Abstracting the Hardware / Software Boundary through a Standard System Support Layer and Architecture

Erik Anderson
4 May 2007
Agenda

• Publications and proposal review.
• Problem background.
• Abstracting HW/SW boundary.
• Analysis, comparing HW and SW.
• Results
• Conclusions
Special Thanks to

- David Andrews
- Perry Alexander
- Douglass Niehaus
- Ron Sass
- Yang Zhang
- Jason Agron
- Fabrice Baijot
- Ed Komp
- Andy Schmidt
- Jim Stevens
- Wesley Peck
- Seth Warn
Academic Background

• Bachelor of Science in Computer Science, University of Kentucky, 1993 - 1997, Magna Cum Laude

• Doctoral Candidate in Electrical Engineering, Kansas University, 2003 - 2007
Publications

- “Enabling a Uniform Programming Model Across the Software/Hardware Boundary,” FCCM 2006
- “Supporting High Level Language Semantics within Hardware Resident Threads,” submitted to FPL 2007
- “Memory Hierarchy for MCSoPC Multithreaded Systems,” ERSA 2007
- “Run-time Services for Hybrid CPU/FPGA Systems on Chip,” RTSS 2006
Proposal Review

Proposed
- Augment the HWTI
- Extend support for key subset of Hthread API.
- Semantic and implementation differences.
- Hthread test suite.
- Application suite.

Completed
- Augmented HWTI with:
  - User interface and protocol.
  - Globally distributed local memory.
  - Function call stack.
- Extended support for key subset of Hthread API.
  - Remote procedural calls.
- Chapter 5 in dissertation
  - Context similarities.
- Hthread test suite.
  - Abstractions held.
- Application suite.
  - Framework for HLL to HDL.
History of Reconfigurable Computing

- 1959: Gerald Estrin’s Fixed plus Variable Architecture.
- 1984: Xilinx is founded.
- 1993: Athana proposed PRISM-I
- 2006: First 65nm FPGA released.

<table>
<thead>
<tr>
<th>Conference</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCCM 2006</td>
<td>25</td>
</tr>
<tr>
<td>FPL 2006</td>
<td>30</td>
</tr>
<tr>
<td>FPGA 2006</td>
<td>22</td>
</tr>
</tbody>
</table>

Reconfigurable Computing Technology

• Post-fabrication circuit design.
• Embedded cores, memory, and multipliers.

From xilinx.com
“Blessing and a Curse”

• FPGA’s can take on any computational model post-fabrication.
• But which one to use?

Flynn’s Taxonomy

Data Stream

Instruction Stream

SIMD  MIMD
SISD  MISD
Hardware Acceleration Model

- SISD or SIMD.
- Advantages:
  - Can be successful.
  - C to HDL tools.
- Disadvantages:
  - Custom interfaces between HW and SW.
    - Write once, run once.
  - Costly design-space exploration.
  - Does not use today’s MIMD programming models.
Abstract Interfaces

- Parallel programming models to abstract CPU / FPGA interface.
- CPU and FPGA both target an equivalent abstract interface.
- OS / Middleware layer provides communication and synchronization mechanism.
The thesis statement is:

Programming model and high level language constructs can be used to abstract the existing hardware/software boundary that currently exists between CPU and FPGA components.
Extending the Shared Memory Multi-Threaded Model to Hardware

- Pthreads programming
Extending the Shared Memory Multi-Threaded Model to Hardware

• Key Challenges
  – HW access to API library.
  – HW access to application data.
  – Eliminate custom interface to HW.
Extending the Shared Memory Multi-Threaded Model to Hardware

- Hthread’s Solutions
  - Access to the same communication medium.
  - Equal or equivalent synchronization services migrated to HW.
  - Standard system support layer.

![Hybridthreads System Diagram]
Hardware Thread Interface

- HWTI provides a standard register set for communication and synchronization services.
Creating a *Meaningful Abstraction*

- Communication and synchronization are solved.

- Problems persist:
  - “scratchpad” memory:
    - How to instantiate?
    - How to maintain the shared memory model?
  - System versus user function calls?
  - Creating threads from hardware?
Globally Distributed Local Memory

- Dual ported BRAM.
- “Globally Distributed” = All threads have access.
- “Local” = User logic access is through LOAD and STORE protocols.
Function Call Stack

- Abstract access to local memory.
- Consistent function call model.
  - Recursion.
- Works analogously to software based stack.
  - Only difference, user logic pushes “return state” value instead of “return instruction.”
Remote Procedural Calls

- Some functions too expensive to implement in HWTI.
- Utilize existing synchronization primitives to callout to a special software system thread to perform function.
Hardware / Software Duality

```c
hthread_syscall( syscall_xx, arg1, arg2 );
```
Hthread System Call Implementation Differences

• HW has dedicated resources allocated at synthesis time.
  – HW explicitly blocks.

• SW has shared resources allocated at runtime.
  – SW context switches.
Hthread Size and Performance Comparison

• Size
  – Definition
  – hthread_create / hthread_join
  – hthread_yield

• Performance
  – Definition
  – HW outperforms SW
    • Create/join notable exception
  – Hardware’s bus transactions
Demonstrating an Abstract Interface

• POSIX Test-suite adapted for Hthreads.
• Conformance tests
  – Version for SW, HW, and mixed.
• Stress tests
  – Version for SW, HW, and mixed.
• Abstractions held across SW/HW.
Demonstrating HLL Constructs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Demonstrates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort</td>
<td>Recursion, local variables, access to shared memory.</td>
</tr>
<tr>
<td>Factorial</td>
<td>Recursion, local variables.</td>
</tr>
<tr>
<td>Huffman</td>
<td>Sharing data between HW and SW</td>
</tr>
<tr>
<td>Haar DWT</td>
<td>Local array access, access to shared memory.</td>
</tr>
<tr>
<td>IDEA</td>
<td>Task level parallelism, local variables.</td>
</tr>
</tbody>
</table>
Function Call Stacks and Recursion

• Quicksort
  – Recursive
  – $O(n \log n)$ performance
Memory Latency

- IDEA encryption
  - Key and data location comparison.

![IDEA Encryption Comparing Location of Data and Key](image)
Task Level Parallelism

- Haar DWT
  - Software’s pseudo-concurrency.
  - Hardware’s true concurrency.

- Performance
  - 2 SW = 31.1ms
  - 1HW/SW = 16.5ms
  - 2 HW = 16.6ms
Future Work

• Memory latency for hardware threads.
• Leveraging reconfigurable computing.
• High level language to hardware descriptive language translation.
Conclusions

• Parallel programming models may be used to abstract CPU/FPGA boundary.
  – Threads communicate and synchronize with other threads without regard to location.

• Abstract virtual machine can be implemented in either HW or SW.
  – Created a framework for HLL to HDL.
Questions?
Supplemental Material
Function Call Stack Example

```c
void * threadFunction(int * argument) {
    int a;
    int b;
    int c;
    foo(&a, &b);
    //return state = x0102
    ...
}

void foo( int *a, int *b ) {
    int d;
    int e;
    //Stack shown here
    ...
}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td></td>
</tr>
<tr>
<td>0088</td>
<td>E</td>
</tr>
<tr>
<td>0084</td>
<td></td>
</tr>
<tr>
<td>0080</td>
<td>D</td>
</tr>
<tr>
<td>007A</td>
<td>x0000 0102</td>
</tr>
<tr>
<td>007B</td>
<td>x6300 0060</td>
</tr>
<tr>
<td>0074</td>
<td>2</td>
</tr>
<tr>
<td>0070</td>
<td>x6300 0060</td>
</tr>
<tr>
<td>006A</td>
<td>x6300 0064</td>
</tr>
<tr>
<td>0068</td>
<td>C</td>
</tr>
<tr>
<td>0064</td>
<td>B</td>
</tr>
<tr>
<td>0060</td>
<td>A</td>
</tr>
<tr>
<td>005A</td>
<td>x0000 0000</td>
</tr>
<tr>
<td>0058</td>
<td>x0000 0000</td>
</tr>
<tr>
<td>0054</td>
<td>1</td>
</tr>
<tr>
<td>0050</td>
<td>x0000 2340</td>
</tr>
</tbody>
</table>

Address | Value | Meaning
---------|-------|------------------------
next declared variable or parameter push
declared variable E
displayed variable D
user logic’s return state
frame pointer restore value
number of parameters passed to foo()
first parameter passed to foo(), address of A
second parameter passed to foo(), address of B
declared variable C
displayed variable B
displayed variable A
user logic’s return state
frame pointer restore value
number of parameters passed to the thread
argument of thread
Globally Distributed Local Memory

code:

```c
threadZero() {
    ...
    b++;
    ...
}

threadOne() {
    ...
    f++;
    ...
}

threadTwo() {
    ...
    h++;
    ...
}

threadThree() {
    ...
    q++;
    ...
}
```
Dynamic Memory Allocation

- Pre-allocated Heap.
- Light version of `malloc`, `calloc`, and `free`. 
Demonstration HLL Constructs: Quicksort

- HWTI maintains $O(n\log n)$ behavior.
- Cache-like performance.
Demonstration HLL Constructs: IDEA

- Benefits of task level parallelism.
- Comparison with Vuletic’s hardware threads.
Demonstration HLL
Applicability: Huffman

- Abstract data passing between SW and HW threads.
- Data cache on CPU.
Demonstration HLL
Applicability: Haar DWT

- Abstract interface vs meaningful abstract interface.
- Performance.
- Complexity.
Globally Distributed Local Memory

• “Cache like” performance.
• Maintains shared memory model.
• User access without bus transactions.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Global</th>
<th>Local</th>
<th>HWTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>51</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>Store</td>
<td>28</td>
<td>1</td>
<td>19</td>
</tr>
</tbody>
</table>
Join Danger

- Thread "Parent" signals the thread "RPC" to create a new thread.
- RPC creates the thread "Child" on behalf of Parent.
- Parent signals RPC to join on Child.

- RPC must wait for Child to exit before it can complete join operation.
- Child signal RPC to perform floating point operations. RPC can not respond since it is waiting on Child.
Remote Procedural Calls

- **Advantages:**
  - HW Access to shared libraries.
  - Complete support for hthread APIs.
- **Disadvantages:**
  - Interrupts the CPU.
  - Comparatively slow.

<table>
<thead>
<tr>
<th>Library Call</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>hthread_create</td>
<td>160µs</td>
</tr>
<tr>
<td>hthread_join</td>
<td>130µs</td>
</tr>
<tr>
<td>malloc</td>
<td>122µs</td>
</tr>
<tr>
<td>free</td>
<td>120µs</td>
</tr>
<tr>
<td>printf</td>
<td>1.66ms</td>
</tr>
<tr>
<td>cos</td>
<td>450µs</td>
</tr>
<tr>
<td>strcmp</td>
<td>114µs</td>
</tr>
<tr>
<td>hthread_join</td>
<td>130µs</td>
</tr>
</tbody>
</table>