Nallatech Software Tools on the Maxwell FPGA Parallel Computer

Craig P. Steffen
NCSA
csteffen@ncsa.uiuc.edu

June 18, 2008

Abstract

The FPGA High Performance Computing Alliance built a 64 FPGA parallel Computer called "Maxwell" at EPCC in Edinburgh. Half of the FPGAs are from Nallatech and half from AlphaData. This poster paper presents the experience of beginning to program the Nallatech half of the machine with the Nallatech software stack, consisting of DIME-C and DIMEtalk. Also covered is the glue-layer template library written to allow user software to interface to the FPGA more easily with the DIMEtalk libraries.

1 Introduction

Even though Field Programmable Gate Arrays (FPGAs) have increased in complexity enough to perform tasks normally associated with processors, their higher cost and difficult programming model have kept scientific collaborations with high computational needs from embracing the technology. This continued reluctance has led to a stagnation of interest in using FPGAs as computing devices, despite continued advances in FPGAs themselves and the compiler tools to program them. This situation is a classic chicken-and-egg problem; no one will take FPGAs for HPC (High-Performance Computing) seriously until someone proves an HPC problem works.

The FPGA High Performance Computing Alliance (FHPCA)[1] in Scotland built a 64-FPGA parallel computer system called "Maxwell" in 2007 and has been hosting academic researchers from around the world to come and use it as a test platform for scientific computation using FPGAs. Craig Steffen was invited by FHPCA to use Maxwell in 2007 and he was able to take them up on the offer in April of 2008. This paper highlights some of the experiences of programming this unique FPGA computing resource from the point of view of extending the Nallatech[2] software stack to make it a more streamlined tool for scientific computing.

2 Background

The Innovative Systems Lab (ISL) at NCSA[3] believes that scientific knowledge of computational problems is vital to the success of creating an application, and that asking such experts to design below the level of programming languages is not a reasonable thing to expect. ISL believes that programming should be accomplished using entirely software programming tools (not hardware design tools) or in the case of DIMEtalk[4], a hardware tool used only with the most explicit of instructions. The more hardware-level knowledge (clocks, buffers, and wires) can be abstracted away from the programmer, the more efficient the programming effort.

Many codes using FPGA for processing, including two of the three demonstrator applications on Maxwell, look at the FPGA as an off-load co-processor. That is, the FPGA is treated as a library where intensive routines are sent, each one from the local host processor. However, the communication fabric of Maxwell lends itself to another processing paradigm: 64 FPGAs connected in a tight communications structure that does not involve interaction with the host processor during algorithm execution. ISL is interested in Maxwell as a stand-alone machine, but also as a way of investigating the new Slipstream[5] FPGA CPU-socket module to be available from Nallatech.

3 The Maxwell Machine

Maxwell[6] consists of 32 computational blades contained within five IBM BladeCenter[7] chassis. Each double-width blade runs a conventional 2.8 Xeon processor and also holds two PCI-X FPGA accelerator cards. Half of the blades hold two Nallatech H101[8] cards each containing a Xilinx[9, 10] Virtex-4-160 FPGA, and the other sixteen hold two AlphaData[11] ADM-XRC-4FX each with a Xilinx Virtex-4-100. In addition to its 133 MHz PCI-X connection to its host, each card is cabled via Rocket-IO to its four nearest neighbors in a 2-D 8x8 torus configuration.

As an initial step, this work focused on the Nallatech software stack to program the Nallatech accelerators, so the effective topology of what follows is a cylinder 4 nodes high and 8 nodes in circumference.

The CPU side of the nodes are connected via gigabit Ethernet on the back-plane of the Blade Center crates.

4 The Nallatech Software Stack

The software stack to control an FPGA consists of three layers. First is the software tool that creates the computational logic on the FPGA. This is either a compiler or the user writing execution logic in a hardware design language (HDL)(either VHDL or Verilog). Second is the wiring layer, where the processing logic is connected to memory interfaces and other parts of the accelerator hardware to form a physical and logical device. The third layer is the software glue layer, which forms the interface between the user's application and the logical accelerator device. In addition to manufacturing FPGA accelerator hardware, Nallatech provides software tools that fill all of these roles to some degree.

4.1 DIME-C

Nallatech's FPGA compiler is DIME-C, which takes ANSI-like C programs and creates a logical element to be imported to DIMEtalk (see below). The DIME-C code is compiled in a logical block suitable to be placed on the fabric of an FPGA. Array arguments of the top-level DIME-C function become memory devices in the FPGA design.

The most important function of the DIME-C compiler is to *pipeline* the innermost loop of a calculation to maximize its efficiency. When code is compiled in DIME-C, a visualizer shows the loop structure of the program and indicates which sections have been pipeline successfully. This is the primary design consideration to make codes run fast using the DIME-C software stack.

The DIME-C compiler takes code at a reasonable level of abstraction for an algorithm designer and compiles it into a circuit description that will ultimately run on an FPGA. The tool also contains basic diagnostics to indicate when parts are pipeline, providing a guidance tool for tuning code. ISL has not yet explored the limitations of what the DIME-C tool can pipeline successfully.

4.2 DIMEtalk Design Tool

DIMEtalk[4] is a graphical design tool used to "wire" together logical functional blocks on the fabric of an FPGA. Each logical block is an HDL design that has been optimized for its task. The DIMEtalk tool connects logical blocks to each other and to elements that represent the pins on the FPGA and other devices outside the chip.

DIMEtalk is very much targeted at the electronics designers or system architects. It presents more detail to the user than is useful to a scientific application programmer. With sufficiently detailed documentation, DIMEtalk is a usable but cumbersome tool for connecting the user's DIME-C code into the hardware framework. Ideally, the HPC toolkit that comes with the H101 card (which is, after all, a scientific computing product) would merely have a tool at the wiring part of the software stack that would allow the user to select the number of SRAMS, DRAMS, and communication channels. Since there must be a one-to-one correspondence between array arguments of the uppermost DIME-C function and physical memories, perhaps the main functions of DIME-C could be performed by an entirely automatic piece of software?

4.3 DIMEtalk Example Code

The DIMEtalk tool is used to create the FPGA design, which is compiled using the Xilinx ISE tools to create a bitstream that is used to configure the FPGA. DIMEtalk also produces a file of template C code that configures Nallatech accelerators on that machine and provides a place to insert user code. DIMEtalk produces a project file suitable for building on Windows and a Makefile for compiling on a Linux OS.

The C example file produced by DIMEtalk is a vitally important template for initializing, loading, and controlling the Nallatech FPGA card(s) , but it is not structured in a way that lends it to being made part of a larger system code. Its model is to have the main() C function configure and start the Nallatech cards with a designated place set aside for user code.

Unfortunately, the model of inserting the user application within code built around the hardware is not suitable for use with most scientific applications. Scientific programs tend to be very large, with elaborate input, control, and initialization systems, to say nothing of their data movement. Software glue is much better able to be linked as a library into existing code. The ideal situation is to have function calls exported from the FPGA glue code as library function calls; as in "insert_this_function_call_in_your_code()".

5 User Software Abstraction

Prior to this work, the NCSA Innovative Systems Lab ported code blocks to the Nallatech H101 cards in the early part of 2007. We created a simple functional abstraction to the Nallatech card initialization and control functions. The functions in this abstraction were pieces of the Nallatech stub code that comes from the DIMEtalk tool. These functions were built such that the constants defined in the Nallatech network.h file would populate the proper constants to make the DIMEtalk network function properly.

The DIMEtalk function and the initial abstraction functions we created from it were designed assuming that any process invoking that library would have one and only one Nallatech card to initialize and control. The DIMEtalk compiler produces a network.h file for each FPGA bitstream that supplies C code with information about that bitstream's DIMEtalk network. If one piece of C code needs information about multiple bitstreams, the network.h files won't work in their raw form because the variables they define overlap will conflict if both are included. To use multiple unique bitstreams with multiple network.h files, the variables in those files must be modified to

prevent name conflicts before including them in the C code.

In this work, to make code work on Maxwell, we created newer abstraction functions, a Makefile, and code templates that would allow one program to be able to initialize and run two Nallatech accelerator cards on one machine using two different bitstreams and corresponding network.h definition files. It was decided that the user code must be written with the number of cards and bitstreams in mind. Any definitions that come from the network.h files must be pre-pended with a prefix that indicates which of the cards it comes from. For instance a raw network.h file contains a definition file of the variable NUMDEVICES. The user will create the corresponding function call for that card that uses the modified variable Nall00_NUMDEVICES, and the function call corresponding to the second card will use variables pre-pended by a different prefix, for example Nall01_NUMDEVICES.

The Makefile setup included with the example code includes several safeguards against the user accidentally confusing one card for the other. Before compiling each user file is checked for any instances of non-pre-pended versions of the network.h variables, and aborts compilation if any are found. The user must also move the network.h file for each bitstream to another file, network0.h or network1.h, and change the location of the corresponding bitstream within each of those. After checking the files for un-pre-pended variables, a perl script modifies the the networkN.h files to add the corresponding prefix to the defined variables in each of those files.

5.1 Multi-card API functions

First, the global and per-card initialization and shutdown functions are listed in the appendix in section A.1. Except for my_collection, the arguments of the per-card initialization correspond to constants defined in network.h. An unmodified network.h defines NUMDEVICES. When the user constructs their nallatech_init_next_card call, the user inserts real argument Nall00_NUMDEVICES corresponding to formal argument my_NUMDEVICES, and so on for the other arguments of that function.

General card-control functions are listed in section A.2. The run function starts the logical block that is the top DIME-C function. The wait function blocks until the DIME-C logical block stops executing. The write integer scalar arguments writes parameters to the DIME-C function that are single-value (non-array) arguments. Finally, functions operate DMA calls to load arrays of values into the memories on or attached to the FPGA and also reads values back to main RAM:

The function calls presented here and their accompanying Makefiles, templates and examples will be released as open-source software under the University of Illinois open-source license.

6 Summary

Parallel FPGA computing is still in its infancy. FPGA programming tools are still mostly targeted at single-machine applications. This work shows that adding a API glue layer to the Nallatech software stack creates a glue layer that can be used to begin building a scientific software tools on a parallel FPGA fabric.

7 Thanks

I would like to thank the FHPCA for the opportunity to come and work on the Maxwell machine. The staff of EPCC, the supercomputing centre at The University of Edinburgh, were very warm and welcoming. I would like to particularly thank Mark Parsons and Barry Collard for being my liaisons and fielding my questions.

Also great thanks to the Nallatech software team in Bristol, particularly Dan Denning and Richard Chamberlin, who have helped immensely with getting this software system off the ground.

References

[1] http://www.fhpca.org, June 2008. The FPGA High Performance Computing Alliance.

[2] http://www.nallatech.com. Nallatech Inc.

[3] http://ncsa.uiuc.edu. The National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign.

[4] http://www.nallatech.com/?node_id=1.2.2&id=19 , June 2008. Dime-C: Nallatech's C-to-HDL design tool.

[5] http://www.nallatech.com/emailDownload.php?downloadId=6. SlipStream: Nallatech's new FPGA in a CPU socket module.

[6] http://fhpca.org/publications.html. This page contains several presentations and on FHPCA and Maxwell.

[7] http://www-03.ibm.com/systems/bladecenter/, June 2008. IBM's Blade Center high-density computing solution.

[8] http://www.nallatech.com/?node_id=1.2.2&id=41 , June 2008. Virtex-4 based Nallatech HPC processor card.

[9] S. Trimberger. A reprogrammable gate array and applications. In *Proceedings of the IEEE*, volume 81, pages 1030–1041, 1993. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=231341.

[10] http://www.xilinx.com, June 2008. FPGA vendor.

[11] http://alphadata.com, June 2008. Alpha-Data: FPGA solution vendor.

A Function Prototypes

A.1 Initialization and shutdown functions

```
nallatech_accelerator_collection_t *nallatech_initialize_global(void);

int nallatech_init_next_card(nallatech_accelerator_collection_t *my_collection,
int my_NUMDEVICES, char **my_devicefiles,
DWORD **my_deviceinfo, int my_memory_map_0, DWORD my_default_timeout);

void nallatech_close_global(nallatech_accelerator_collection_t *my_collection);

void nallatech_close(nallatech_accelerator_t *my_accelerator);
```

A.2 Execution Functions

```
int nallatech_run(nallatech_accelerator_t *my_accel);

int nallatech_wait(nallatech_accelerator_t *my_accel);

int nallatech_run_wait(nallatech_accelerator_t *my_accel);

int nallatech_write_integer_scalar_args(nallatech_accelerator_t *my_accel,
int num_args, ...);
```

A.3 Data Transfer Functions

```
int nallatech_write_4byte_values(nallatech_accelerator_t *my_accel,
void *source_pointer, int destination_address,
int destination_node, int num_values);

int nallatech_write_8byte_values(nallatech_accelerator_t *my_accel,
void *source_pointer, int destination_address,
int destination_node, int num_values);

int nallatech_read_4byte_values(nallatech_accelerator_t *my_accel,
void *destination_pointer,int source_address, int source_node, int num_values);

int nallatech_read_8byte_values(nallatech_accelerator_t *my_accel,
void *destination_pointer,int source_address, int source_node, int num_values);
```