Make SoCs flexible with embedded FPGA

Geoff Tate, Flex Logix - November 30, 2016

Systems designers have long sought to provide programmability and flexibility in their systems designs to meet varying customer needs and evolving standards. The two most common approaches - FPGAs and MPUs/MCUs - provide different kinds of capabilities and complement each other, but have typically been separate devices. Now, chips with both processors and embedded FPGA are becoming a design option.

With the growth of connectivity, information, and data, there is a growing need for new processing capabilities that can span from ultra-low power <$1 microcontrollers to very large networking chips. Moore’s Law has given rise to the availability of new SoCs and MCUs for existing and new markets with each one of these SoCs/MCUs designed specifically for the market segment it is being targeted. This widespread use of special-purpose architectures greatly increases the need for new designs, however, and the rise of new markets (e.g., IoT) and device types (e.g., sensors) for these new markets is growing faster than the SoC types available.

System architects must also make tradeoffs between the latest technological opportunities with product, market, and application requirements - balancing features, performance, power, and cost. When budgets allow only a few cents for critical components, cost can be more important than performance or features. In other cases, the product’s features (e.g., quality of service) might mandate high performance. Power is also a big factor in computer system architecture, especially for mobile devices where battery life is important. And integration of functionality as a means of reducing the cost and power of the product while yielding higher performance by reducing on/off chip communication latencies is also desirable.

What designs need in cases like these is flexibility built into the system that can offer the system architect better options to tailor the design to the segment targeted. A reconfigurable block of RTL embedded in an SoC or MCU design that can be redefined as required for each application is one way of providing such flexibility. An FPGA is essentially an “RTL engine” at the chip level and thus, because of the reconfigurability and parallel compute nature of its fabric, an embedded FPGA meets the requirement. Such embedded FPGA is now available as IP in a wide range of sizes and configurations for chips.

FPGA Basics

Flexibility in the form of reconfigurable logic has long been available to system architects through the use of discrete FPGAs, which combine an array of programmable/reconfigurable logic blocks with a programmable interconnect fabric. The programmable logic block of an FPGA typically consists of several Look-Up-Tables (LUTs), which can be programmed to implement any Boolean logic function with a varying number of inputs (4, 5, or 6 typically.) The output of the LUT typically can be stored in a Flip Flop or sent out to the interconnect network. There is also typically logic in the logic block to implement the carry function to efficiently implement adders of any size.
The interconnect network connects the programmable logic blocks so that signals can be routed from any logic block to any other, as well as to/from surrounding IO. The interconnect network is programmable, as well, so the connections can be changed.

Each of the programmable logic elements and interconnects is controlled by a configuration bit that programs the LUTs, the interconnects, and other programmable aspects of the FPGA. To execute the RTL as desired in the FPGA, specialized design software takes Verilog or VHDL code and generates a “bitstream” which, when loaded into the FPGA, programs the configuration bits.

For more detailed information, here is a link to a basic FPGA tutorial on YouTube.

**Embedded versus discrete**

With discrete FPGAs, however, the outer rim of the chip provides a combination of GPIO, SERDES, and specialized PHYs such as DDR3/4. In advanced FPGAs, the IO ring is roughly 1/4 of the chip and the “fabric” is roughly 3/4 of the chip. The “fabric” itself is mostly interconnect in today’s FPGA chips, with 20-25% of the fabric area being programmable logic and 75-80% as programmable interconnect.

An embedded FPGA, on the other hand, is an FPGA fabric, but without the surrounding ring of GPIO, SERDES, and PHYs. Instead, the embedded FPGA can connect to the rest of the chip it is embedded into using standard digital signaling. This connectivity enables very wide, very fast, and very low-latency interconnects unobtainable with discrete FPGAs.

For example, a discrete FPGA connected to a CPU via a x16 PCIe gen3 bus gives a maximum full duplex bandwidth of 28.4GB/s. The bandwidth to a 28nm embedded FPGA with a 64byte interface (input and output combined) clocked at 500 MHz would yield 32GB/s. Further, the bandwidth into the embedded FPGA can be doubled from 32GB/s by doubling the data width, whereas the bandwidth into the discrete FPGA is limited to the x16 PCIe gen3 bandwidth.

Similarly, the latency of writing 256bytes of data into a discrete FPGA would be in sub-microseconds compared to only 16ns for an embedded FPGA (256B/32Bytes * 1/500MHz). The sub-microseconds of latency for the discrete FPGA is due to many factors such as the PCIe host controller forming the data packet and getting it ready to transmit, transmitting the data packet over the PCIe link, and the far end device receiving and processing the data packet and sending back an acknowledge. Transmitting the 256-byte data packet takes 18.1ns over the PCIe link and the other latency factors are multiple clock cycles of the PCIe controllers both on the sending and receiving side.
Another advantage of embedded FPGAs over discrete devices is lower power. The interface power for the discrete FPGA would be ~1.1 W for the x16 PCIe gen3 SERDES (0.55W for the CPU SERDES plus 0.55W for the discrete FPGA SERDES). The 0.55W is based on a 28nm SERDES at 4.3mW/Gbps. The embedded FPGA doesn’t have the 1.1W SERDES power adder or the additional signal pins required on the CPU to interface with the FPGA fabric.

**Implementing embedded FPGAs**

Implementing embedded FPGAs

An embedded FPGA can be provided either as soft IP or hard IP. Choosing to use hard IP provides an advantage, however. Because an FPGA is an array and not random logic, it lends itself to a much denser implementation (~4-6 times denser) if provided as a hard IP.

Hard IP also provides a validation opportunity. Because of the very high cost of chip designs today, chip designers will likely want to see the embedded FPGA IP proven in silicon before committing to integrate it into their SoC or MCU. If arrays are custom designed, each must be separately proven in silicon, which takes time and adds cost. But hard IP cores designed to operate standalone or be “tiled” or “arrayed” to build larger embedded FPGA arrays can eliminate the need for proving a custom design. IP that can be tiled enables the core to be implemented in silicon in, say, a 2x2 array, to prove out both the core and the core-to-core interconnect. Such proof also guarantees that any larger NxM array will work as well, freeing chip designers to implement such arrays with confidence - no further test required.

Embedded FPGAs have many other useful attributes, as well. An embedded FPGA may have options for MACs (multiplier-accumulator hardware accelerator blocks) for DSP from a little to a lot, for instance. Embedded FPGAs may also have embedded RAM, although different suppliers have quite different approaches to this. Discrete FPGA chips tend to have fixed amounts of DSP and RAM relative to programmable logic, whereas embedded FPGAs can be more customized to give chip designers only what their application needs. Embedded FPGAs also offer high IO count, especially relative to discrete FPGAs. Even a small embedded array has hundreds of IOs available while moderate sized arrays can have thousands of IOs.

Embedded FPGA arrays can be integrated into an SoC or MCU in three common ways:

For an application in an MCU where different customers have different needs for specialized RTL blocks (logic to full coprocessor) running as a slave on the processor bus, the embedded FPGA can directly connect to the bus. Any kind of processor bus will work: APB, AHB, AXI. Below is a simple example of a direct connection to an APB bus with the logic in the embedded FPGA itself (the logic
For an application such as protocol processing, the array can sit in the control and/or data path. In an application such as always-on processing in a battery-backed system, for instance, the embedded FPGA array can directly connect to and monitor/process data from a range of IOs.

And of course, a combination of the above functions is possible as well, such as an embedded FPGA block on a processor bus with direct IO connections.

**Configuring the embedded FPGA**

To program an embedded FPGA, Verilog or VHDL that describes the desired logic must be transformed into a bit stream that will set the configuration bits during chip boot-up or on-the-fly reconfiguration. First, the RTL is fed into Synopsys’ Synplify or Mentor’s Leonardo or some other synthesis program. The output of the synthesis step is an .edif file, which is the input to another tool that varies with the FPGA IP vendor. In the case of FlexLogix, that tool is the EFLX Compiler, which does packing, placing, routing, and worst-case path timing, then generates the bitstream file that programs the EFLX array to execute the RTL.

The bitstream configuration file in a system design is typically stored in Flash that also stores the code for the embedded processor. This may be on-chip Flash or use an external chip. In either case, the embedded FPGA IP comes with a configuration interface, allowing the SoC/MCU design to shift in bits from Flash memory to configure the programmable logic and the programmable interconnect. (The configuration interface often has a range of widths to allow the chip designer to enable faster loading if desired.) Once configuration is done, the embedded FPGA can execute the RTL that was programmed.

In use cases described above, the RTL for the embedded FPGA is defined by the SoC designer and is transparent to the system designer using the chip; the chip designer provides a packaged solution to the systems designer. Eventually for some applications, however, the system designer may wish to gain access to the array. In such cases the chip’s user rather than the chip designer becomes responsible for programming the SoC’s embedded FPGA, much as users do for the processor in an MCU today.
Xilinx and Altera, with their SoC FPGA/ARM combinations, have shown already that user-defined embedded FPGAs and embedded processors can get along on the same chip. However, there is a learning curve for customers to integrate their C programming for the processor with RTL programming for the array. Systems designers who see the value in embedded FPGAs may want to encourage their chip suppliers to provide the capability of configuring both through C programming alone.

**Conclusion**

System level reconfigurability provides many benefits:

**Customization** -- Embedded FPGA in an SoC/MCU can increase the variety of economical customizations possible at 40nm and below without incurring the NRE costs for customizing the SoC to make it suitable for specific market requirements. For example, an MCU can be designed with a reconfigurable RTL block connecting the GPIOs to the ARM bus and some hardwired SPIs/I2Cs/UARTs/etc. with the capability to “soft wire” additional serial IOs as needed in the reconfigurable RTL block. This allows the system designer to use a single SoC for a variety of system designs, each with different IOs specific to the product requirement needed for that system.

**Upgradability** -- Embedded FPGA in a SoC/MCU can be used to make upgrades to a system without redesign. For example, if the SoC/MCU uses the embedded FPGA to support protocol changes in the network controller, the system can be “soft upgraded” in a manner similar to firmware updates that are done in system.

**Innovation** -- Embedded FPGA allows for end-user innovation. If an embedded FPGA is used in an SoC as a peripheral to the AXI/AHB bus, the system designer has the capability to program the array to any unique function which offers differentiation for their product.

In all of these cases, systems are cheaper, more optimized, and can stay effective longer, thereby staving off early obsolescence.

Long considered the holy grail of chip design, embedded FPGAs are finally available to provide chip designers the flexibility to quickly, easily, and cost-effectively update or change RTL at any time after fabrication, even in-system. As every chip designer knows, having to change the RTL blocks at any point in the design process could easily cost multiple millions of dollars and add three-to-six months to the design schedule. With embedded FPGAs, this risk can now be eliminated, providing designers with a higher level of confidence when undertaking a new SoC/MCU project.

**Geoff Tate is the CEO of Flex Logix. Originally from Edmonton, Canada, Geoff earned his BSc, Computer Science, at the University of Alberta and an MBA from Harvard. He also worked toward an MSEE at Santa Clara University from 1979-1990. He has held a variety of lead technical positions, including Senior VP for Microprocessors and Logic at AMD and as founding CEO at Rambus. He has served on several high tech boards, including Everspin, the leading MRAM company.**
Also see:

- How FPGAs and multicore CPUs are changing embedded design
- FPGA IP cores enable flexible SoC design