Design Methodology for Hardware Acceleration for DSP

Sheac Yee Lim
Altera Corporation
101 Innovation Dr
San Jose, CA 95111
(408) 544-7000
sylim@altera.com

Paul Ekas
Altera Corporation
101 Innovation Dr
San Jose, CA 95111
(408) 544-7000
pekas@altera.com

ABSTRACT

In many signal processing systems, designers are faced with the challenge of requiring more processing power to compute complex digital signal processing (DSP) functions than the processor in use is able to provide. A new methodology from Altera Corporation has been developed that enables application of co-processors without requiring new application specific digital signal processors. Altera's methodology can be applied to any processor through the use of field-programmable gate arrays (FPGAs) acting as co-processor platforms for user defined co-processors.

This paper will describe an architectural approach for co-processors that integrate smoothly into a standard DSP oriented software development flow. The approach is applicable to FPGAs with embedded soft-processors or a combination of off-the-shelf processors plus an external FPGA.

We will demonstrate this methodology through an implementation of an FPGA with an embedded soft processor and a (finite impulse response) FIR filter co-processor. The methodology includes software profiling to identify which functions should be implemented as co-processors and what their performance specifications should be. A novel methodology for implementing co-processors will be described. Finally, a hardware/software integration methodology enabling a complete co-processor development and implementation solution will be described.

This methodology utilizes standard off-the-shelf tools and components and is applicable to many applications where signal processing performance requirements are driving up the cost and risk of system implementation.

1. INTRODUCTION

The use of signal processing for applications such as 3G wireless and digital video/image processing has grown significantly in the last several years. This growth has resulted in the need for more signal processing ability for improved speed, faster throughput and increased channelization, commonly offered by special purpose application-specific integrated circuits (ASICs). However, factors like fast time-to-market and high silicon fabrication costs tend to favor other alternatives such as the use of DSP processors with application specific hardware co-processors designed to perform dedicated, computationally-intensive signal processing functions. An example of this is the C6416 DSP processor by Texas Instruments with dedicated on-chip Turbo and Viterbi co-processor hardware, primarily targeted at 3G wireless applications [1]].

While there are advantages to using DSP processors instead of custom ASICs from both a developmental cost and flexibility standpoint, the fact remains that co-processors offered by DSP processors, while fast, are largely fixed in terms of function and features. The decision to include particular co-processors in a product offering is primarily determined by demand and maturity of the target end-application as well. Semiconductor vendors find it difficult to justify development and fabrication costs for co-processors targeted at anything but mature telecommunication and video/imaging applications. In emerging applications, designers are forced to utilize either suboptimal performance on a standalone DSP processor or the development of costly dedicated ASIC solutions for their specialized signal processing functions. Either one of these solutions could pose a myriad of design risks including multiple design cycles to meet design performance specifications, increased board and silicon fabrication costs as well as complex system integration and verification.

Altera has developed a new methodology that enables the application of co-processors to off-the-shelf DSP-based systems without requiring the development of new application-specific DSP processor devices. Altera's novel approach employs FPGAs as platforms for developing custom defined co-processors. These co-processors can be specifically designed to suit any end application and can be combined with either external off-the-shelf processors or an internal embedded processor implemented within the same FPGA device. Using provided, pre-defined APIs, device drivers and real-time operating systems (RTOSs), this methodology closely integrates and eases hardware and software development, making the methodology more appealing for designers accustomed to existing DSP processor design environments.

2. CODE PROFILING AND HARDWARE ACCELERATION USING CO-PROCESSORS

Code profiling involves obtaining the execution time of each function within the user's software code and analyzing each function to get an idea of where the processor is spending most of its time. Core algorithms that consume the majority of the processor's CPU time can be identified after which methods of enhancing the performance of the system are investigated. The

example C-code shown in Figure 1 illustrates a function that has been designed to run entirely on a single processor.

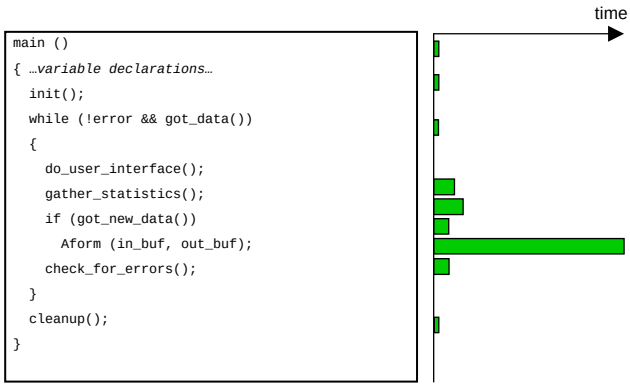

Figure 1. Example C-code and code profile of an arbitrary DSP function called Aform

Profiling of the code illustrates a key bottleneck where we find a function called Aform[1] taking up a large portion of the processor's entire CPU time. This bottleneck poses a significant performance inhibitor which can be easily overcome with the use of a custom Aform co-processor to offload the CPU hogging function. What we consider to be the "heavy lifting" required by this specialized signal processing task is now offloaded to the co-processor while the majority of the CPU's processing capabilities are utilized for general purpose control functions.

The performance improvement afforded by offloading this computationally intensive task to a co-processor can be very significant (Figure 2), especially if the function that is causing the bottleneck is utilized repeatedly within the user's application.

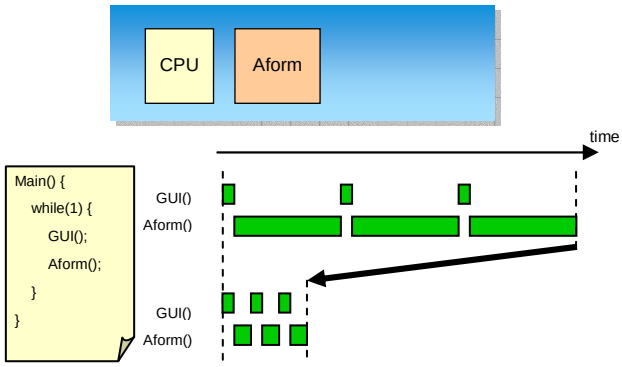

Figure 2. Performance improvement achieved by implementing custom hardware co-processors for "heavy lifting"

3. PARALLELISM IN A SEQUENTIAL ENVIRONMENT

[1] The function Aform is used in this context to represent some arbitrary DSP function like a transform or a filter.

One of the key advantages of using co-processors to perform hardware acceleration is the employment of parallelism, where multiple tasks can be executed simultaneously, providing additional performance improvements. In Figure 3, we see how task execution could occur with the addition of other co-processors in the system. These tasks are running in a sequential nature and do not take advantage of parallel execution. Hardware functions that are not data dependent could be executed in parallel, resulting in an overall drop in hardware utilization and improved computation time and performance. The execution of these functions is determined by the user's C/C++-like language however, which does not map to parallel processes because it executes functions sequentially, in the order that they are called within the code.

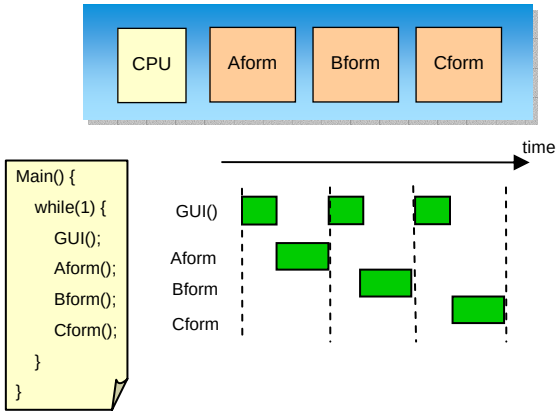

Figure 3. Implementing multiple co-processor hardware blocks

How do we provide parallel acceleration in a sequential environment? One solution enabling software code to perform multiple tasks at once involves the use of threads. The term thread or thread of control denotes a section of code executed independently of other threads or threads of control within a single program [2]]. While threads, executed on a single CPU are still executed sequentially and thus do not affect the actual parallelism of the execution, threads used in conjunction with co-processors provides the ability for the processor and all co-processors to operate in parallel, affording the performance increase described earlier in this paper.

Modifying the example such that it calls the various functions in threads enables the parallelism illustrated in Figure 4. The amount of parallelism and utilization afforded by the use of threads is governed by the critical path of the dataflow through the various functions implemented in within the design.

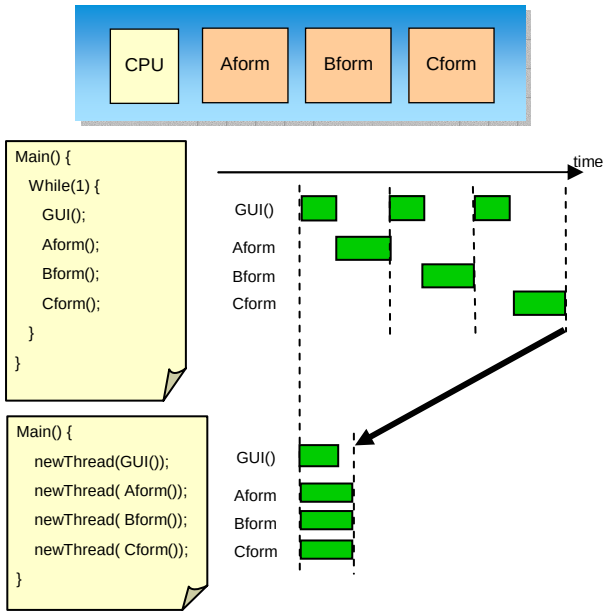

Figure 4. Parallelism provided by the use of threads

4. THE HARDWARE INTERFACE

So far, we have discussed the software portion of this design methodology. On the hardware portion, we are proposing the interface illustrated in Figure 5. Here, the co-processing function could be a piece of custom logic or pre-configured off-the-shelf IP like a FIR filter, a fast Fourier transform (FFT), equalizer or discrete cosine transform (DCT) sandwiched between two similar dynamic memory access (DMA) peripherals. A DMA peripheral allows for DMA data transfers between two memories, between a memory and a peripheral, or between two peripherals and is used in conjunction with streaming-capable peripherals, allowing data transfers to occur without intervention from the CPU [2]].

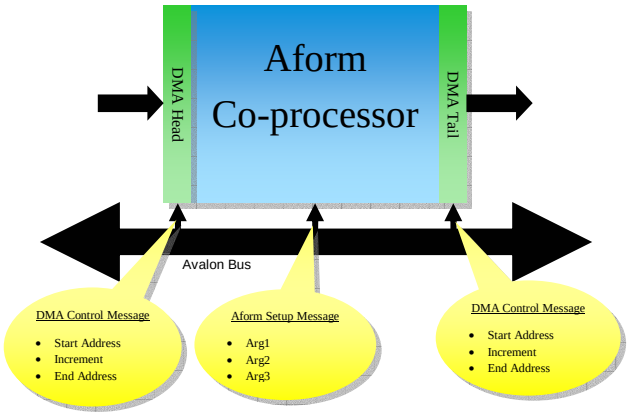

Figure 5. Co-processor interfaced to the Avalon bus via two half-DMA peripherals

The DMA Head of Figure 5 is used for data accesses that feed the input of the co-processor. The read master of the DMA head would most likely be fed by some form of on-chip memory or could alternately come from an off-chip memory source or analog-to-digital converter (ADC). The output of the DMA head is connected directly to the input of the Aform co-processor which eliminates the write master of the DMA head and its accompanying logic. Because of this, about half of the logic required for a regular DMA[2] is eliminated in this half-DMA read implementation.

The DMA Tail is used to access the output data of the co-processor and possibly feed it to an on or off-chip memory bank to wait for further processing. Similar to the DMA Head, the input of the DMA Tail is connected directly to the output of the Aform co-processor which eliminates the read master of DMA Tail and its accompanying logic in this half-DMA write implementation.

The half-DMA peripheral illustrated in this paper is a standard peripheral to the SOPC Builder system integration software that comes standard with the Altera® Quartus® II design software. The Avalon™ bus is the primary bus interface employed by the Nios® soft core embedded processor. The Avalon bus is a simple bus architecture designed for connecting on-chip processors and peripherals together into a system–on–a–programmable chip (SOPC). The Avalon bus is an interface that specifies the port connections between master and slave components, and specifies the timing by which these components communicate. The Avalon bus also supports simultaneous multi-mastering where multiple masters are allowed to perform bus transactions at the same time, as long as they do not access the same slave during the same bus cycle [3]]. In the event that multiple masters attempt to access the same slave at the same time, a built-in arbitrator prioritizes accesses to that slave.

SOPC Builder's flexible Avalon bus fabric combined with its simultaneous multi-mastering capability enables several co-processors, connected using this DMA/co-processor enabled approach, to run in parallel with minimum bus contention. An embedded processor such as Nios can be used to perform the control portion of the user's application while the various co-processors do the more computationally intensive tasks, meeting the parallel processing requirements of the rest of the signal processing application. [4]]

5. EASE OF USE AND INTEGRATION

The DMA/co-processor enabled concept shown in Figure 5 can be easily implemented using Altera's SOPC Builder system integration tool. SOPC Builder is an automated system development tool that dramatically simplifies the task of assembling system-on-a-programmable-chip (SOPC) architectures comprised of various commonly used components including processors, busses, memories, co-processors, and external interfaces [5]].

SOPC Builder comes packaged with a large library of parameterizable components and can easily incorporate user-

[2] A regular 32-bit DMA with a length register width of 16 and FIFO depth of 8 utilizes approximately 584 Logic Elements (LEs) in an Altera Stratix device.

defined custom logic blocks or 3rd-party intellectual property (IP) [6]]. Building custom co-processors like those described in this paper is vastly simplified through the availability of predefined IP blocks for common signal processing functions for general and specific applications including filtering, communications and image processing. Altera and the Altera Megafunction Partners Program (AMPP) offer a large portfolio of these parameterizable IP blocks fully optimized for Altera architectures. These IP blocks can be further enhanced for use as co-processors utilizing the DMA enabled architecture defined above.

SOPC Builder allows the user to specify bus topologies with multiple master and slave components [7]]. Each of these components can be selected from an extensive library within SOPC Builder and uniquely customized using dedicated wizards. If a system contains multiple masters (such as two processors, or a processor and a DMA peripheral), SOPC Builder automatically generates the arbitration logic to intelligently connect them. SOPC Builder uses slave-side arbitration technology to optimize system performance by enabling simultaneous multi-mastering on the Avalon bus. SOPC Builder will also automatically generate interrupt controllers (IRQs), based on the interrupt priority specified by the user within the SOPC Builder user interface [6]].

Users are able to specify the base memory address for each component using the SOPC Builder user interface. SOPC Builder will immediately inform the user if an addressing overlap has occurred due to conflicting assignments of addresses. SOPC Builder will then generate the appropriate memory map and header files for the user's customized design.

A key aspect in integrating software systems with hardware architectures is the definition and availability of device drivers and API routines. SOPC Builder is capable of automatically creating the API routines and device drivers for the specific co-processor IP function that you include in your system within SOPC Builder [6]]. Also available are µC/OS-II, µClinux by Microtronix, and ATI Nucleus by Mentor Graphics, three real-time operating system (RTOS) kernels that are supported by Altera's Nios embedded processor, each allowing the use of multi-threaded processes in your C/C++ design [8]].

Using SOPC Builder, repetitive tasks like creating HDL for commonly used peripherals, integrating these peripherals and the processor, designing master/slave arbitration schemes and interrupt controlling are all handled by the tool. Apart from the hardware-generation aspect, SOPC Builder also provides the user with an integrated software development environment by automatically generating header files, peripheral drivers and custom software libraries. With the integrated hardware/software development environment in SOPC Builder, changes to the hardware are immediately reflected in the software development environment. The design of a Nios embedded processor with any combination of co-processor or peripheral is as simple and convenient as clicking a few buttons in the SOPC Builder user interface.

6. EEMBC FIR BENCHMARK

To demonstrate the performance of this methodology, we tested it using the EDN Embedded Microprocessor Benchmark Consortium (EEMBC) Automotive/Industrial FIR Benchmark algorithm [9]]. This EEMBC benchmark algorithm simulates an embedded automotive/industrial application where the CPU performs a FIR filtering sample on 16-bit or 32-bit fixed-point values [10]]. The benchmark algorithm flowchart is shown in Figure 6.

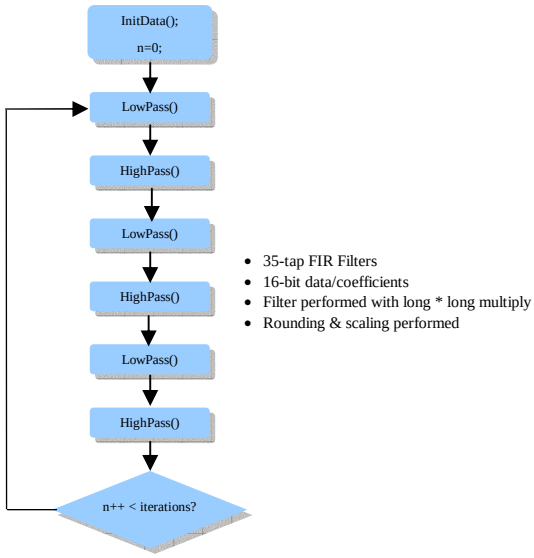

Figure 6. EEMBC Automotive/Industrial FIR Benchmark Algorithm [10]]

We ran this benchmark algorithm on Altera's Stratix™ FPGA architecture and compared it to several different processor vendors including AMD, IBM and Motorola. The benchmark results published by EEMBC for these vendors are shown in Table 1.

Table 1. EEMBC FIR Benchmark Results [12]]

Item #	Architecture (System f_{MAX})	MFIRs/sec[3]
1	AMD-K62 (400 Mhz)	0.10
2	AMD-K62 (500 Mhz)	0.12
3	IBM PPC7400 (500 Mhz)	0.24
4	IBM PPC603e (300 Mhz)	0.13
5	Motorola MPC7455 (1 Ghz)	0.60

The comparison results achieved by Altera for this benchmark algorithm are tabulated in Table 2. The FIR Compiler tool was used to design the FIR filter. The system f_{MAX} of 90 MHz for a Stratix implementation was easily achievable with a simple push-button compile.

[3] MegaFIR operations (1,000,000 FIR operations) per second.

Table 2. Altera Comparison Benchmark Results

Item #	Architecture (System f_{MAX})	MFIRs/sec	Size (LEs)
1	Stratix – Parallel FIR (90 Mhz)	15.00	21860
2	Stratix – Serial FIR (90 Mhz)	0.80	4155

7. SUMMARY AND CONCLUSIONS

The concept of using co-processors to perform hardware acceleration is not limited to DSP processors. With the use of the methodology described in this paper, it is possible to extend the current hardware accelerator co-processor offerings beyond that provided by current DSP processor vendors. With this method, users are given the flexibility to design and include their own custom hardware FPGA-based co-processor with their choice of processor whether it is an off-the-shelf processor or an on-chip embedded processor such as the Nios embedded processor. Tools like SOPC Builder enable the user to easily integrate and interface their peripheral to the Nios embedded processor. The use of multi-threaded software eliminates the bottleneck of sequential execution of processors, thus fully utilizing the parallelism provided by hardware co-processors and hardware accelerators.

8. REFERENCES

[1] TMS320C6414, TMS320C6415, TMS320C6416 Fixed-Point Digital Signal Processors Data Sheet, Texas Instruments, February 2001.

[2] S. Oaks and H. Wong "Java Threads", 2nd edition, O'Reilly, 1999.

[3] Nios Embedded Processor Peripherals Reference Manual, Altera Corp., January 2002.

[4] Avalon Bus Specification Reference Manual, Altera Corp., July 2002.

[5] Nios Soft Core Embedded Processor Data Sheet, Altera Corp., June 2000.

[6] SOPC Builder. http://www.altera.com/products/software/system/products/sopc/sop-index.html

[7] SOPC Builder Design Flow & Features. http://www.altera.com/products/software/system/products/sopc/design/sop-design_flow.html

[8] SOPC Builder Data Sheet, Altera Corp., September 2002.

[9] Third Party Tools for the Nios Embedded Processor. http://www.altera.com/products/devices/nios/utilities/nio-third_party.html

[10] EEMBC Website. http://www.eembc.org

[11] EEMBC Automotive/Industrial FIR Benchmark Datasheet. http://www.eembc.org/Benchmark/datasheet/AutoIndy/Autoindy16.asp

[12] EEMBC Benchmark Results. http://www.eembc.org/benchmark/benchmain.asp