Across a wide spectrum of applications, signal processing algorithm complexity is exceeding the processing capabilities of standalone digital signal processors. In some of these applications, software developers have used hardware coprocessors to off-load a variety of algorithms including Viterbi decoding, Turbo encoding/decoding, butterfly processing, discrete cosine transforms (DCT), and 1D and 2D filters. In a few cases, DSP processors include on-chip hardware coprocessors where the end application supports the expense of designing such a market-specific solution.

In third-generation wireless systems, the addition of the Turbo, forward-error-correction algorithm had a huge impact on the amount of processing required per user data channel in a channel element card. Texas Instruments has successfully conquered this challenge of utilizing coprocessors for Turbo and Viterbi processing.

Unfortunately, the high cost of implementation makes the availability of DSPs with end-market-specific coprocessors unattainable. In applications where no coprocessors are available, design tools and methodologies that enable companies to develop their own coprocessors using the latest FPGAs that easily interface with a wide range of DSP and general purpose processors (GPP), and provide increased system performance and lower system costs are now available.

This article discusses the technical development and integration of FPGA coprocessors, including:

- Profiling applications to identify high-load software algorithms suitable for off-loading to coprocessors
- Development of custom coprocessor blocks
- Viable-coprocessor system architectures
- Processor interface selection
- Hardware and software system integration
- FPGA-coprocessor development systems
- Cost and performance improvement attainable with FPGA coprocessors
In addition, this article provides a design example for implementing an FPGA coprocessor for a TI DSP to increase performance and lower the cost of a modem system, and will highlight the methodology and application of FPGA coprocessors. This example assumes that software initially implemented the target system, therefore having no foresight into any optimal hardware/software partitioning.

Identifying software that can be off-loaded to a coprocessor

Often times in DSP processing applications, 20 percent of the program code consumes 80 percent of the required MIPS. This 20 percent of the program code often requires time-consuming, error-prone, and difficult-to-maintain assembly coding to increase overall system performance. This code also becomes far less portable than the remaining 80 percent of the code that focuses on initialization and system execution control. At the same time, that 80 percent of the code reflects the majority of the system’s complexity. This utilization outcome creates a double challenge for DSP software engineers. They must reduce the processing load in 20 percent of the software and manage the complexity of the remaining 80 percent of the code.

FPGA coprocessing is well suited to addressing the 80 percent processing load caused by 20 percent of the algorithm code. The challenge is to identify what the DSP should off-load to a coprocessor.

The key to identifying what the DSP should off-load to a coprocessor is using the profiling tools that the software developer used. Profiling tools parse the program code and identify the percentage of processing that each function and subroutine consumes. Every software development system includes tools for profiling the program code and for identifying which functions consume the majority of the processing MIPS.

Code profiling can help identify the functions that consume the majority of the MIPS and can provide options, for example, such as accelerating the code by use of a hardware coprocessor. Not all functions are appropriate for off-loading to a coprocessor. First, the goal is to identify a group of algorithms that together occupy more than half of the processing load. Second, the identified group of algorithms should be clustered together so that once data reaches the coprocessor, there is no processor dependency in the calculation until the processing is complete and the coprocessor can return the result to the DSP. A third criterion is that the processing be straightforward to implement in hardware. A simple definition is that the algorithm be heavily looped, thus implying a very repetitive computational structure.

The system example in this article uses a TI processor, although the applied principles are applicable to all DSP processors. A product called Code Composer Studio (CCS) encapsulates the TI development tools. CCS includes a debugger, compiler, linker, assembler, code profiler, and other assorted capabilities to enable a software developer to fully describe and develop the DSP program code in one environment.

The system utilizes one of the application examples, modem.c, that comes with the TI development kit, specifically the TI6x series of development systems. Modem.c implements a Quadrature Amplitude Modulation (QAM) modem. When the application compiles and executes modem.c on the TI development system, it takes 177,000 instruction cycles to execute.

CCS profiled the modem.c example to identify what could be off-loaded to an FPGA coprocessor. The analysis identified that the modem transmitter algorithm, modem tx, required the majority of the processing. The modem tx consumed 96.5 percent of the processing MIPS (see Figure 1). The modem tx can also off-load to a single FPGA coprocessor that implements the modem tx data flow. The contents of the modem tx include:

- Shaping filter using 82 percent MIPS
- Modulation using 8 percent MIPS
- Sine lookup using 2.5 percent MIPS
- Cosine lookup using 3.5 percent MIPS

Coprocessors, as defined by Altera, include at least a data interface and a control interface. The CPU uses the control interface(s) to set up and monitor the operation of the coprocessors. The data interface(s) communicate to memories, peripherals, or other coprocessors, as both sources and sinks of data. To maximize system performance,
engineers define each data interface to include integrated direct memory access (DMA) controllers. The CPU programs these DMA controllers through the control interface of the coprocessor. In general, the CPU sets up the operation of a coprocessor, and then the coprocessor autonomously executes the CPU.

Many powerful capabilities are inherent in this architecture that yield extremely high performance systems. The first of these is that the CPU can set up the coprocessors to automatically source and sink data without the coprocessors having any dynamic interaction with the CPU. The flexibility of DMA programming along with architectural selections of the FPGA coprocessing system enables this capability. A linked list of source or destination addresses that automatically enable the coprocessors to continuously execute without CPU interaction can control the DMAs. These source and destination locations can be memories to which the CPU or some other coprocessors source or sink the data. The source and destination locations could also be peripherals such as Universal Asynchronous Receivers/Transceivers (UARTs), Administrative Domains (ADs), or Desktop Accessories (DAs).

The overall architectural flexibility of FPGA coprocessors enables system definition such as tight coupling to the master CPU, or loose coupling to a data processing plane that has only minimal setup and status interaction with the master CPU. This wide variation in capabilities makes FPGA coprocessors suitable for dealing with systems having a wide range of performance and flexibility requirements.

The coprocessing block identified in the modem.c example requires an integration of a Finite Impulse Response (FIR) filter, a modulator, and two look-up tables. In this case, designers used Altera’s DSP Builder, an add-on tool to the Mathworks MATLAB and Simulink tool set, to assemble the design. See Figure 2 for an illustration of the modem coprocessor as captured in the DSP Builder processor interface selection.

When an FPGA coprocessor connects to a separate DSP or GPP, there must be an interface between the DSP and the FPGA coprocessing subsystem. This interface is dependent on the interface specifications of the target processor. Most processors support a variety of standard and proprietary interfaces. The standard interfaces today and for the future include PCI and its permutations, RapidIO, Hypertransport, and others. There are also many proprietary interfaces available including EMIF (TI), MPX (Motorola), and Link-Port (ADI) among others. For any processor that links to an FPGA coprocessing system, an FPGA interface IP block must be available.
The application characteristics as well as the available interfaces on the processor will drive the interface selection between the processor and the FPGA. For example, the TI c6x DSPs support several different interfaces. The alternative interfaces include the 16/32/64-bit Extended Memory Interface (EMIF), the 16/32-bit Host-Port Interface (HPI), 32-bit/33 MHz PCI interface, and the Multichannel Buffered Serial Ports (McBSPs). The configuration of these interfaces is different across available devices, and in some cases, the specific features of the interface are device-specific.

“The application characteristics as well as the available interfaces on the processor will drive the interface selection between the processor and the FPGA.”

The system example uses the EMIF because it is common to all of the c6x devices, with some minor variations in features and number of bits, and provides high performance (≥ 100 MHz). EMIF has a variety of permutations including support for 16-, 32-, or 64-bit transfers and asynchronous and synchronous signaling. This example uses asynchronous signaling on the 32-bit interface.

**FPGA coprocessor architecture**

When the DSP or GPP processor communicates with the coprocessor, the efficiency of data movement often becomes the dominant factor in the overall system performance. Today, high performance DSP processors rely on DMA controllers to minimize CPU overhead when communicating outside of the CPU core and its memory cache. Typically, the CPU core will access cache memory as the primary memory in the core DSP algorithms. The DMA engine moves data into and out of the cache memory.

When interfacing to a coprocessor, whether it is on-chip or on an adjacent FPGA, the coprocessor must interface with the cache memory via the DMA controller, thus off-loading the CPU core to continue processing other tasks. On the FPGA side, it is also advantageous to include a memory buffer to act as a local cache to the coprocessors. In this way, the DMA control on the CPU side is simply moving data from memory to memory, and the CPU and coprocessors maintain a stronger independence.

The modem example shown in Figure 3 uses an FPGA coprocessor as defined in DSP Builder.

Coprocessors by their very nature change the software implementation from an algorithmic description to a data passing and function control description. The new function call initializes the coprocessor and controls the flow of data to and from the coprocessor. This interaction requires that hardware-specific information be available to the software engineer that includes address information for controlling the coprocessor as well as source and destination address information. It also requires a description of the control structure of the coprocessor. The engineer can pre-configure these capabilities as software drivers for controlling the FPGA coprocessing data flow.
Tools are available to integrate FPGA coprocessing blocks into subsystems that directly interface to standard processors. For example, Altera’s SOPC Builder can support a variety of IP types, including coprocessors. Associated with each IP block is a predefined set of software routines used to configure and control that IP block. Within SOPC Builder, users identify which blocks to assemble and how to parameterize and interconnect them. SOPC Builder then automatically generates the hardware architecture as well as a software driver file called Excalibur.h. Excalibur.h includes all the software interfaces for the blocks in the system and automatically eliminates their references to the register and memory map as defined by the user’s architectural selections. See Figure 4 for an illustration of SOPC Builder hardware and software integration flow.

SOPC Builder can work with coprocessors having both a parameterized-hardware architecture definition and a full set of software routines to configure, communicate, and generate status information. When a designer uses SOPC Builder to assemble a coprocessing system, SOPC Builder not only generates the hardware architecture, but it also assembles the software routines into the Excalibur.h.

SOPC Builder can also support external processors by implementing the targeted processor’s interface logic as an IP core that interfaces into the SOPC Builder Avalon bus. In addition, this feature is applicable to all of the interfaces discussed previously.

The example modem system utilizes SOPC Builder to integrate the DSP Builder transmit-data-flow coprocessor with the TI EMIF interface. When SOPC Builder executes, it creates the hardware for the FPGA-based coprocessor and the Excalibur.h software to control the coprocessor from the attached CPU. The Excalibur.h file includes the address for all registers and memories inside the SOPC Builder as well as associated software APIs for IP blocks that include APIs. This correct-by-construction file accelerates system integration by months, by eliminating error prone and tedious manual development of low-level software drivers. In addition, once the blocks integrate into SOPC Builder, they become easily reusable.

The development system that enables this kind of integration must have both a processor and an FPGA adjacent to each other with the appropriate connections, such that the FPGA can integrate with the available processor buses. A designer can integrate these development systems onto a single board or onto two or more development boards, each hosting a subset of the complete system components.

For example, the Altera DSP Development Kit Stratix Edition includes a standard TI daughtercard connector, allowing a direct connection to most of the TI development systems including the standard kits for the c6x family of processors.

The modem.c example required 155,000 cycles to compute an iteration of the modem functionality. When designers added the FPGA coprocessor to the system architecture, the total clock cycles dropped to 455 clock cycles. The modem coprocessor consumes 6209 LEs, or about half of a low-cost Cyclone EP1C12 device. Off-loading the modem to a coprocessor enables an increase in
channels, functionality, performance, and a significant cost reduction as well, by using a less expensive variant of the processor.

Integration benefits
With the advent of DSP-optimized high-performance FPGAs, it is clear that FPGA coprocessing provides a powerful approach to increasing system performance and reducing costs. These benefits are available without changing the software development environment or the DSP platform, except for the addition of a low-cost adjunct FPGA. In applications that are forced toward leading edge DSPs for performance reasons, this approach can reduce costs by ten times.

This integration approach also provides a handy way to future proof a system when future performance requirements might increase the processing performance demanded on a board. Engineers can do this by designing an empty FPGA socket onto the production boards. The board then will not utilize the FPGA socket until future evolutions of the system demand increased processing performance. Through straight-forward software revisions and the inclusion of one or more FPGA coprocessors, system performance can dramatically increase with minimal component cost increases to the system.

Paul Ekas is senior DSP marketing manager for Altera Corporation. He joined Altera in August of 2002, and has more than 17 years of business experience in electronic design automation and complex semi-conductor systems. Before joining Altera, Paul was a director of product marketing at Morphics Technology where he was responsible for the 3G WCDMA infrastructure product line. He has also held sales and engineering positions at Mentor Graphics, Silicon Designs, and Seattle Silicon. Paul holds a B.S.E.E. and M.S.E.E. from the University of Washington.

Brian Jentz joined Altera in October 2000. He is responsible for new product definition and marketing of DSP Products. Before joining Altera, Brian spent seven years at Texas Instruments, most recently as a DSP product specialist. Brian holds a B.S.E.E. from Purdue University and has completed course work toward an M.S.E.E. at Georgia Tech.

Altera Corporation is a pioneer of system-on-a-programmable-chip (SOPC) solutions. With annual revenues in CY 2002 of $711 million, Altera combines the programmable logic technology originally invented in 1983 with software tools, intellectual property, and design services to provide high-value programmable solutions to customers worldwide. Committed to helping customers achieve their business goals, Altera has continued to aggressively invest in research and development efforts.

Contact Altera directly for further information.