Tinker: Generating Custom Memory Architectures for Altera's OpenCL Compiler

Dustin Richmond Jeremy Blackstone Matthew Hogains Kevin Thai Ryan Kastner
Department of Computer Science and Engineering
University of California San Diego

***Abstract*—Tools for C/C++ based-hardware development have grown in popularity in recent years. However, the impact of these tools has been limited by their lack of support for integration with vendor IP, external memories, and communication peripherals. In this paper we introduce Tinker, an open-source Board Support Package generator for Altera's OpenCL Compiler. Board Support Packages define memory, communication, and IP ports for easy integration with high level synthesis cores. Tinker abstracts the low-level hardware details of hardware development when creating board support packages and greatly increases the flexibility of OpenCL development. Tinker currently generates custom memory architectures from user specifications. We use our tool to generate a variety of architectures and apply them to two application kernels.**

1. Introduction

The use of High Level Synthesis languages has grown in recent years from academic tools to industry standards. The current generation of industry High Level Synthesis tools use a common language like C/C++ to create hardware IP cores. C/C++ High Level Synthesis tools attempt to raise the abstraction of hardware development from the RTL level, to the system design level. However, integrating these ip blocks into working systems still requires a communication framework and hardware design experience to deploy an IP core in a system.

Thus, academics and FPGA vendors have developed end-to-end tools for deploying C/C++ code onto accelerator cards using a parallel runtime like OpenCL or CUDA [1], [2], [3]. This technique uses a C/C++ high level synthesis tool to compile computation kernels into IP Cores, system design tools to integrate the result into pre-specified development cards, and software to integrate the working system into an application. The board and vendor specific information these tools rely on to eliminate the details of system integration are called *Board Support Packages*. Depending on the vendor, Board Support Packages define memory and custom IP interfaces, memory and system address maps, data-memory layout, timing parameters and pin locations, to name a few.

These end-to-end frameworks completely eliminate low-level RTL development within the framework. However, the Board Support Packages can also be a hindrance to development. For example, in Altera's OpenCL Compiler board support packages are static, difficult to modify. They typically provide a subset of board IP for high level synthesis developers to avoid waste. Modifications require significant hardware design expertise. Given these drawbacks, the only attractive option is to allow developers to specify a board architecture prior to compilation.

One particular frustration in our research has been the inability to easily design custom memory architectures in Altera's OpenCL Compiler. At a high level a memory architecture defines the memory interfaces available on a board are grouped, resources are shared, and the interleaving pattern. A memory architecture can have profound impacts on timing closure, resource utilization, and performance of an OpenCL Kernel.

To address these issues we have developed Tinker, an open-source memory architecture generator for Altera OpenCL. Tinker generates custom memory architectures from user specifications, handling top level files, constraints, and IP instantiation, in one portable tool. Our tool is written in Python and Tcl and is available under a BSD 3-clause license at https://github.com/drichmond/tinker.

In this paper, we use our tool to demonstrate our preliminary work generating memory architectures and perform experiments to demonstrate its utility. Using our tool we compare the vendor-produced board to a generated board with identical parameters to evaluate our tool relative to baseline performance. Next, we use our tool to vary the memory type available and demonstrate application kernels using QDR memories.

There are three main contributions of this paper:

- An open-source tool for creating custom board support packages
- A study on the performance of our tool compared to existing board support packages
- Preliminary results using our tool to generate custom architectures for the Altera OpenCL Compiler

Section 2 describes the organization of an Altera Board Support Package and all the necessary details for understanding our tool, Tinker, described in Section 3. Section 4 describes the results of our experiments with baseline comparisons and heterogeneous optimizations. Section 5 concludes the paper.

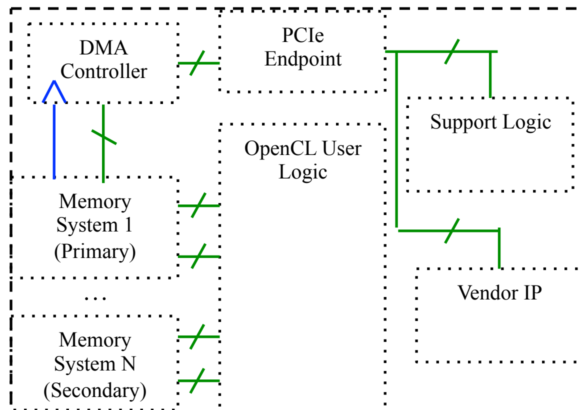

Figure 1. Hardware components of an OpenCL Board Support Package. Green wires indicate data busses, blue indicates clock networks.

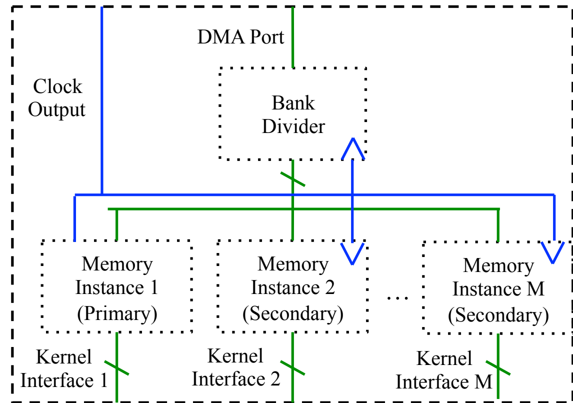

Figure 2. A simple memory system instantiating M memory instances, each providing a kernel interface, and a bank divider providing a contiguous memory space on top of an interleaving pattern

2. Board Support Packages

A Board Support Package contains three parts:

1) Libraries and drivers for communication [4]
2) A hardware system, project file, top level file, with kernel interfaces.
3) A metadata with hardware interfaces, address map, and data layout for the OpenCL Compiler.

2.1. Software

The software libraries in a board support package are broken down into two sub-components: A Memory-Mapped device library implementing memory-transfer API functions for OpenCL, and a Kernel-Mode PCIe driver for transferring data. We defer a discussion of the the Altera OpenCL Software stack to [4].

2.2. Hardware

There are three pieces that comprise the hardware system of an OpenCL Board Support Package:

1) A QSys system, instantiating the PCIe Endpoint, DMA Engine, memory interfaces, and other IP.
2) A top level file instantiating ports, the QSys System, and redefining wires to match the QSys System ports (If applicable).
3) A project file with timing, I/O and pin constraints.

A *memory architecture* defines how available *memory instances* are grouped into *memory systems*. Figure 2 shows a memory system composed of M memory instances and Figure 1 shows a memory architecture with two memory systems. A memory instance is a physical package, e.g. a DIMM. A memory system defines how low-level resources like PLL's, DLL's are shared to reduce resource usage and clock crossings. Each of the memory instances in a memory system provide a single non-uniform access memory port to compiler generated logic.

Figure 1 shows a simple memory architecture with two memory systems, with two memory instances in each system. A memory architecture defines the interleaving pattern, bandwidth, and data placement. Consequently a memory architecture has profound impacts on timing closure, resource utilization, and kernel performance.

Section 3 describes how memory roles (Primary, Secondary and Independent) affect memory architectures and systems.

2.3. Metadata

The central component of a Board Support Package is the board_spec.xml file that defines the memory architecture described in Section 2.2. An example board_spec.xml is shown in Listing 1; The global_mem tag describes a memory system and its bandwidth. An interface tag describes a memory instance, the kernel port name, width address size and latency in nanoseconds. These parameters are described in more detail by [4].

3. Tinker

We describe Tinker, our tool for customizing memory architectures in board support packages.

3.1. Setup

To generate a custom memory architecture, Tinker requires a skeleton project that defines pin locations and parameters and instantiates the QSys system.

Creating a Tinker skeleton project is simple: First, engineer creates a top level file defining the host communication interface, memory, IP, and reference clock pins. Pins are guarded by *'ifdef* macros, which are placed in a board-specific XML file, described in Section 3.2. Second, the engineer instantiates the QSys wrapper for the OpenCL System, and uses System Verilog's auto-connection semantic (.*), which connects pins and ports during compile time.

Listing 1. An example board_spec.xml file.

```
1  <board version="0.9" name="de5net_a7">
2   <global_mem max_bandwidth="25600">
3     <interface port="kernel_mem0" width="512" address="0x00000000" size="0x80000000"
      latency="240"/>
4     <interface port="kernel_mem1" width="512" address="0x80000000" size="0x80000000"
      latency="240"/>
5   </global_mem>
6   </board>
```

Finally, the engineer creates a quartus project file, with pin locations, I/O Standard, and timing constraints.

With the pin definitions, top level instantiations, and macros no further verilog modifications are necessary in Tinker. We emphasize that many vendors provide “Golden” skeleton projects with pin locations, I/O standard, and timing constraints pre-defined so we envision that board Vendors can easily support Tinker this as part of their standard OpenCL Board Package.

3.2. Generating an Architecture

To generate a custom memory architecture, the user writes a memory architecture specification in a text file with required architecture and grouping parameters, and optional parameters like interleaving pattern [4]. Tinker reads the specification and generates a complete board support package from the skeleton project that can be used in the OpenCL Compiler.

During generation Tinker decides how to group instances and assign roles to improve timing performance, share limited resources like PLLs, DLLs and On-chip Termination pins, and leverage physical locality (encoded in the board-specific XML file). Roles do not affect kernel interfaces, but they do affect kernel-data layout, host-memory transfer performance and timing closure. How these roles affect instantiation is described in Section 3.3. Each memory instance can theoretically take one of three roles, shown in Figure 2

- A *primary* memory instantiates it’s own PLL, drives the bank divider, and the clock output. It is the master for PLL, DLL, and OCT sharing relationships.
- A *secondary* memory instance shares all PLLs, DLLs and OCT pins from a primary memory instance, and thus operates in the same clock domain.
- An *independent* memory instance instantiates its own PLL and DLL, and can share an OCT pin. An independent memory operates in a separate clock domain with clock crossing interfaces.

Finally, the tool determines the role of each memory system to build the architecture. A memory system of M memory instances is shown in Figure 2. A primary memory system drives the clock of the DMA Engine, and an secondary memory system operates in its own clock domain and instantiates clock crossing logic.

Tinker sets the memory architecture by writing two files: First, a *board.vh* verilog macro file to enable pins in the top level file. Second, Tinker creates a board_specification.xml, an extension of the board_spec.xml files shown in Listing 1. The board_specification.xml file defines how memories are instantiated and connected during compilation. The board_specification.xml file is necessary because Quartus fails to compile when the board_spec.xml file contains unknown tags.

Listing 2 shows how the board_specification.xml has additional parameters that define the role of each memory system, a unique index, type, clock frequencies frequency, and the fabric-to-memory-clock ratio under the global_memory section. Each memory instance is assigned an id corresponding to a memory interface id in the board-specific xml file, a role, and if the interface is not primary, which interfaces are shared.

3.3. Compiling a board

During compilation, all IP on a custom board must be instantiated and connected according to the user specification. This is done using parameterized Tcl scripts, replacing unparameterized .qsys files used by board vendors.

The process of instantiating starts with a script we have written called Tinker.tcl. Tinker.tcl instantiates DMA, PCIe, support and status logic, and then instantiates the memory systems. Memory systems are generated using the parameters defined in board_specification.xml. These scripts instantiate clock crossing logic (non-primary) or pipeline stages (primary) and memory instances. Each memory instance instantiates clock-crossing logic (independent) or a pipeline stage (primary, secondary), provides reset and clock inputs, outputs and sharing interfaces, and instantiates support logic.

Using layers of .tcl scripts provides the abstraction of unified ports and parameters between different types of memory types of memory (e.g. DDR, QDR, and Local).

4. Results

We now present preliminary results using Tinker. These results demonstrate that our tool can generate architectures with comparable in resource consumption and performance, and provide alternative avenues for optimization.

Our tests were compiled using the Altera OpenCL compiler and Quartus Prime 15.1, for a Terasic DE5-Net FPGA Development with 2 800-MHz DDR3 banks, and

Listing 2. An example board_specification.xml file that defining a Tinker memory architecture.

```
1  <board version="0.9" name="de5net_a7">
2   <global_mem name="DDR0" max_bandwidth="25600" interleaved_bytes="1024" type="DDR3"
    index="0" mem_frequency_mhz="800.0" ref_frequency_mhz="50.0" ratio="Quarter"
    role="primary" width="512">
3    <interface port="kernel_0_ddr3a_rw" width="512" address="0x00000000"
     size="0x80000000" latency="240" id="a" role="primary" />
4    <interface port="kernel_0_ddr3b_rw" width="512" address="0x80000000"
     size="0x80000000" latency="240" id="b" role="secondary" primary="a"
     shared="pll,dll,oct"/>
5   </global_mem>
6  </board>
```

4 450-MHz QDRII banks. We have compiled our kernels for three Board Support Packages: A Vendor BSP with 2 DDR3 instances (Vendor BSP), a Tinker board with 2 DDR3 instances (2x DDR 2GB), and a Tinker board with 4 QDRII instances (4x QDR 8MB). All board settings that could influence area such as maximum burst size, number of in-flight requests or responses, and clock-crossing depths were consistent across architectures.

For our preliminary results we test two application kernels: Dense-Matrix Multiply (DMM) and Sparse-Matrix Multiply (SMM). Each kernel instantiated a single work-item on the FPGA without annotation-based optimizations. Performance results are reported as average Integer Operations Per Second (IOPS) over 10 runs, normalized to 250 MHz.

Our results are shown in Tables 1, and 2.

TABLE 1. Area and Performance Results for DMM

Board Type	% ALMs	% DSPs	% M20Ks	IOPS
Vendor BSP	20	3	17	220M
2x DDR 2GB	16	3	18	210M
4x QDR 8MB	10	3	8	236M

Table 1 demonstrates three results: First, comparing the Vendor BSP on (Line 1) and our Tinker board (Line 2) we can see that our tool produces an architecture with similar resource consumption and performance. Second, comparing all three BSPs, we can see that using different types of memories can reduce resource consumption. Finally, by comparing the performance results in the final column we can see that our BSPs perform comparably to the vendor BSP. Line 3, using QDR memories, has slightly higher performance that either DDR based board. Our current theory is that QDR provides slightly higher performance because of its simultaneous read-write ports. However, we emphasize that in an embarassingly parallel algorithm like Dense Matrix Multiply, DDR still has higher performance potential due to its bandwidth capacity. For this reason, we also demonstrate results from Sparse Matrix Multiply, which has a more random access pattern and less parallelism.

Table 2 corroborates the results found in Table 1. Resource consumption of the Vendor BSP and our two-DDR BSP are roughly equivalent. One surprising result from Sparse Matrix Multiply is that QDR does not perform substantially better, even though it supports better random-memory and read/write performance. Because the performance results in Table 2 and Table 1 are of similar magnitude, we suspect that the performance is limited by the kernel frequency, and we intend to explore the benefits of using QDR with random memory access patterns in depth in our future work.

TABLE 2. Area and Performance Results for SMM

Board Type	% ALMs	% DSPs	% M20Ks	IOPS
Vendor BSP	20	2	17	228M
2x DDR 2GB	16	2	18	238M
4x QDR 8MB	11	2	8	245M

5. Conclusion

In this paper, we have demonstrated our initial results from Tinker, our tool to create custom memory architectures for Altera's OpenCL Compiler. Our results demonstrate that our tool produces Board Support Packages that are comparable to those provided by vendors. Our future work will build on this, creating heterogeneous boards, and using them to solve larger problems, such as graph traversal.

References

[1] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, "From opencl to high-performance hardware on fpgas," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 531–534.

[2] A. Papakonstantinou, K. Gururaj, J. Stratton, D. Chen, J. Cong, and W.-M. Hwu, "Fcuda: Enabling efficient compilation of cuda kernels onto fpgas," in *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, July 2009, pp. 35–42.

[3] L. Wirbel, "Xilinx sdaccel: A unified development environment for tomorrows data center," 2014. [Online]. Available: http://www.xilinx.com/publications/prod_mktg/sdx/sdaccel-wp.pdf

[4] A. Corporation, *Altera SDK for OpenCL: Custom Platform Toolkit User Guide*, Altera Corporation. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/ug_aocl_custom_platform_toolkit.pdf