SimXMD: Simulation-based HW/SW Co-Debugging for FPGA Embedded Systems

Ruediger Willenberg and Paul Chow
High-Performance Reconfigurable Computing Group
University of Toronto

September 9, 2014
The “When Harry Met Sally” rule of CAD
The “When Harry Met Sally” rule of CAD

“Women and Men can never be friends.”
The “When Harry Met Sally” rule of CAD

“Designers and Tools can never be friends."
The “When Harry Met Sally” rule of CAD

“Designers and Tools can never be friends. The Sex will always get in the way.”
The “When Harry Met Sally” rule of CAD

“Designers and Tools can never be friends. The Semantics will always get in the way.”
FPGA embedded systems
FPGA embedded systems
FPGA embedded systems

- CPU
  - Application
- Peripheral A
- Peripheral B
- other on-chip hardware

FPGA
FPGA embedded systems

CPU
Application

Peripheral A (not verified)

Peripheral B

other on-chip hardware (not verified)

FPGA
FPGA embedded systems
FPGA embedded systems

CPU
Application
Driver code (untested)

Peripheral A (not verified)

Peripheral B

other on-chip hardware (not verified)

FPGA
FPGA embedded systems

Optimal: Debug hardware, software and their interaction together
FPGA embedded systems

Optimal: Debug hardware, software and their interaction together
Embedded SW debugging chain

- GUI
- GDB
- TCP
- XMD
- JTAG
- CPU
- Xilinx FPGA

High-Performance Reconfigurable Computing Group ∙ University of Toronto
Embedded SW debugging chain

- Debugger on host system
Embedded SW debugging chain

- Debugger on host system
- Software runs on target system
Embedded SW debugging chain

- Debugger on host system
- Vendor-specific interface software
- Software runs on target system
Embedded SW debugging chain
HW debugging

ModelSim

system_tb.v

system.v
HW debugging

ModelSim

system_tb.v
system.v

• Cycle-accurate digital simulator
HW debugging

• Cycle-accurate digital simulator
• A system model in HDL
HW debugging

ModelSim

- system_tb.v
- system.v

- Cycle-accurate digital simulator
- A system model in HDL
- Testbench instantiates and stimulates model
HW debugging

ModelSim

- system_tb.v
- system.v

- Cycle-accurate digital simulator
- A system model in HDL
- Testbench instantiates and stimulates model
- All internal signals observable
HW debugging
HW/SW Co-Debugging?

GUI
GDB

ModelSim
system_tb.v
system.v

GDB
GUI
HW/SW Co-Debugging?

GUI
GDB
TCP
SimXMD
TCP
ModelSim
system_tb.v
system.v
GUI
GDB
TCP
HW/SW Co-Debugging?

- SimXMD: Simulation-based eXperimental Microprocessor Debugger
**HW/SW Co-Debugging?**

- **SimXMD**: Simulation-based eXperimental Microprocessor Debugger
- Translates debugger requests into simulator commands
Key SimXMD enablers

• GDB Remote Serial Protocol
  – Defines requests and replies for:
    • Setting/deleting breakpoints
    • Advancing execution by instruction, line, breakpoint
    • Reading registers or memory state
Key SimXMD enablers

• GDB Remote Serial Protocol
  – Defines requests and replies for:
    • Setting/deleting breakpoints
    • Advancing execution by instruction, line, breakpoint
    • Reading registers or memory state

• Xilinx MicroBlaze Trace Port
  – Reports all information about a finished instruction:
    • Instruction code and address
    • Register and memory writes
Key SimXMD enablers

- GDB Remote Serial Protocol
  - Defines requests and replies for:
    - Setting/deleting breakpoints
    - Advancing execution by instruction, line, breakpoint
    - Reading registers or memory state

- Xilinx MicroBlaze Trace Port
  - Reports all information about a finished instruction:
    - Instruction code and address
    - Register and memory writes

- ModelSim (Tcl) TCP server capability
  - Can receive remote commands and send back results
Operating SimXMD: Preparation

1. Simulation model generation by design tool
   – Currently: Xilinx Platform Studio

2. SimXMD is started (background operation)
   – Examines embedded project information
   – Modifies simulation model for Co-Debugging

3. Compilation of simulation model

4. Start of simulation

5. Start of preferred debugger (GUI)

6. Debugging at will
Operating SimXMD: Modes

- In *Run mode*, debugging drives the simulation

```c
int main()
{
    a = functionA();
    b = functionB(a);
    c = b + 5 * d;
    return c;
}
```
Operating SimXMD: Modes

- In **Run mode**, debugging drives the simulation.
Operating SimXMD: Modes

- In Run mode, debugging drives the simulation
- In Replay mode, debugging iterates over previously simulated data
Operating SimXMD: Modes

- In **Run mode**, debugging drives the simulation
- In **Replay mode**, debugging iterates over previously simulated data
SimXMD at work

```c
// store to buffer
for (t=0; t<PACKET_SIZE; t++)
{
    buffer_ptr[t] = rx_packet_decrypted[t];
    XGpio_DiscreteWrite(&LEDs, 1, buffer_ptr[t]);
}
```
Implementation: Debugging memory
Implementation: Debugging memory

- Digital hardware simulation models the complete memory hierarchy:
  - On-chip and external memory
  - All cache levels
  - Memory-mapped peripherals
Implementation: Debugging memory

• Digital hardware simulation models the complete memory hierarchy:
  – On-chip and external memory
  – All cache levels
  – Memory-mapped peripherals

• Software debugging uses a flat, linear memory model:
  – The debugger requests a (virtual) memory address
  – The target hardware determines and reads the physical location
SimXMD memory access logging
SimXMD memory access logging

GUI

GDB

TCP

SimXMD

TCP

ModelSim

system_tb.v

system.v

Boot Memory
SimXMD memory access logging

VPI: Verilog Procedural Interface
SimXMD memory access logging

VPI: Verilog Procedural Interface
SimXMD memory access logging

VPI: Verilog Procedural Interface
SimXMD tool support

• Xilinx Embedded Development Kit  >= 13.x
  – Xilinx MicroBlaze Processor  >= 8.x
• MentorGraphics ModelSim  >= 6.6g
• Linux Operating System
• Debuggers
  – Command-line GDB
  – Xilinx SDK (Eclipse)
  – DDD
  – KDbg
  – Nemiver
SimXMD modular architecture

- debugger: debug_base
- processor: core_base
- simulator: sim_base

debug_base
- processor: core_base
- debug_sock: QTcpServer
- debug_GDB

core_base
- debugger: debug_base
- simulator: sim_base
- core_Microblaze

sim_base
- processor: core_base
- sim_sock: QTcpSocket
- sim_Modelsim
SimXMD limitations
SimXMD limitations

• Debugger can’t modify variables, registers
SimXMD limitations

• Debugger can’t modify variables, registers
• Volatile memory locations might be inaccurate
  – Shared-memory multiprocessing
  – DMA, Busmastering
  – Memory-mapped peripherals
SimXMD limitations

- Debugger can’t modify variables, registers
- Volatile memory locations might be inaccurate
  - Shared-memory multiprocessing
  - DMA, Busmastering
  - Memory-mapped peripherals
- Trace Port reports actions after instruction completes; several cycles difference
SimXMD limitations

• Debugger can’t modify variables, registers
• Volatile memory locations might be inaccurate
  – Shared-memory multiprocesssing
  – DMA, Busmastering
  – Memory-mapped peripherals
• Trace Port reports actions after instruction completes; several cycles difference
• Not all MicroBlaze special registers reported
SimXMD limitations

- Debugger can’t modify variables, registers
- Volatile memory locations might be inaccurate
  - Shared-memory multiprocessing
  - DMA, Busmastering
  - Memory-mapped peripherals
- Trace Port reports actions after instruction completes; several cycles difference
- Not all MicroBlaze special registers reported
- Instruction code only from on-chip RAM
SimXMD and multiprocessors?
SimXMD and multiprocessors?

- Any one core in a multicore system can be selected for debugging
SimXMD and multiprocessors?

• Any one core in a multicore system can be selected for debugging

• Future work:
  – On-the-fly switching between cores
  – Concurrent debugging of several cores
SimXMD and multiprocessors?

• Any one core in a multicore system can be selected for debugging

• Future work:
  – On-the-fly switching between cores
  – Concurrent debugging of several cores

• The same memory volatility issues apply:
  – Logging of virtual memory accesses per processor
    • Different virtual addresses - same physical address?
  – Race conditions likely
SimXMD Performance

• How much do the SimXMD modifications slow down simulation?
SimXMD Performance

• How much do the SimXMD modifications slow down simulation?

• How slow is SimXMD debugging in comparison with debugging a real target?
SimXMD Performance

• How much do the SimXMD modifications slow down simulation?
• How slow is SimXMD debugging in comparison with debugging a real target?
• Test system:
  – Host: Intel i5 Nehalem 4-core, 2.5Ghz, 12GB RAM
  – Target: Xilinx Spartan 6 (Atlys board), JTAG Microblaze @ 100MHz, 64kB on-chip BRAM AXI bus, one GPIO peripheral
  – Application: Writing 32kB byte-by-byte into BRAM
# SimXMD overhead

<table>
<thead>
<tr>
<th>Write size</th>
<th>w/o SimXMD</th>
<th>w/ SimXMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 kByte</td>
<td>6.9 s</td>
<td>7.3 s</td>
</tr>
<tr>
<td>2 kByte</td>
<td>13.8 s</td>
<td>14.5 s</td>
</tr>
<tr>
<td>4 kByte</td>
<td>27.3 s</td>
<td>29.0 s</td>
</tr>
<tr>
<td>8 kByte</td>
<td>54.9 s</td>
<td>57.7 s</td>
</tr>
<tr>
<td>16 kByte</td>
<td>109.0 s</td>
<td>117.1 s</td>
</tr>
<tr>
<td>32 kByte</td>
<td>218.9 s</td>
<td>231.7 s</td>
</tr>
</tbody>
</table>
## SimXMD overhead

<table>
<thead>
<tr>
<th>Write size</th>
<th>w/o SimXMD</th>
<th>w/ SimXMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 kByte</td>
<td>6.9 s</td>
<td>7.3 s</td>
</tr>
<tr>
<td>2 kByte</td>
<td>13.8 s</td>
<td>14.5 s</td>
</tr>
<tr>
<td>4 kByte</td>
<td>27.3 s</td>
<td>29.0 s</td>
</tr>
<tr>
<td>8 kByte</td>
<td>54.9 s</td>
<td>57.7 s</td>
</tr>
<tr>
<td>16 kByte</td>
<td>109.0 s</td>
<td>117.1 s</td>
</tr>
<tr>
<td>32 kByte</td>
<td>218.9 s</td>
<td>231.7 s</td>
</tr>
</tbody>
</table>

Average overhead: 6.0%
SimXMD debugging speed

- Same system and application
- Let GDB execute script of 50 “steps” (1 code line)
- Average time for a single code step:

<table>
<thead>
<tr>
<th>Hardware with JTAG</th>
<th>1.350 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimXMD Run mode</td>
<td>0.850 s</td>
</tr>
<tr>
<td>SimXMD Replay mode</td>
<td>0.313 s</td>
</tr>
</tbody>
</table>
Conclusions
Conclusions

• SimXMD enables:
  – Simultaneous debugging of software and hardware
  – Hardware debugging “timed” by software sections
  – Software debugging without existing/implemented HW
Conclusions

- SimXMD enables:
  - Simultaneous debugging of software and hardware
  - Hardware debugging “timed” by software sections
  - Software debugging without existing/implemented HW
- SimXMD does not significantly slow down reasonable debugging efforts
Conclusions

• SimXMD enables:
  – Simultaneous debugging of software and hardware
  – Hardware debugging “timed” by software sections
  – Software debugging without existing/implemented HW

• SimXMD does not significantly slow down reasonable debugging efforts

• SimXMD is open source
Conclusions

• SimXMD enables:
  – Simultaneous debugging of software and hardware
  – Hardware debugging “timed” by software sections
  – Software debugging without existing/implemented HW

• SimXMD does not significantly slow down reasonable debugging efforts

• SimXMD is open source

• SimXMD’s modular architecture facilitates extension to other processors and tools
Conclusions

SimXMD can be downloaded at:

http://www.eecg.toronto.edu/~willenbe/simxmd
Conclusions

SimXMD can be downloaded at:
http://www.eecg.toronto.edu/~willenbe/simxmd

Thank you for your attention!

Questions?