Compiling Software Code to FPGA-Based Application Accelerator Processors for the XD1

Doug Johnson
David Gardner
Celoxica, Inc.
doug.johnson@celoxica.com
+1-310-543-2468
HPC Leader

XD1 FPGA systems

Tools for Programming FPGA from Software

- Make FPGA programming transparent for High-Performance Computing users
- Enable FPGA use to enhance compute system performance and flexibility
- Open new applications and new market opportunities for Cray HPC solutions
Algorithm Implementation in FPGAs

CUG 2006

Slide 3

Algorithm Design (C code)

SW Compiled for Processor

Celoxica’s Software Compiled Silicon

Traditional HW Methodologies

C Debug, Optimization, and Compilation

HW/SW Partitioning

Hand Translation to RTL

RTL Design Flow

Physical Design Flow

Application-Specific Hardware

Microprocessor

- Serial execution
- High power, cost
- Software programmable

HW/SW Accelerated System

- Optimized performance
- Optimized power, cost
- SW programming flexibility

Conventional HW Design Path

Celoxica Software-to-Silicon Programming Path

Software-to-Silicon Programming Path

Traditional HW Methodologies

Software-to-Silicon Programming Path

Conventional HW Design Path

Microprocessor

- Serial execution
- High power, cost
- Software programmable

Application-Specific Hardware

- Parallel performance
- Low power, cost
- Long, expensive design cycle

SW Compiled for Processor

Celoxica’s Software Compiled Silicon

Traditional HW Methodologies

C Debug, Optimization, and Compilation

HW/SW Partitioning

Hand Translation to RTL

RTL Design Flow

Physical Design Flow

Application-Specific Hardware

Microprocessor

- Serial execution
- High power, cost
- Software programmable

HW/SW Accelerated System

- Optimized performance
- Optimized power, cost
- SW programming flexibility

Conventional HW Design Path

Celoxica Software-to-Silicon Programming Path

Software-to-Silicon Programming Path

Conventional HW Design Path

Microprocessor

- Serial execution
- High power, cost
- Software programmable

Application-Specific Hardware

- Parallel performance
- Low power, cost
- Long, expensive design cycle

SW Compiled for Processor

Celoxica’s Software Compiled Silicon

Traditional HW Methodologies

C Debug, Optimization, and Compilation

HW/SW Partitioning

Hand Translation to RTL

RTL Design Flow

Physical Design Flow

Application-Specific Hardware

Microprocessor

- Serial execution
- High power, cost
- Software programmable

HW/SW Accelerated System

- Optimized performance
- Optimized power, cost
- SW programming flexibility

Conventional HW Design Path

Celoxica Software-to-Silicon Programming Path

Software-to-Silicon Programming Path

Conventional HW Design Path

Microprocessor

- Serial execution
- High power, cost
- Software programmable

Application-Specific Hardware

- Parallel performance
- Low power, cost
- Long, expensive design cycle
Programming the Cray XD1 FPGA with DK

Algorithmic C-Code

Hardware-Software Partitioning

C-Debug, Optimization and Compilation

Direct synthesis from algorithmic C-code to XD1 FPGA

Enables software engineers to program FPGAs without HDLs

Area and performance optimization by exploiting parallelism in FPGA architecture

Cray XD1 Compute Blade

Celoxica’s Software-to-Silicon Programming Path

Software Compiler

Accelerated FPGA

C-Debug, Optimization and Compilation

Software-to-Silicon Programming Path

Digital design suite

C-Debug, Optimization and Compilation

Software-to-Silicon Programming Path
Designing FPGA Hardware from C using the DK Design Suite
Algorithm design

- **Co-design**
  Provide rapid iteration of partitioning decisions throughout flow

- **Co-verification**
  Drive continuous system verification from concept to hardware

---

C-Synthesis

- **C to RTL**
  Generate human-readable VHDL and Verilog for ASIC RTL hand-off

- **C to FPGA**
  Direct implementation to device optimized programmable logic
Handel-C adds constructs to ANSI-C to enable DK to directly implement hardware

- fully synthesizable and based upon standard ANSI-C
- Implements C algorithm direct to optimized FPGA or outputs RTL from C

Software-only ANSI-C constructs:
- Recursion
- Side effects
- Standard libraries
- Malloc

Majority of ANSI-C constructs supported by DK:
- Control statements (if, switch, case, etc.)
- Integer Arithmetic Functions
- Pointers
- Basic types (Structures, Arrays etc.)
- #define
- #include

Handel-C Additions for hardware:
- Parallelism
- Timing
- Interfaces
- Clocks
- Macro pre-processor
- RAM/ROM
- Shared expression
- Communications
- Handel-C libraries
- Fixed-point library
- Bit manipulation
- Single/Double floating point library
Behavioral Synthesis is:

- the *automated transformation* from a *higher level of abstraction*
  (C/SystemC-based Algorithm or TLM implementation)
- *to a hardware implementation*
  (RTL description or gate netlist)

To do this ALL Behavioral Synthesis tools using ANY C/C++-based language must address the following:

- Concurrency
- Timing
- Data Types
- Communication
- Resource Sharing and Implementation

Celoxica Philosophy: Put control in the user’s hands!
Defining Concurrency – Handel-C

- Course and fine grain parallelism in Handel-C

```c
void main(void) {
    par{
        processA(...);
        processB(...);
        ...
    }
}
```

// 1 Clock Cycle
```
par{
    a=1;
    b=2;
    c=3;
}
```

// 3 Clock Cycles
```
a=1;
b=2;
c=3;
```
Assignments and delay statements take 1 clock cycle

Combinatorial expressions computed between clock edges

- Most complex expression determines clock period
- Example: takes 1+n cycles (n is number of iterations)

```c
index = 0; // 1 Cycle
while (index < length){
    if(table[index] = key)
        found=index; // 1 Cycle
    else
        index = index+1; // 1 Cycle
}
```
- Handel-C has one basic type - integer
- May be signed or unsigned
- Can be any width, not limited to 8, 16, 32 etc.
- Fixed-point data types also supported for fractional representation

Variables are mapped to hardware registers.

```c
void main(void)
{
    unsigned 6 a;
    a = 45;
}
```

\[
a = \begin{array}{cccc}
1 & 0 & 1 & 1 \\
\end{array}
\begin{array}{c}
0 & 1 & 0 & 1 \\
\end{array} = 0x2d
\]
Allow communication and synchronization between two parallel branches

- Semantics based on CSP: unbuffered (synchronous) send and receive

Declaration

- Specifies data type to be communicated

```plaintext
Chan unsigned 6 c;

{ ...
  c!a+1; //write a+1 to c
  ...
} ... ... ...

{ ...
  c?b; //read c to b
  ...
} 
```
Interfaces allow Handel-C designs to connect to external hardware and logic.

Three types of interfaces

- Buses – used for connecting to external pins
- Ports – used for creating connection points for external logic.
  - e.g. Creating the ports for a VHDL entity
- User Defined – used for including external logic blocks inside a Handel-C design.
  - e.g. Including an EDIF black box inside a design.
Mix C/C++ & HC (for the description of parallel algorithms)

C/C++ code used in simulation, functional test benches & HW-SW co-design

HC simulated cycle accurately and implemented as EDIF & RTL

- Project management
- Symbolic source browsers
- Syntax highlighting
- Cycle-accurate multithreaded symbolic debugger:
  - single step execution
  - break points;
  - variable watch windows
  - thread focus.
- Mix C/C++ & HC code
Mixed C/ C++/ HC Simulation

- Supports TLM, mixed abstraction modeling & simulation
- HC, C & C++ native support

- Source-level parallel debug
- Single stepping & break points
- HW simulated within the system model
- Fast simulation
- Functional verification with mixed languages
- Architectural exploration
- Refine & transform to HW
- C/C++ testbenches developed into models & used throughout code
- Connects with 3rd party simulators
- VHDL/ Verilog/ SystemC/ Simulink/ Software ISS models
- Auto Generation of EDIF netlist and IEEE RTL
- Static timing related directly to areas of source code for optimization
- Detailed time/area estimations supports experimentation
- Output nets names relate directly to source code.

- Set timing constraints for I/O signals
- Technology mapped (to LUTs) EDIF output
- Use of embedded ALU primitives (e.g. for Stratix DSP blocks & Virtex-II/IV multipliers)
- Automatic pipelining of RAM accesses: (e.g. Actel BlockRAM, Altera EAB and Xilinx BlockRAM)
- Retiming Synthesis
Implementing a Search Kernel Algorithm on Cray’s XD1 Using DK
Application Acceleration on the XD1

Application Acceleration

- Reconfigurable Computing
- Tightly coupled to Opteron
- FPGA acts like a programmable co-processor
- Performs vector operations
- Well-suited for:
  - Searching, sorting, signal processing, audio/video/image manipulation, encryption, error correction, coding/decoding, packet processing, random number generation.

Slide Courtesy of Cray Inc.
Hypercube Range Search Implementation

- Algorithm has a set of \( p \) points and a set of \( h \) hypercubes, both of which having \( d \) dimensions.
- System identifies each point-hypercube combination where the point lies completely within the hypercube
- All of the point and hypercube parameters are represented by 16-bit positive integers
- Result of the search kernel should be represented as bit-vectors where each bit represents a hit/miss flag for every point-hypercube combination
- Resulting bit-vectors must be written back into the processor’s main memory
- Parameters
  - Number of points will be limited to around \( 10^7 \)
  - Number of hypercubes, \( h \), was set at 64
  - Number of dimensions was set to 4
Handel-C Design Flow with DK4.0 for XD1

1. User Applications written in Handel-C
2. DK 4.0 for Cray XD1: Design/Verification/Debugging
3. Handel-C
4. DK RTL/EDIF Compiler
5. VHDL
6. EDIF
7. Xilinx XST PAR
8. bitfiles
9. Program XD1 FPGA

RTL Simulators: ModelSim/Riviera for additional timing analysis and verification
2.2 GHz Opteron

FPGA processing 64 hypercubes/cycle

140 million/sec \times 4 \text{ dimensions} \times 2 \times 64 \approx 72 \text{ billion comparisons/sec}

128,000 points processed 500 times
Calculation takes about 0.47 seconds
Acceleration vs Number of Hypercubes

Number of Hypercubes

Acceleration

0.0 20.0 40.0 60.0 80.0 100.0 120.0

1 113 225 337 449 561 673 785 897 1009