothing to alleviate the basic one-master problem that is inherent in all buses *(6)*.

In an ASIC or FPGA, chip-level buses are easily implemented using on-chip wiring resources. Standard chip fabrication techniques provide relatively long, straight metal layers on the top of the chip that are convenient for implementing buses (as well as for distributing power and global clock signals).

Network topology is quite similar to that of a traditional bus. Networks also are designed for one-to-one or one-to-many transactions over a shared medium. They also arbitrate control of the bus (often using

collision-detection and retry algorithms). In addition, they can slightly overlap adjoining transactions to save time. Apart from their physical embodiments, networks and buses are quite similar.

Switch Fabrics

Crossbar switches and their more generalized siblings, switch fabrics, are at once both simpler and more complex than standard buses. Crossbar switches provide a many-to-many communications mechanism for chips or systems with multiple masters and multiple slaves. Unlike a bus or network, crossbars and switch fabrics support multiple simultaneous transactions. This offers obvious improvements in bandwidth, except in the case where only one master wants to conduct a transaction at a time. In that case, a conventional bus or network would work just as well. In the more common case of multiple masters initiating transactions at unpredictable intervals (and often simultaneously), a switch fabric (shown in Figure 2) yields better results *(7)*.

Figure 2. Typical Switch Fabric Topology
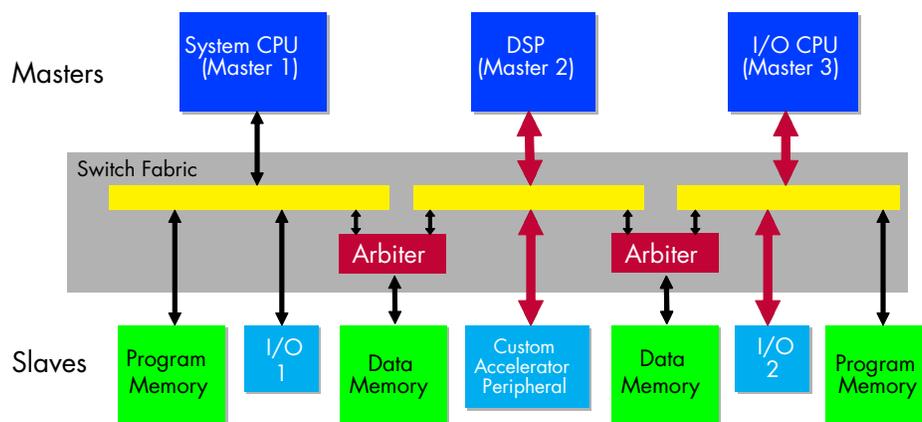
The multi-master scenario is quite common, even in systems with a single microprocessor or processor core. According to a Gartner/Dataquest study, the average number of processors in a system-on-chip (SoC) was about 3.5—and growing. In other words, most chips include more than one processor, where "processor" is defined as a RISC, CISC, video, or network processor that executes software.

Even standalone microprocessor chips often include more than one processor "core." Intel's well-known Core 2 Duo and similar processors famously include two or more heterogeneous processors within a single silicon chip. Freescale (née Motorola) has produced dual-processor QUICC and PowerQUICC communications chips for more than a decade. Fabless chip companies in the networking and communications markets routinely produce devices with four, ten, or dozens of processors in each chip. Not incidentally, these devices also use switch fabrics internally.

Crossbar switches are so named because they were once made literally from crossed metal bars placed at right angles to one another. Switches or relays connected orthogonal pairs of bars to create electrical connections. Because any bar in the X direction can connect to any bar in the Y direction, they are sometimes known as X-Y crossbars. Crossbar switches were (and still are) quite common in telecommunications equipment and are particularly valuable for their ability to make any-to-any connections.

Apart from increasing overall system bandwidth, switch fabrics also avoid many of the arbitration delays and overhead of buses. The bus is a single resource, whereas switch fabrics are shared. Any number of transactions may proceed simultaneously so long as two masters are not addressing the same slave (or vice versa). When resource conflicts occur, switch fabrics arbitrate like any other shared resource; barring any such conflicts, arbitration is unnecessary.

Switch fabrics thus provide both better bandwidth and lower latency. Memory latency is particularly important in many high-performance designs where processors are fetching code, storing data, or retrieving data and do not wish

to incur the time penalties inherent in a shared bus. Total bandwidth (that is, the number of simultaneous master/slave transactions) also improves when multiple masters can address multiple slaves at once.

Amdahl's Law says that memory bandwidth must increase as the number of processors increases *(8)*. Yet semiconductor memory bandwidth has not increased as rapidly as processors' appetite for that bandwidth. Memory is often the bottleneck preventing higher performance, and this mismatch has led to various workarounds, including multi-level caches, wide data paths, writable code stores, and different instruction sets. Opening the bottleneck to memory would seem to be every chip designer's first order of business. Squeezing all that traffic over a shared bus runs counter to that goal.

Implementing Interconnects

As with any interconnect, switch fabrics must be implemented with an eye toward the underlying silicon's strengths. Early crossbar switches were just that, crossed bars. Modern switch fabrics use semiconductor gates and metal routing layers. Compared to buses, switch fabrics rely less on long metal traces and more on logic gates. This makes them ill suited for printed-circuit boards but ideally suited for logic-rich chips like FPGAs. "Hard" ASICs fall somewhere in between, as they tend to be metal-rich but comparatively logic-poor, making them better candidates for buses unless specifically designed around a switch fabric, as many are.

In the logic-rich environment of an FPGA, switch fabrics make perfect sense. They play to FPGAs' strengths as well as to the larger trends in the industry. As designers move to "soft" hardware IP (intellectual property), switch fabrics are inherently softer in nature, working well with other soft IP that will likely be included in the design. Buses, in contrast, are tightly defined and carefully controlled. Switch fabrics are therefore easier to synthesize and easier to adapt to existing (and future) functional blocks.

Without a fixed bus interface, functional blocks or components can evolve more quickly and not be constrained by a bus definition. This is good news for the IP designer, IP user, and the IP itself. The switch fabric becomes an "IP integration backbone" instead of a bus; it connects various areas of the chip, but does so in a more flexible manner *(9)*.

Switch fabrics also leverage progress in EDA tools. They are a more abstract form of interconnection, defined at a higher level of abstraction than a Verilog bus model would provide, for example. The details of the connection are abstracted away while the designer concentrates on data flow and architecture, not bus timing and latency.

Raising the Level of Abstraction

Software developers have become accustomed to delegating the details of their craft to their tools. Instead of toggling ones and zeroes, they use instruction mnemonics, and then instead of mnemonics, they use high-level languages like C or Java. Instead of specifying exactly where every variable is stored and how it's aligned, they leave those details to the compiler. By delegating the details, programmers are able to concentrate on flow control, calculation, and logic while the compiler correctly implements all the variables, registers, arithmetic operations, storage, and so forth. C programmers don't even know whether a "word" is 16 or 64 bits—and it doesn't matter.

Good hardware-development tools, like good compilers, can map the designer's wishes onto the target hardware with a minimum of handholding. And like good compilers, they do it with an understanding of the underlying platform's structure and resources. By doing so, the tool accomplishes two things: it relieves the burden of detail from the designer and it produces a better mapping than the designer himself could have done. Over-specifying or over-constraining the implementation often leads to worse hardware (or software) because the compiler isn't allowed to do its job. Good compilers are not mere translators, they're optimizers.

In today's terms, bus specifications over-constrain the "solution space" that a hardware tool can provide. Freeing the tools to explore the more appropriate solution for the underlying silicon is not only easier, it's better. Particularly in the logic-rich environment of an FPGA, buses are a poor complement to the device's strengths. An interconnect might not ever be perfect, but it's always necessary. Whether it's made with a bus, a switch fabric, a network, or some other topology, it will be the foundation upon which the rest of the chip (and perhaps the rest of the system) is

designed. Done well, it will support the system's many goals. Done poorly (or inappropriately), designers will battle its limitations instead of doing productive work.

A Solution

Altera's SOPC Builder software offers a solution that addresses the need for designers to work at a higher level of abstraction by automating the integration of IP and design blocks. SOPC Builder provides a system design interconnect fabric that has the capability of integrating both the data plane and the control plane of a design, by supporting components with streaming and memory-mapped interfaces. The interconnect fabric supports slave-side arbitration enabling simultaneous multi-master bus access, which improves system performance over master-side arbitration schemes.

Further capabilities exist to manage topology and improve performance with bridges. These enable the designer or system architect to isolate system domains requiring more performance and throughput from domains with lower performance requirements. SOPC Builder's streaming interface provides a high-throughput, low-latency connection for high-bandwidth applications like packet processing, multiplexed streams, and DSP data. The memory-mapped system interconnect fabric uses minimal FPGA logic resources to support datapath multiplexing, address decoding, wait-state generation, peripheral address alignment (including support for native or dynamic-bus sizing alignments), and interrupt-priority assigning. Figure 3 shows an example of a system design using the System Interconnect Fabric.
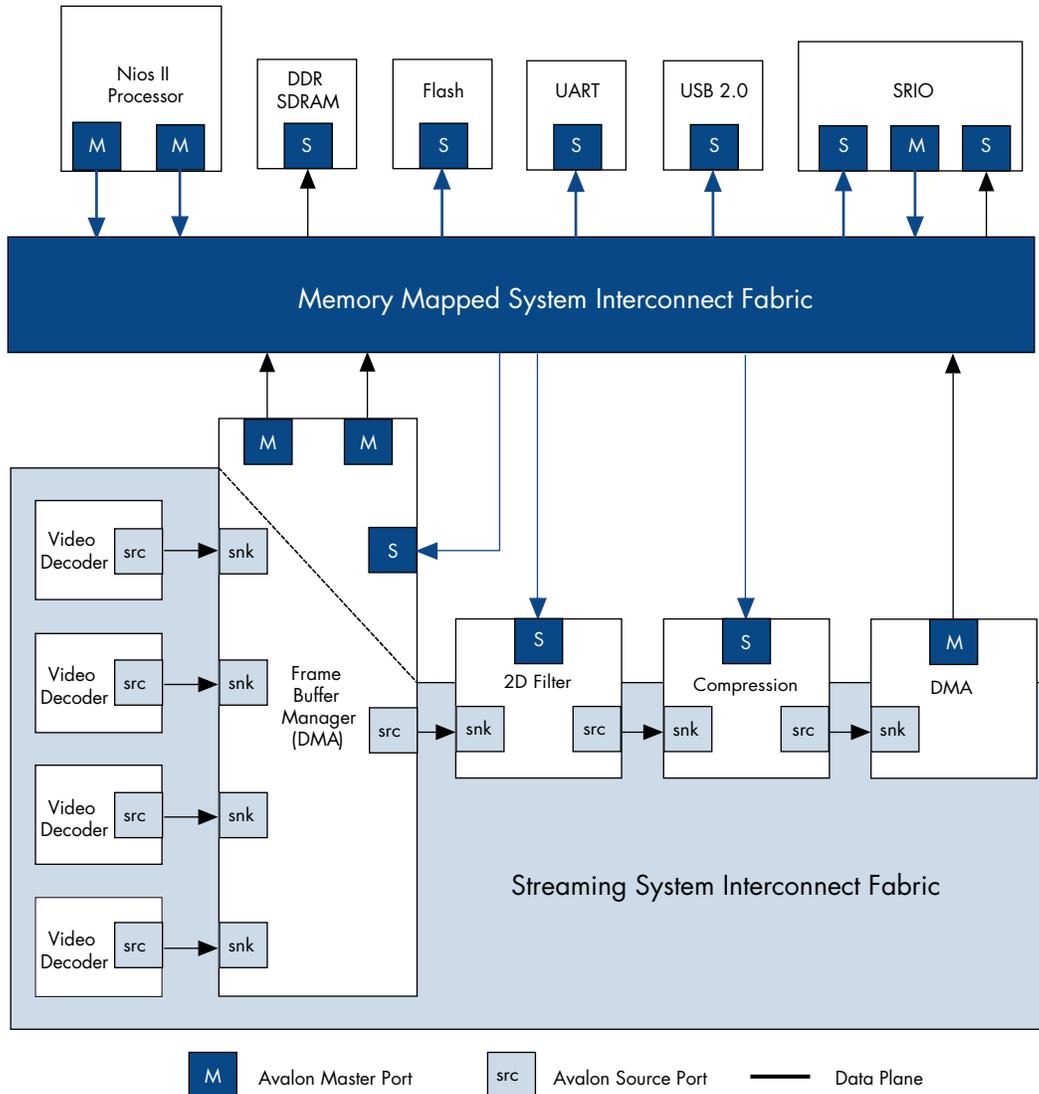
Figure 3. System Interconnect Fabric Example
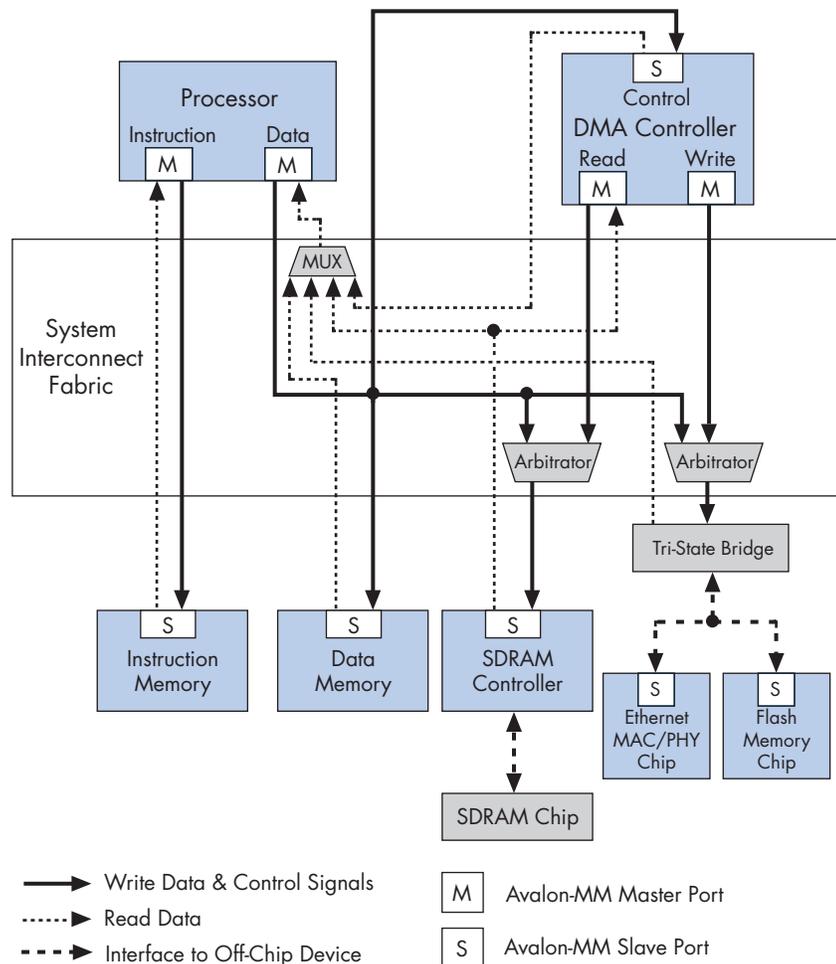
Figure 4 shows an example of a system with both the Memory-Mapped System Interconnect Fabric and the Streaming System Interconnect Fabric.

Figure 4. System Example Showing the Memory-Mapped and Streaming System Interconnect Fabrics

SOPC Builder can automatically generate a new optimized system interconnect fabric each time you add a new component or change the peripheral access priorities in your system. Because SOPC Builder automatically generates the system interconnect fabric, you can quickly and easily modify your system to improve performance or add capabilities. SOPC Builder is included with the subscription and web editions of Altera® Quartus® II design software.

Cited References

1. V.E. Benes, "Mathematical Theory of Connecting Networks and Telephone Traffic," Academic Press, 1965.
2. Foty, "MOSFET Modeling With Spice," Prentice Hall, 1996.
3. Sung-Mo Kang, "CMOS Digital Integrated Circuits," McGraw-Hill, 2003.
4. Johnson and Graham, "High Speed Digital Design: a Handbook of Black Magic," Prentice Hall, 1993.
5. A. Brinkmann, J.-C. Niemann, I. Hehemann, D. Langen, M. Porrmann, U. Rückert, "On-Chip Interconnects for Next-Generation System-on-Chips," in *Proceedings of the 15th Annual IEEE International ASIC/SoC Conference*, IEEE, 2002.
6. David J. Kuck, "The Structure of Computers and Computations, Volume 4," Wiley, 1978.
7. Brinkmann, et al.
8. Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *AFIPS Conference Proceedings*, pp. 483-485, 1967.
9. Weste, Eshraghian, "Principles of CMOS VLSI Design," Addision Wesley, 1992.

Further Information

- SOPC Builder:
  www.altera.com/products/software/products/sopc/sop-index.html
- Quartus II Handbook, Volume 4: SOPC Builder:
  www.altera.com/literature/quartus2/lit-qts-sopc.jsp

Acknowledgements