USING FPGA CO-PROCESSORS IN SOFTWARE DEFINED RADIO SYSTEM ARCHITECTURES

Paul Ekas (Altera Corporation, San Jose, Calif., U.S.A., pekas@altera.com)

ABSTRACT

SDR systems require flexibility, long-lifecycles, and low-costs and power. An optimal implementation architecture leverages the advantages of processors and field programmable gate arrays (FPGAs) which provides a fully programmable architecture capable of delivering the high signal processing performance demands of SDR. In these heterogeneous architectures, the control processing is implemented on the processor and the dataflow processing is implemented on the FPGA fabric. In many cases, this can be considered a co-processing architecture as defined by the 3 permutations of co-processors: pre-processing, post-processing, and co-processing. Altera supports these three architectures with development tools enabling co-processing design and integration.

1. INTRODUCTION

When designing a cellular baseband SDR system, long-term system flexibility is always a particular concern. SDR systems must be able to operate under present conditions, but also be easily enhanced to meet changing future requirements. As a result, in SDR design implementing an architecture that can meet reconfigurability requirements, as well as cost, power and performance demands is always a key challenge. These reconfigurabilty requirements can be implemented in hardware using processor-based components, FPGA components, or a combination of the two. The author believes that, in general, optimal SDR system architectures are best implemented using a combination of these two types of semiconductor technologies.

2. SDR CONCEPTUAL ARCHITECTURE – DYNAMIC FUNCTIONS

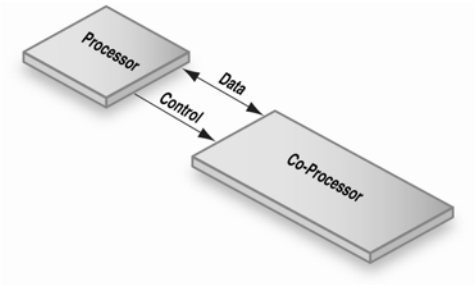
As has been discussed elsewhere, [1] three different techniques can be used to implement system reconfigurabilty: using parameterized radio and protocol modules, exchanging a single component within a module, or exchanging complete radio modules or protocol levels.

In the case of parameterized radio modules, the module design must take into account all the permutations necessary to implement the system. This approach is feasible and even required for a narrow range of operations, such as within global system for mobile communications (GSM)- or universal mobile telecommunication system (UMTS)-based standards. Typically, wireless communications standards are defined with a variety of operational modes that must be dynamically supported in real-time to take advantage of operating conditions, such as data throughput demands or to compensate for physical system impairments, such as multipath and user loading. Parameterized modules, however, are not a practical approach for supporting reconfiguration requirements across complex standards. In the case of GSM and UMTS implementations, for example, the physical layer communications are so complex and radically different technologically that in commercial implementations they are almost invariably designed separately in hardware and software and then merged at a higher level of software and hardware integration. Because of this inability to support reconfiguration across complex standards, parameterized modules do not offer an optimal approach for supporting future, and as yet undefined, radio standards.

The second technique—exchanging a single component—is useful where particular algorithms do not overlap in implementation, but serve similar functions. This can be seen, for example in forward error correction with Viterbi and Turbo decoding, where either algorithm can be chosen within the same standard, but where the physical implementation is significantly different. Texas Instruments, for example, has embedded two hardware co-processors on its TIC6416 device that implement these two algorithms as separate parameterized functions to support a variety of requirements across different wireless systems. Even these parameterized co-processors, however, are only capable of limited variations in reconfigurability and may not support new features that emerge as wireless standards evolve as has occurred, for example, in the 3GPP WCDMA standard with high-speed downlink packet access (HSDPA).

The third approach, exchanging complete radio modules or protocol layers, works well if the majority of the system's operations are significantly different. In GSM and UMTS where some components are effectively identical or simply permutations of the other, the physical layer is entirely different. In another example, GSM to cdma2000 requires a complete replacement of both the physical and network layer implementations because not only are the

Figure 1. Data sourced from and synched to a processor.



physical layers entirely different, the upper layer protocols, GSM-MAP and ANSI-41 are also too different to use the same software implementation across both.

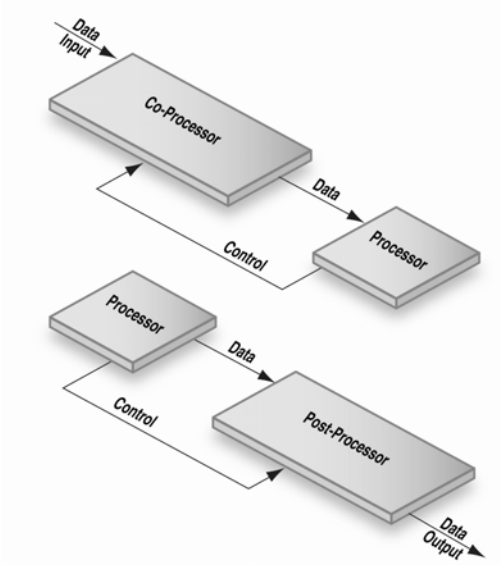
A combination of these approaches is required to implement existing and future radio standards. Based on the requirements for SDR to support both narrowband and wideband systems an architecture composed of a combination of processors and FPGAs can handle these three reconfigurable approaches across all required processing loads quite effectively. The processor can be used to switch dynamically between major sections of software when switching between standards. At the same time, the FPGA can be completely reconfigured, as necessary, to implement the architecture that has been customized for the particular standard currently in use. This particular approach meets the design demands for SDR by enabling the implementation of radio configurations that can be independently developed, tested and loaded onto radios.

3. IMPLEMENTING THE ARCHITECTURE

Having decided to use a combination of processors and co-processors, the next step in implementing a reconfigurable architecture is to identify the different types of operations required by the system and the type of processing best suited to each of the operations. These operations can be divided into two groups: first, system control and configuration and second, signal processing data path and control.

System control and configuration are related functions focused on maintaining and controlling the state of the system. System control is the dynamic operation within a wireless standard, while dynamic configuration changes the system from one wireless standard to another. These two control-intensive tasks require complex software implementations with a light computational load. In SDR they are generally written on top of CORBA [2] and the SCA [2]. In general, these system control and configuration functions are performed by control processors running large C-based or similar programs requiring both high levels of

Figure 2. Data sourced from a high-speed interface through the co-processor to the processor.

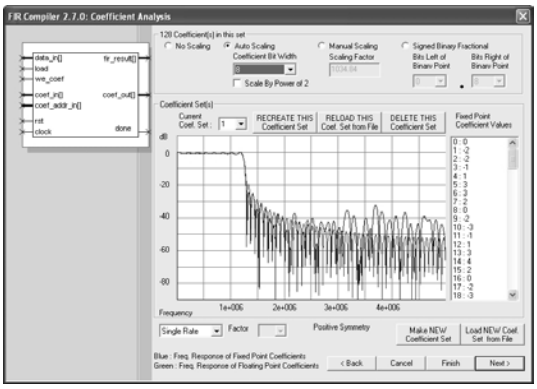


memory efficiency and maintenance by high-level C language tools. These system functions will likely reside on control processors.

Typically, the bulk of the processing load is taken up by the signal processing data path and control tasks. The physical layer communications in TDMA-, code-division multiple access (CDMA)- and OFDM-based systems along with encryption and some of the networking functions are all excellent examples. The concentrated computational load in signal processing makes it amenable to parallel and cascaded data path elements. In cases where data path elements are used, the related active real-time control function may also require that dedicated control logic be integrated with the data path processing. In those cases, such as TDMA and FM systems, where the signal processing does not consume significant processing capabilities, the signal processing and related control functions can be software implemented.

Depending on the wireless communications standard being supported, processing demands can vary significantly. Those with light processing demands may be best implemented as software-only systems, while those with higher processing demands are best implemented as software-plus-hardware based systems. The latter will be implemented in a combination of digital signal processors and FPGA-based dedicated logic architectures.

Figure 3. Data sourced from the co-processor to the processor.



4. USING CO-PROCESSORS TO OPTIMIZE PERFORMANCE

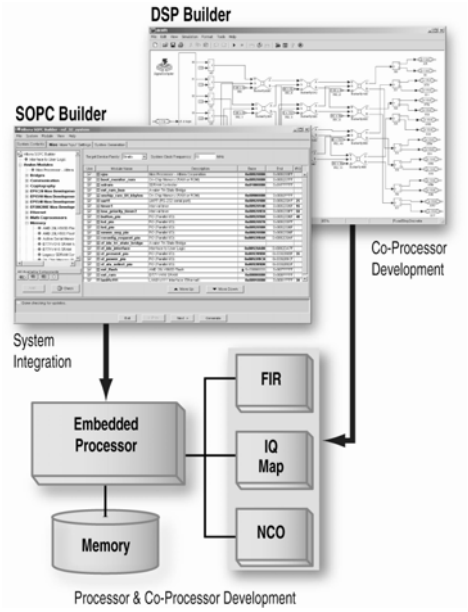
In many SDR systems a small percentage of the program code consumes the majority of the MIPS in time-consuming, error-prone and difficult-to-maintain recursive algorithms that are used to increase overall system performance. The majority of the code, which consumes only a small percentage of the MIPS, still reflects the majority of the system complexity.

FPGAS co-processors can be an efficient means to reduce the process load and minimize latency in the system. Benefiting from this approach, however, requires knowing how to determine whether any individual system design will benefit from a co-processor implementation. A poor choice, rather than improving system performance, adds unnecessary system complexity.

There are three very common system configurations where a co-processing implementation is very likely to enhance overall system performance. The first is a system where data is sourced from and then synched to a processor (figure 1). The second and third respectively are systems where data is either sourced from a high-speed interface through the co-processor to the processor (figure 2), or this path is reversed (figure 3). In all three of these configurations, the processor controls both the co-processor's functionality and operating parameters.

Examples of this first type of configuration include forward correction or channel equalization in narrowband systems for wireless applications. Examples of the second and third types include filtering, digital down-conversion or pre-distortion in wireless systems.

Figure 4. Altera's SOPC Builder.



5. AUTOMATING CO-PROCESSOR INTEGRATION

.

Traditionally, co-processor integration has been a very manual task, but today there are some tools on the market that can be used to automate the integration process. One of the most robust is Altera's system-on-a-programmable-chip (SOPC) Builder (figure 4).

SOPC Builder has an established history of several years use integrating complex peripherals into single- and multi-processor-based FPGAs. Applying SOPC Builder to support co-processor integration actually simply involved an evolution of how the tool was applied to address performance issues in customer systems. It did not require any technical changes. The tool supports a broad range of processors, peripherals and now co-processors. Using SOPC Builder's interactive menus, designers are able to set the parameters of the components they intend to use and then can choose the optimal Avalon switch architecture to connect the selected components. The tool will then automatically assemble the system in register transfer level (RTL) very high density language (VHDL) or Verilog, after which it generates a software driver file called Excalibur.h. Excalibur.h contains all the software interfaces for the blocks used in the system. It also includes software directions for the register and a memory map defined by the user's architectural selection. This correct-by-construction

approach accelerates system integration by months by eliminating the error prone and tedious manual development of low-level software drivers. In addition, once such blocks have been integrated into SOPC Builder they are easily reusable in subsequent designs.

While SOPC Builder enables the integration of complex processor-, peripheral- and co-processor-based systems, the co-processors used will likely be unique, tailored to the specific application and requiring custom development for each individual system. There are a number of options to choose from when building a co-processor. The specifications and C code, for example, can be translated into a VHDL or Verilog hardware architecture and subsequently be recaptured in a system block diagram using Altera's MathWorks Simulink interface—DSP Builder. On the other hand, some methodologies in currently in development allow behavioral synthesis of the source C code directly into the co-processor. SOPC Builder is readily able to accept the output from any of these design flows and likely to add intellectual property (IP) relying upon these more automated flows in the near future. At the present time, however, the most robust flows include VHDL, Verilog and DSP Builder.

DSP Builder is an extension of the MathWorks Simulink environment that allows block capture through hardware implementation. It has been closely integrated into SOPC Builder specifically to enable the development of co-processors. With DSP Builder, system designers can assemble parameterized blocks representing a plethora of functions ranging from muxes through fully parameterized finite impulse response (FIR) filters. Once a dataflow system has been captured in DSP Builder, it can be exported for use as a co-processor in any processor-based system assembled by SOPC Builder.

6. **CONCLUSION**

SDR systems must be designed in a way to ensure they are sufficiently flexible and reconfigurable to meet both present and changing standards and operating requirements. A system architecture that combines processors and co-processors in a hardware implementation is likely to produce an optimal result.

Three different techniques are often used to implement system reconfigurability, parameterized radio and protocol modules, exchanging a single component within a module or exchanging complete radio modules or protocol levels. Whichever technique is chosen, a combination of processors and FPGA-based co-processors can handle it.

It is critical when implementing the reconfigurable architecture to identify the operations required and to determine the best type of processing required. Generally, these operations can be divided into two different groups, system control and configuration and signal processing path and control. System control functions will largely reside on the control processor, while the signal processing data path and control functions are better assigned to an FPGA-based co-processor or DSP.

By off-loading MIPS-intensive algorithms to a co-processor, a designer can derive several significant system benefits. Most importantly, performance can be improved by an order of magnitude or more since a co-processor can significantly outperform the DSP-based software implementation. Once the processing load has been reduced, the main processor is available handle the majority of the code that absorbs only a small percentage of the required MIPS. This allows the designer to choose a lower performance and less costly main processor. In general, this reduction in main processor cost more than compensates for the expense of the FPGA-based co-processor implementation.

While integrating co-processors and processors has traditionally been a time-consuming manual process, tools are now available to automate the process. One of the most robust is Altera's SOPC Builder, which includes an extension of the MathWorks Simulink environment, known as DSP Builder, to facilitate co-processor integration. Using these automated tools, which can be expected to become even more powerful and flexible in the future, has greatly simplified co-processor implementation and can help significantly reduce system development time and costs. In addition, function blocks created using SOPC Builder can be stored for reuse in future designs, providing additional time and cost benefits.

7. References

[1] SDR: Enabling Technologies, ed. W. Tuttlebee, pub. Wiley and Sons, 2002

[2] "Joint Tactical Radio System (JTRS) SCA Developer's Guide", Rev. 1.1, Raytheon Company

NOTE: Many of the concepts discussed in this paper are also covered in a chapter in the yet to be published *Software Defined Radio: Baseband Technology for Cellular Systems,* Vol. 2 of the Wiley SDR Series "Enabling Technology".