electronic design
THE Tool for Architecting the Unwired World
Wire Systems

[Wireless Networking]

FPGAs Build Bridges To Wireless Connectivity

Programmable logic can link the incompatible bus structures of embedded CPUs and wireless-networking chips.

Bernhard Andretzky
February 2005

The need for wireless connectivity in embedded systems is growing. In general, connectivity is becoming widespread as consumers and enterprises alike become accustomed to having continual access to information resources. As a result, information appliances like PDAs are adding capabilities for voice and data transmission. This connectivity allows more traditional embedded systems to report information to a central clearinghouse and obtain information from outside sources. In addition, systems like point-of-sale terminals and surveillance cameras become easier to implement when they offer wireless connectivity.

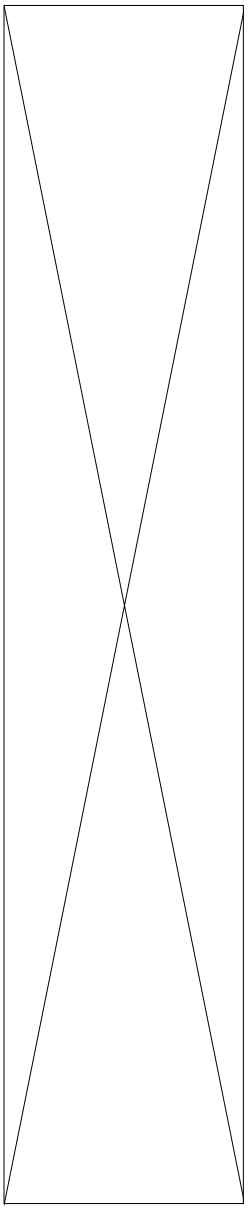
A wireless approach to connectivity is an essential ingredient for portable embedded systems, which by definition must operate without cables. When cabling is impractical or costs are high, however, wireless also has its benefits for stationary systems. A factory, for instance, may need printers on the production floor in areas where there are no walls along which cables can be run. Wireless connections avoid the need to provide below-the-floor channels for such installations.

Some embedded systems that need wireless connectivity may actually be incompatible with the devices that will provide that connectivity, however. That's because the bus structure of the embedded CPU often does not match the PCI interface found in most wireless-networking chips. Programmable logic can bridge the two buses. But the choices for bridge architecture and logic technology have a significant impact on the design's size, the software-development effort, and battery life.

A legion of choices exists for implementing wireless data connections. Designers can choose from several varieties of Ethernet (802.11), Bluetooth, cellular telephony, ZigBee, and other standards for implementing a wireless link. But bandwidth, software availability, and other considerations favor the use of Ethernet for many embedded-systems designs. Unfortunately, most of the chip sets for implementing the communications protocols and RF links for wireless Ethernet were developed for the personal-computer market. Because of this, many chip sets utilize the PCI bus for their host-processor connection.

Embedded systems, on the other hand, use a wide range of low- and mid-performance processors. Often, these processors don't offer a PCI-bus interface, which is required to interface to the common wireless chip sets. This situation leaves embedded-systems designers with a dilemma: The wireless chip sets that they need can't directly connect to the processors that they want to use. Unfortunately, cost considerations and the existence of legacy software make changing processors an undesirable option. The only alternative is for designers to implement logic that will bridge the CPU interface to the PCI-bus structure. Often, this task is

challenging. For one thing, the CPU bus may not have the same bus width as the PCI bus. As a result, the bridge will have to provide data formatting. Furthermore, the PCI bus doesn't allow for the simple memory mapping that's typically used in embedded systems. Rather, the PCI bus requires a complex protocol for the transfer of data. This requirement forces the bridge to provide data buffering along with the protocol logic. The CPU can then perform other tasks during the transfer.



Another constraint on the bridge design comes from power concerns—especially for portable systems. Most embedded designs now use active power management to keep systems cool and extend battery lifetime in portable systems. This approach requires the ability to shut down power to circuit blocks when they're not actively engaged in their function. Yet many embedded systems need connectivity only occasionally, such as when sending periodic reports to a central office. The wireless connection therefore needs a power-down option. Power-down requires shutting down both the wireless chip set and the bus-bridge logic. The bridge design must therefore provide a method for entering a standby or power-down mode.

Typically, cost is yet another constraint on the bridge design. Many embedded systems for the consumer market are under market pressure to minimize cost. Component cost and the costs associated with extra board space are critical elements. As a result, the bridge design needs to be as compact as possible. This implies a single-chip solution rather than discrete logic. Yet development cost also is a major concern in many embedded systems. Most designers simply can't afford to pay the mask charges or even spend the time needed to develop a custom bridge device.

A good solution that complies with all of these constraints is the use of a field-programmable gate array (FPGA) to serve as the bridge device. Typically, FPGA libraries of pre-defined functions include PCI interfaces. As a result, the bridge design is already half done. The flexibility of the FPGA allows developers to adapt the design to a variety of different CPUs. The design can therefore be used in multiple projects with minimal modification. Because the flexibility minimizes development time and eliminates mask charges, the FPGA design also is much more cost effective than a custom IC—especially in small production volumes.

For the designers who are seeking to utilize FPGAs, it's essential that they consider which base technology suits their needs. The FPGA market offers three base technologies: SRAM-based, EEPROM-based, and anti-fuse-based devices. SRAM-based devices use SRAM cells to hold the programmed values. Those values control the switch transistors, which make the logic-circuit connections within the array. Although EEPROM-based devices use the same basic approach, they employ nonvolatile EEPROM cells to hold the programmed values. Anti-fuse-based devices don't have memory cells or switch transistors. Instead, programming an anti-fuse-based device creates permanent logic-circuit connections.

Along with the FPGA base technology, designers need to consider the bridge architecture that they will require. A simple bridge connects the processor to the wireless chip set without other system interactions (FIG. 1). The FPGA's internal memory serves to buffer data passing to the network connection. But the processor must fill and empty the buffers under program control. The bus-mastering bridge includes a memory-controller block that allows the bridge to move data between the system memory and the network connection without processor intervention (FIG. 2).

Designers have many parameters to consider in evaluating the choices for bridge architecture and FPGA technology. System performance, including the network data rate and latency, may be important in some applications. System complexity, which affects the cost and design time required, is another factor. Along with the hardware effort, designers need to consider the impact of their choices on the software-development effort. Finally, the power demands and cost of the final configuration need to be evaluated.

When the embedded system needs the highest networking performance, the bus-master-bridge architecture is the optimum choice. A simple-bridge design has both data-rate and latency issues. It therefore requires the processor's intervention to move data across the bridge. The time required for the CPU to respond to a PCI

interrupt and retrieve the data directly adds to the latency of the wireless chip set in communicating over the network. This latency may not pose a problem for data-logging systems, in which communications are intermittent. But it can compromise the effectiveness of systems like point-of-sale kiosks. Such systems may need an interactive audio link between the customer and a remote salesperson.

The CPU's latency in handling data moving across the bridge also can impose a bandwidth limitation on the system. The size of the simple bridge's buffer memory together with the system's network bandwidth requirement set an upper bound on the time that's available for the CPU to respond before data is lost (FIG. 3). The larger the memory, the more time the CPU has available. Because the amount of memory available for buffers in the FPGA is limited, simple-bridge designs may require external memory to achieve the performance required by a system.

A better solution is a bus-master-bridge design, which has none of these issues. The FPGA logic can handle any data rate that the wireless chip set can provide. In fact, the speed of the system memory becomes the only limitation to the bus-master-bridge design's performance.

While performance demands favor the bus-master bridge, system-complexity concerns favor the simple bridge. Because the design requires only a basic PCI slave interface and a simple processor bus interface along with some buffer memory, it can be implemented in a modest-sized FPGA. The bus-master bridge needs both the CPU interface and a memory interface in the form of either an SRAM controller or a DMA channel. This requirement makes it too complex for the smallest FPGAs, requiring a medium-sized FPGA for its implementation. However, simple bridges also may require the use of external memory to meet performance requirements. This demand adds to system complexity.

The choice of FPGA technology also can affect system complexity. An SRAM-based FPGA needs to be configured each time it powers up. This need forces the use of an external boot ROM to supply the FPGA configuration data. EEPROM and anti-fuse FPGAs are nonvolatile, avoiding the need for this additional component.

Choosing the simple-bridge design to ease hardware complexity creates a problem with software development. The bus-master-bridge design provides a direct path to memory that matches the wireless chip set's intended installation in a PC. As a result, existing drivers and other software will run with minor modification on systems with a bus-master bridge. Systems that use a simple bridge, however, will need entirely new drivers to handle the use of buffering. Creating such drivers will require both time and expertise that the design team must provide. Wireless-chip-set manufacturers are hesitant to support the rewrite of their existing drivers for unique designs.

Perhaps surprisingly, FPGA technology choices also can affect software development. The SRAM-based FPGAs require time to load their program from the boot ROM before they are ready for operation. The system startup software that's supplied with the wireless chip set may not be able to handle the delay. This issue will cause a software failure during system initialization.

It's not a surprise that FPGA technology strongly affects the design's power requirements. The bridge architecture does have some effect on power. Although simple bridges use smaller FPGAs, they may need external memory. Bus-master bridges need a more complex FPGA. They also may need an external memory controller for memory-access arbitration. These bridge-architecture differences are relatively minor, however, compared to the effect of FPGA technology choice on system power.

In most cases where power is a concern, it's the system standby power that determines a design's marketability. The typical embedded applications that need wireless connectivity, such as a web-enabled smart phone, run off a battery. They need to provide between 200 and 400 hrs. of standby operation for consumer product acceptance. Calculations using typical battery capacities show that this standby time translates into a current of between 3 and 8 mA for system standby operation. To achieve such low standby currents, the FPGA technology choice is critical.

SRAM-based FPGAs have several significant drawbacks with regard to standby power. The first is that the designs require at least two devices—the FPGA and its boot ROM. The need for these devices increases the standby-power requirement over single-chip designs. The time required to initialize the FPGA upon power-up

also is a drawback, as it interferes with the system's ability to cycle power as part of active power management.

The most important drawback to the SRAM-based FPGA, however, is its inherently high standby and leakage currents. The use of an SRAM memory cell and switch transistor to control each connection point in the array adds up to a significant number of active elements in that array. These elements are above and

beyond the logic itself. In addition, devices with the necessary logic capacity for bridge designs are typically manufactured with deep-submicron semiconductor processes. Smaller processes have larger leakage currents, which add to the device's standby-power demand. A typical SRAM-based FPGA's standby current is 50 mA—far beyond the design limits for portable embedded systems.

An EEPROM-based FPGA avoids the need for a boot ROM, making it an improvement over the SRAM-based technology. Yet it contains the additional active circuit elements as well. It also uses deep-submicron technology to achieve logic density, creating a relatively high standby current. But its nonvolatile configurability does allow the use of active power management to reduce its average power demand.

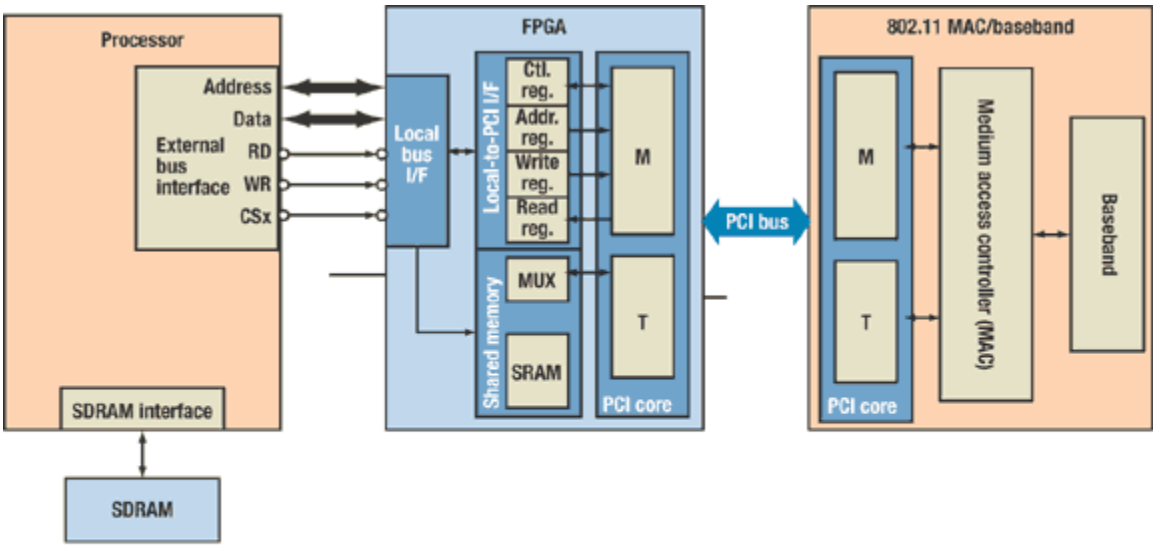
The least power-hungry FPGA technology uses anti-fuses for its configuration. The anti-fuse is a non-conducting link across connection points in the array. This link can be converted to a conducting link during device programming. Because the connections become hard-wired, the array doesn't need any switching transistors or configuration memory within the array. As a result, the anti-fuse FPGA only needs power for the logic array. This aspect reduces the device's standby current. Further, the lack of switching and configuration transistors means that the anti-fuse array is inherently smaller than memory-based arrays. It can therefore achieve the necessary logic density with a larger process geometry that has lower leakage current.

The compactness of the anti-fuse array affects the system cost for bridge designs both in terms of the cost of the component count and the board space needed. Compared to memory-based FPGAs, anti-fuse technology is able to achieve a higher logic-circuit density using a larger process geometry. This feature cuts component cost in two ways. First, the higher density produces a smaller die for a given level of complexity. Smaller die size translates directly into lower component cost.

The other savings comes with the cost of the fabrication technology. Deep-submicron processes are expensive to implement. Plus, mask costs for chips are high in comparison to older process technologies. The compact nature of anti-fuse arrays allows them to be implemented on older technologies with a reduced cost per unit area as well as with lower mask cost.

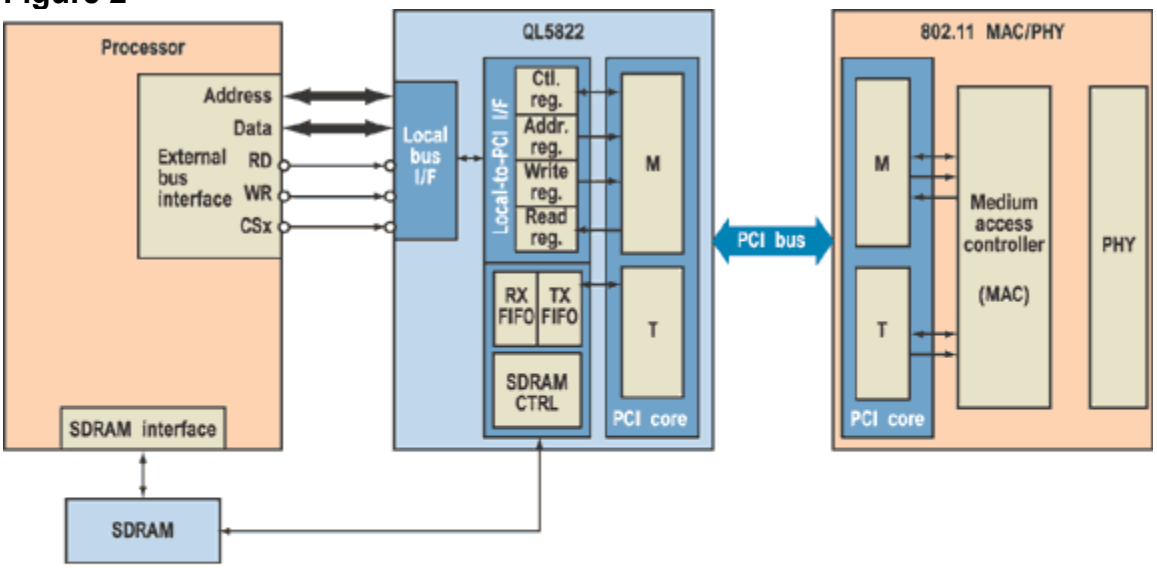
By considering such costs along with power, performance, and the complexity of both hardware and software development, designers will be able to choose the architecture and FPGA technology that's best suited to their project. The architectural choices allow tradeoffs among hardware complexity and cost, software-development effort, and performance. The technology choices favor the anti-fuse approach for its lower power and implementation cost. The right combination of technology and architecture will provide embedded-systems designers with wireless connectivity regardless of which CPU their system employs.

Figure 1



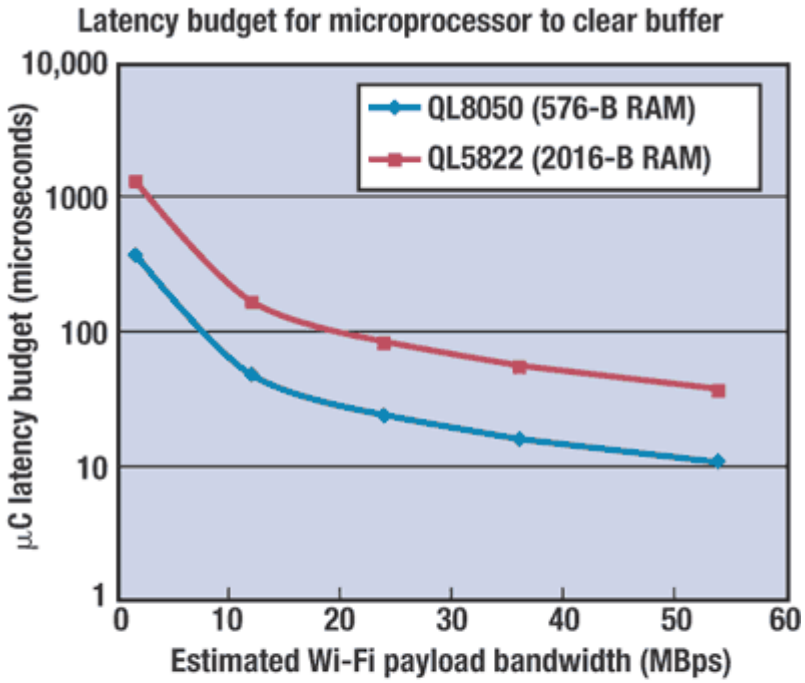
1. A simple-bridge architecture connects a single Peripheral Component Interconnect (PCI) device to the processor. It depends, however, on the processor to be involved in the data transfer.

Figure 2



2. To allow data transfers without processor involvement, a bus-mastering bridge architecture includes a link to system memory. A DMA controller can be used to push data into system memory. If DMA is not available in the system, an SDRAM controller also can be used.

Figure 3



3. The amount of time available for the CPU to empty the buffer in a simple-bridge design is a significant performance parameter that designers must consider. Using external RAM eases the time constraints, but at the cost of added hardware.