

FPGA/DSP hybrid architectures: Satisfying the reconfigurability requirements of the military

By Ron Huizen

Many signal processing applications in the military require reconfigurability coupled with the speed and dynamic range of a floating-point DSP. The solution is a hybrid architecture that employs an onboard FPGA framework to create a seamless transition between I/O, FPGAs, and DSPs.

Embedded signal processing applications in the evolving modern-day military increasingly require flexibility, adaptability, and reprogrammability. The ability to modify a military system in real time based on what is happening in the real world – adapting to weather conditions, new threats, mission changes – is crucial. When this is coupled with the requirements of floating-point signal processing – low latency, high throughput, extended precision, and dynamic range – the ability to properly satisfy all requirements using one processing technology diminishes.

Many times, a design that includes both FPGAs and DSPs – a hybrid architecture – provides the best solution. The challenge then becomes one of integrating two quite different technologies efficiently and effectively so that the end system functions as one entity. A software control framework that can successfully merge the two technologies, allowing the system to play on the strengths of each, is a necessity.

Leveraging strengths, mitigating weaknesses

Embedded signal processing applications that require sustained, low latency, high-throughput data processing traditionally rely on DSPs. When additional dynamic range is needed, a floating-point DSP has been the processor of choice, also

ensuring that the system will support future, more advanced requirements. On the other hand, applications that require enough flexibility to be easily updated and modified usually rely on FPGAs, which, in recent years, have also gained the ability to provide embedded signal processing. When an application requires both the flexibility of an FPGA and the advanced signal processing capabilities of a floating-point DSP, the problem arises of which technology to use.

FPGAs are extremely effective for well-defined, straightforward, high-speed, repetitive problems requiring data flexibility and parallelism, the type of processing that is very often needed at the front end of the signal processing system. They can provide reconfigurable interfacing, with many of the newly released FPGAs able to implement switch fabrics such as Serial RapidIO and PCI Express, making them ideal for on- and off-board data transfer. They have been touted as do-everything silicon, able to perform high-end processing and provide the dynamic range required by many military signal processing applications.

In practice, though, difficulty in programming these devices for floating point can greatly increase time-to-market, making them unlikely candidates for high-end signal processing at any point in the near future. Another drawback of these devices is power inefficiency, a common trade-off with flexibility. A similar function implemented in an FPGA might take two to three times as much power versus a DSP. Note, however, this inefficiency may be an acceptable trade-off for the sake of flexibility.

Floating-point DSPs, on the other hand, are much better at implementing a broad range of highly complex algorithms that require floating-point math and have the tendency to change frequently. Any application that requires some sort of decision making or adaptive processing, where the next processing step is dependent upon the current result, requires a floating-point DSP. They are better in terms of power consumption and have the benefit of programming in C, decreasing development time. What they cannot do efficiently is handle the repetitive, straightforward processing that is many times required on the front end of the signal processing application; they can actually waste large amounts of processing efforts on these types of tasks, translating to a less efficient and more costly system. It's not uncommon for an FPGA implementation of a straightforward, parallel, repetitive task to require two to five times as many DSPs to complete the same task.

Given the strengths and weaknesses of each, it makes sense to use them as complementary technologies, mitigating the risks and weaknesses of each technology when used on its own. A hybrid architecture can intelligently combine the two, providing efficient performance, high throughput, and data control, all within a reasonable power budget. This type of system also has a higher amount

of inherent flexibility, providing reprogrammability for bug fixes and giving system designers the ability to support a different application in a very short amount of time.

The FPGA framework solution

While the benefits of hybrid design are many, along with these benefits come some significant design issues. A successful hybrid design needs to be able to properly allocate data bandwidth (including memory and I/O) among the FPGA(s) and DSP(s), easily connect the onboard data options while retaining the ability to modify them, and last but not least, successfully integrate the FPGA processing with the DSP so that each compute element is handling the part of the problem it is most proficient at. A software framework that can handle each of these is essential. The framework then functions as two separate entities: an I/O interfacing and routing device and a configurable FPGA pre-, post-, or co-processing engine. One example implementation is BittWare's integrated system framework ATLANTiS, shown in Figure 1.

Data interfacing and routing

Given that FPGAs are better at the repetitive, well-defined, straightforward processing – as well as being known for their flexibility – it makes sense that the

board's onboard entry and exit points reside on the FPGA, and that the entire software framework handling this data also resides on the FPGA. This FPGA framework can then handle all data transfer on and off the board and between itself and the DSP(s), acting as a software programmable cable and enabling the dynamic connection of every I/O to any other I/O, where connections can be created and broken without the need to recompile or change cables. The options for data interfacing can include: digital I/O (LVDS or single ended), general purpose I/O, flags, interrupts, link ports, high-speed serial links, rear panel connectors, off-board connectors, board-to-board links, cluster-to-cluster links, and multiport memories.

The ability to easily connect (and disconnect) the external data sources to each other, to IP processing modules within the FPGA, and to the onboard DSPs creates an incredibly flexible system that can be instantly modified for an immediate need, while also retaining the ability to be updated for future modification and expansion. This is achieved via the main framework switch, shown in Figure 2, which controls all connectivity and bandwidth allocation.

The switch is configured via registers by the host or any of the onboard DSPs and comprises two smaller switches, each with eight inputs and eight outputs. The switch connectivity is defined by the specific FPGA load and is controlled by the configuration registers, which allow source and destination time slices. This is all controlled by the system designer via a Windows navigator, command scripts, or C functions. Each switch accommodates single point, multipoint, or broadcast switching amongst any of its eight inputs at up to 125 MHz clock rate at 128 bits per I/O. The bandwidth can be allocated to any or all data inputs by 32 configuration registers that evenly divide the bandwidth slots into 62.5 MBps and can be changed "on the fly."

Configurable FPGA processing engine

Should the signal processing application in question require the FPGA to provide some of the signal processing, the FPGA framework also needs to enable the user to add FPGA processing blocks into the

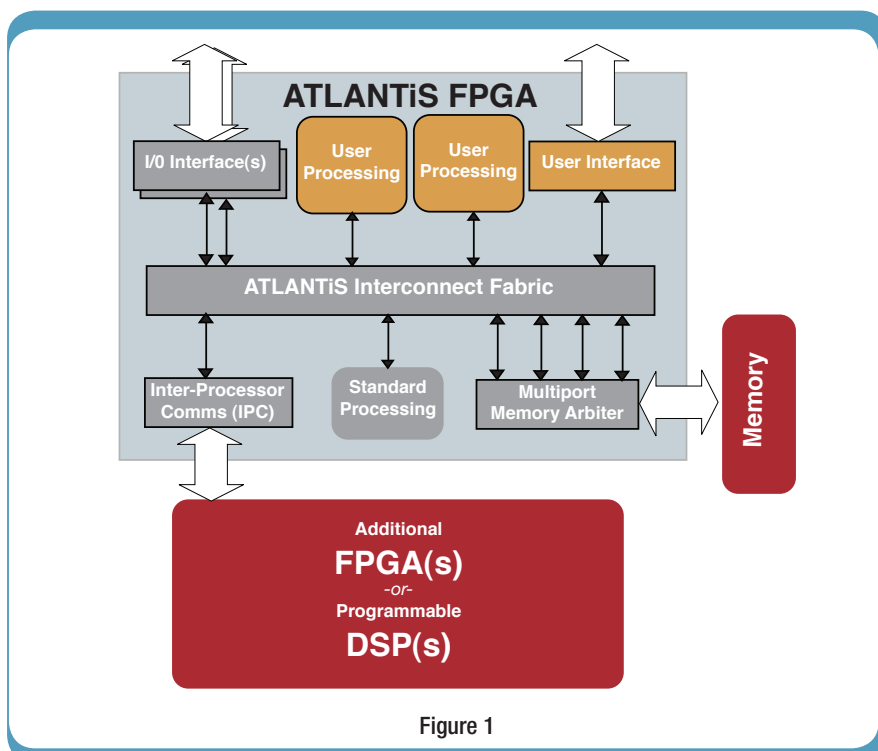


Figure 1

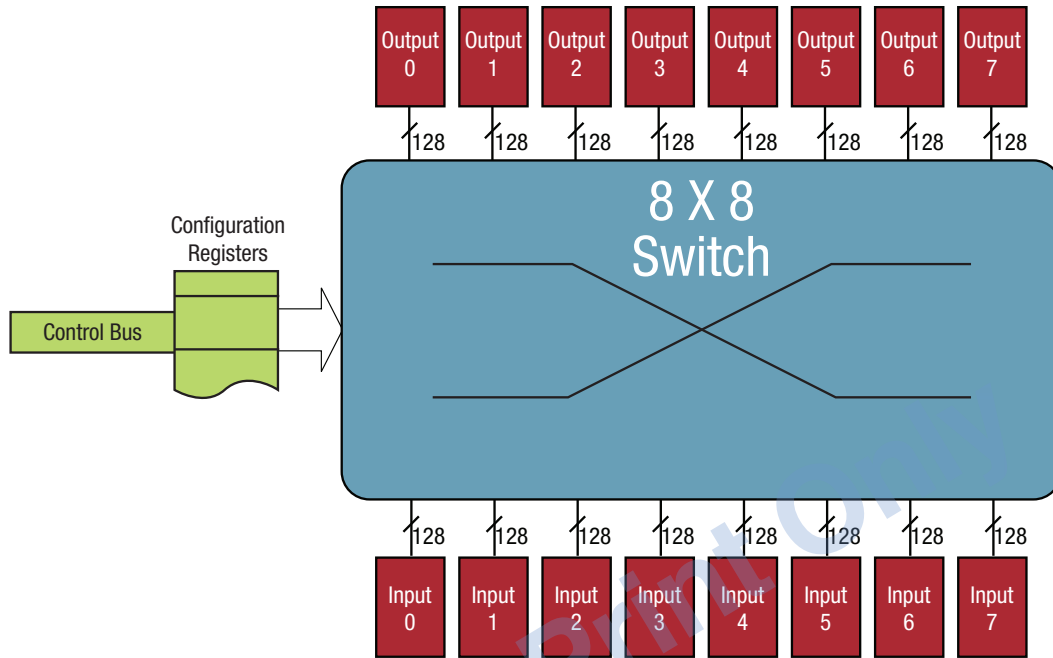


Figure 2

data flow at any time while the application is running. This can also be achieved through register-controlled data routing between all modules within the FPGA, enabling the insertion of FPGA processing modules at any point. As discussed in the previous section, the framework's memory space is accessible by the onboard DSPs or the host via either a peripheral bus or a cluster bus and is accessible to the designer via a GUI that provides a graphical representation of all possible source and destination I/Os. This enables DSP or host control and status, and R/W access to all of the frameworks resources; the most important resource is the main framework switch. Access to the memory space enables instant I/O routing changes and the ability to insert standard and/or custom FPGA processing blocks into the queue at any point during data transfer.

When the user configures the framework for a specific load, the master switch is programmed, defining source and destinations for each I/O from all available connections; the 32 configuration registers are memory mapped by a DSP or the host. Should a different load be required, the framework is reset and then configured with the next load. FPGA IP blocks can be

added into the switch to provide additional processing at any point during the data flow. The result is the efficient integration of two different technologies. With the FPGA framework handling all data routing, as well as providing any pre- or post-processing, the onboard DSPs are freed to handle the highly complex signal processing for which they have been designed.

A flexible, high-end signal processing system

Many of today's military signal processing applications require the flexibility and reprogrammability of an FPGA combined with the ease-of-use and floating-point processing provided by DSPs. The solution is a hybrid architecture that efficiently and effectively combines both by way of an FPGA framework. BittWare's GT-3U-cPCI, shown in Figure 3, is one example of this architecture. The board combines the Altera Stratix II GX FPGA

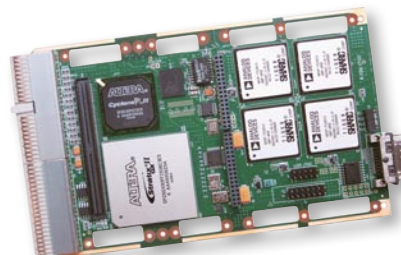


Figure 3

with Analog Devices TigerSHARC TS201 DSPs, creating an intelligent and flexible system with an easy path for future enhancements and upgrades.



Ron Huizen is VP of technology at BittWare, where he focuses on technology direction and new product concept definition. In a previous role

at BittWare, Ron oversaw all aspects of product development. Before joining BittWare, he held various roles in electronic product development at Amirix Systems. During his tenure at Amirix, Ron started Cabotel Systems, a spin-off company focused on developing semi-custom electronic products. Prior to Amirix, he worked at Nortel Networks for several years on SS7 switching systems. He holds a Bachelors of Computer Science from Acadia University and a Masters of Computer Science from Carleton University. He can be contacted at rhuizen@bittware.com.

BittWare, Inc.
603-226-0404
www.bittware.com