VSIPL++/FPGA Design Methodology

Jules Bergmann, Air Force Research Laboratory/IFTC
Susan Emeny, ITT
Peter Bronowicz, ITT

We describe a hardware/software codesign methodology for hybrid hardware and software systems. The methodology integrates VSIPL++ for software design and a portable, composable hardware design method based on streams. The hardware design is portable and scalable from design/test systems to the target system and to future technologies. The methodology increases productivity by providing a concise function description in both hardware and software and by providing a streamlined interface between hardware and software. The methodology supports a design methodology from algorithms to embedded systems with hardware/software co-design, strong unit and system testing, and virtual breadboarding. It simplifies the integration of hardware and software to create high performance applications. It enables the use of predefined FPGA libraries for application acceleration.

A standard high-level synthesis hardware design methodology, using a register Transfer Logic (RTL) description in a Hardware Design Language (HDL), achieves portability and scalability. The design can be synthesized onto a range of Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuit (ASICs). Encapsulation of device specific optimizations into macro cells ensures high performance. Composable and highly reusable hardware units increase productivity. Hardware units use standardized interfaces for data exchange between them. Specifically, we chose a *stream interface* with flow control, appropriate for signal and image processing.

Our software design methodology builds applications using VSIPL++[1]. VSIPL++, a successor to the Vector Signal and Image Processing Library (VSIPL), addresses portability, performance, and productivity issues for embedded high-performance software design. The High Performance Embedded Computing Software Initiative (HPEC-SI)[2] is standardizing VSIPL++. VSIPL++ uses C++ object-oriented language features to improve the readability and expressiveness of programs, while delivering performance on par with traditional C and FORTRAN programs. Generic programming with templates enables custom compilations using machine features such as Single Instruction-Multiple Data (SIMD) instruction sets and the Portable Expression Template Engine[3].

Our infrastructure exchanges data between VSIPL++ and hardware streams, effectively combining hardware and software design. Communication across the software/hardware interface requires the conversion of data formats from VSIPL++ *blocks*, that is, memory-mapped data, to streaming data. We currently convert formats with *memory adapters*, hardware units on the FPGA that either read from FPGA memory to a flow-controlled stream of data or visa versa. `FPGABlocks`, a form of user-defined blocks storing data on the host side, behave in VSIPL++ just like built-in VSIPL++ blocks. A standard description of the functionality implemented on an FPGA helps manage possible FPGA configurations. This configuration file is read at run-time and `FunctionObjects`, corresponding to actual functions implemented on the FPGA, are created. Calling an `FPGAApply` routine moves data onto the FPGA from the hosts and configures the appropriate memory adapters to initiate the operation defined by a function object. `FPGABlocks` are *lazy*, i.e., they only move between the host and FPGA when necessary.

The methodology enables development of hardware to begin well before the target system is available. Individual units can be incrementally integrated and tested. Groups of units, sub-systems, and systems can be integrated into software and tested in the same way. Systems too large for a single FPGA can be spread across multiple FPGAs or multiple FPGA boards on network-connected machines. The stream interface makes this possible by allowing an arbitrary no-op to be placed between two units that will be

directly connected in the target system. In a virtual breadboard environment, this no-op transports a stream between multiple FPGAs, on the same board or on different network-connected hosts.

Our design methodology allows tightly coupled hardware-in-the-loop simulations to be easily constructed. Hardware-in-the-Loop Simulation is the practice of open or closed-loop simulation connecting software (often scene generation or sensor emulation) and hardware (often processing or guidance). The software portion of a hardware/software design implements the soft portions of the target system and simulates the model environment. The flow control nature of stream processing supports a mode of test where hardware operates at speed for a short period and then stalls while new stimulus is created/loaded (i.e. time stops).

We currently employ this methodology in two applications: the embedded system design of a space-based radar and the creation of a signal processing library for application acceleration. In a joint effort with the Jet Propulsion Laboratory, we are developing an on-board processor for a demonstration Moving Target Indicator/Synthetic Aperture Radar (MTI/SAR) space-based radar scheduled to fly in 2008. An FPGA front-end is capable of SAR processing (range compression, azimuth compression) and MTI preprocessing (pulse compression, Doppler filtering). A programmable backend performs MTI processing (adaptive Space-Time Adaptive Processing (STAP)). Although the system will not be fabricated until 2005 or flown until 2008, this design methodology permits hardware design to begin today, and a virtual breadboard of the entire system can be constructed on AFRL's hybrid cluster.

We are developing RStream, a set of stream components that implement common signal processing filtering, transforms, and utility functions. Some of these functions will be used for SBR, but the goal is to provide a library that can be used for VSIPL++ application acceleration.

At the time of this paper, we have implemented a prototype implementation of VSIPL++/FPGA integration extension for the Annapolis Firebird FPGA card, which uses Xilinx VirtexE FPGAs. We are currently extending this implementation to support the Annapolis Wildstar-II FPGA board, which contains two Xilinx Virtex-II 6M gate FPGAs. We are performing preliminary experiments on the Xilinx Virtex-II Pro, an FPGA which integrates a hybrid system (PowerPC processor and reconfigurable logic) onto a single chip.

There are a number of exciting areas for future work. As the benefits and limitations of our initial streaming protocol become better understood, opportunities arise to develop additional streaming protocols with complementary characteristics (e.g., blocks larger than a single word, embedded control). Currently we only interface between hardware and software with memory adapters. A number of other adapters are possible, such as a FIFO adapter that sends the data in a block to the FPGA for processing without logically placing the block onto the FPGA. There are a number of Computer Aided Design (CAD) future work areas. Finally, we could investigate other areas besides signal processing applications and embedded systems that could benefit from FPGA acceleration.

References

[1] VSIPL Vector Signal Image Processing Library http://www.vsipl.org

[2] HPEC-SI High Performance Embedded Computing Software Initiative http://www.hpec-si.org

[3] PETE Portable Extension Template Edward M. Rutledge, MIT Lincoln Laboratory
http://www.acl.lanl.gov/pete/html/index.html