

FPGAworld 2014

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



2

The “When Harry Met Sally” rule of CAD




3

The “When Harry Met Sally” rule of CAD



4


The “When Harry Met Sally” rule of CAD



“Designers and Tools
can never be friends.
The Sex
will always get in the way.”

5

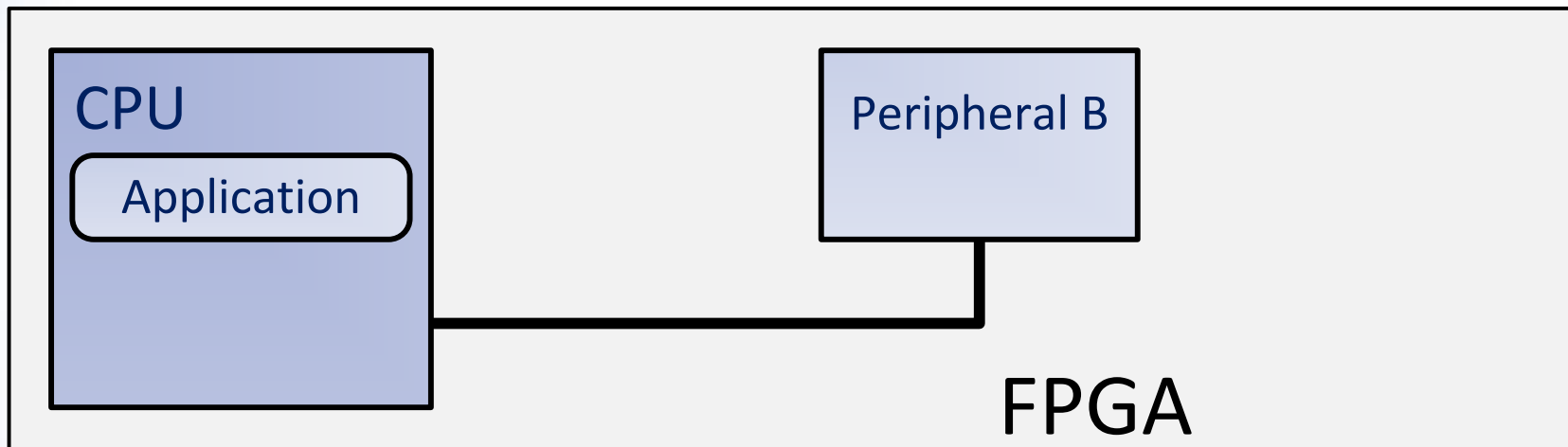
The “When Harry Met Sally” rule of CAD



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

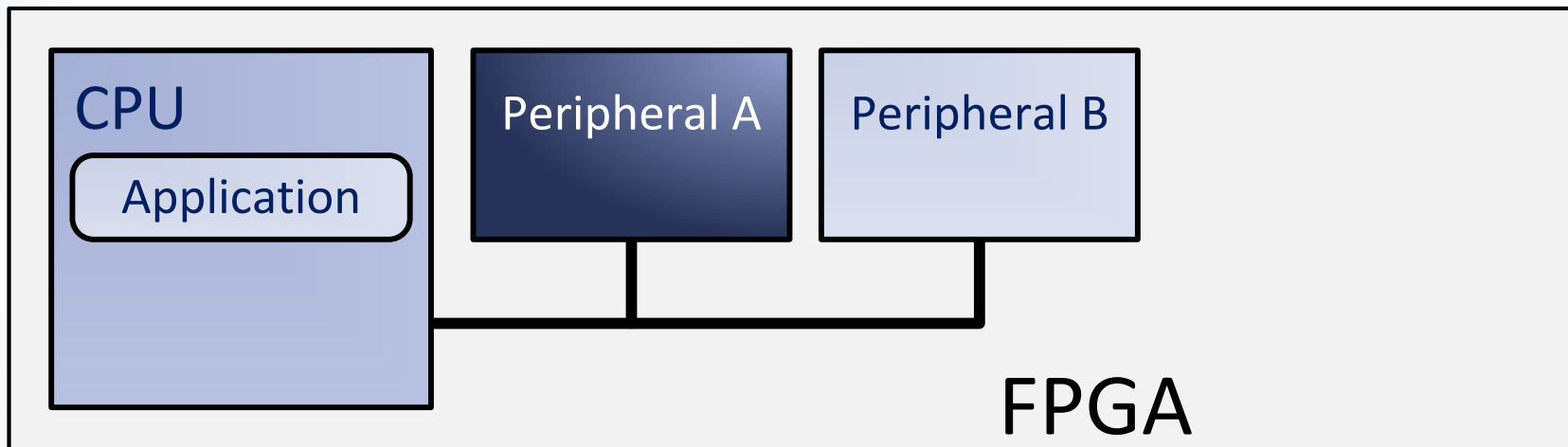
6

FPGA embedded systems



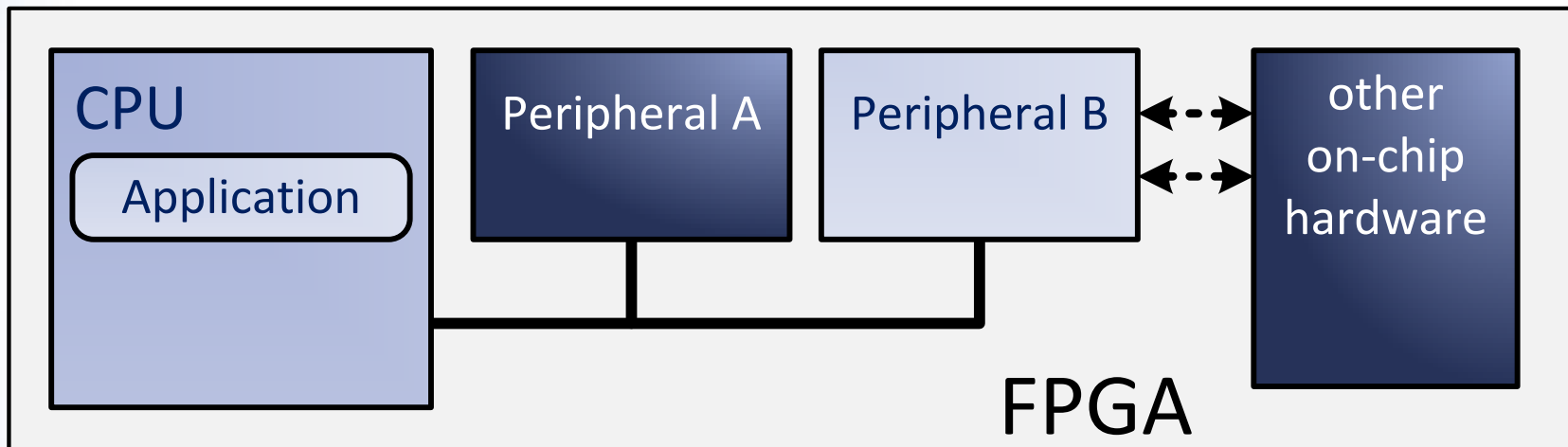
7

FPGA embedded systems



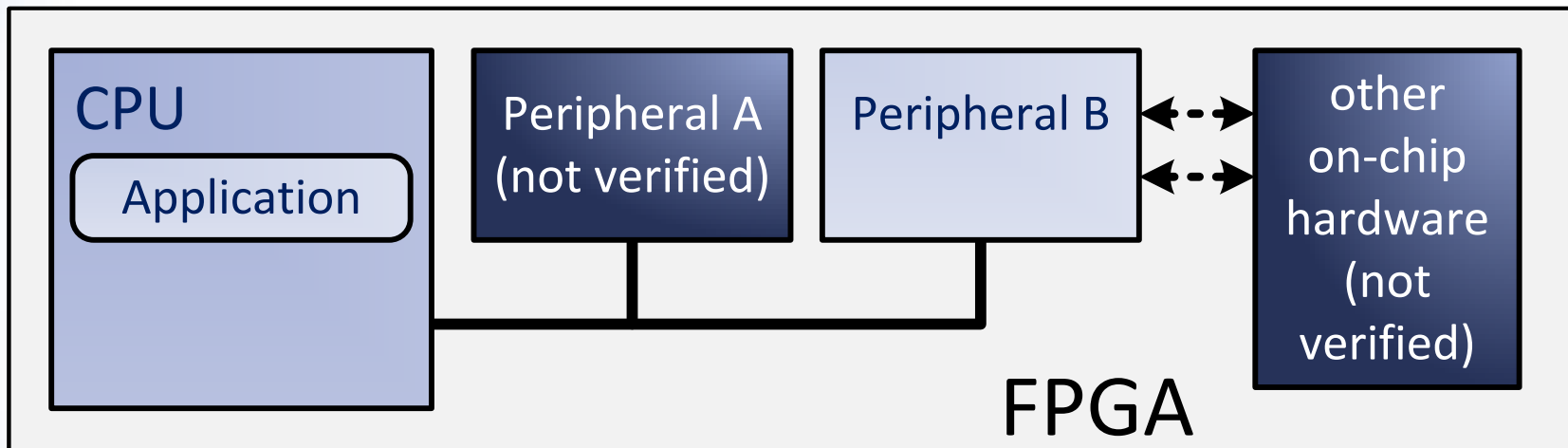
8

FPGA embedded systems

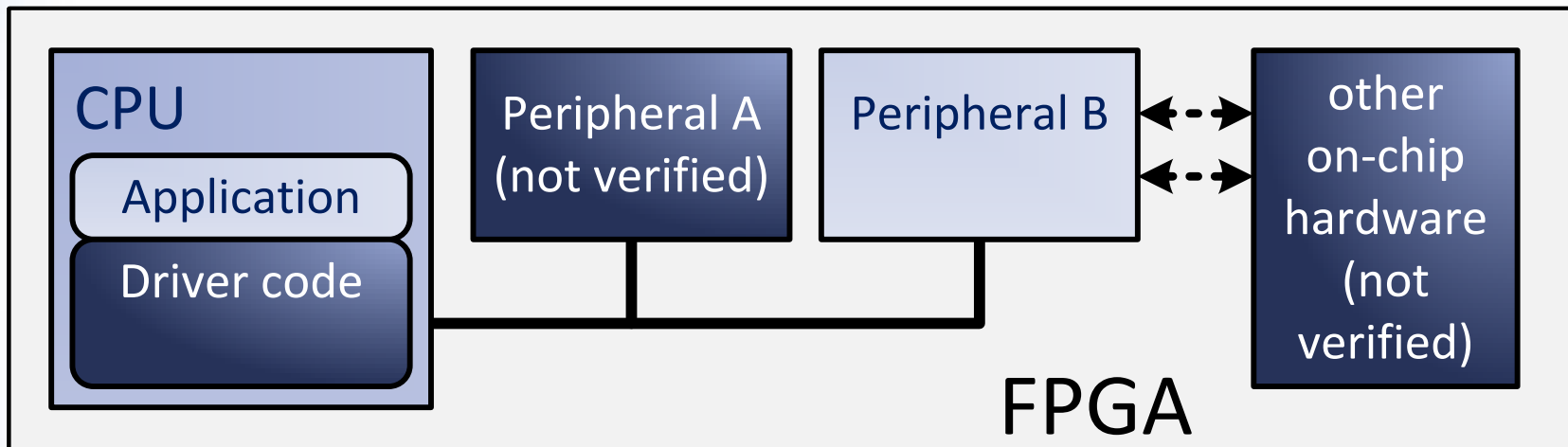


9

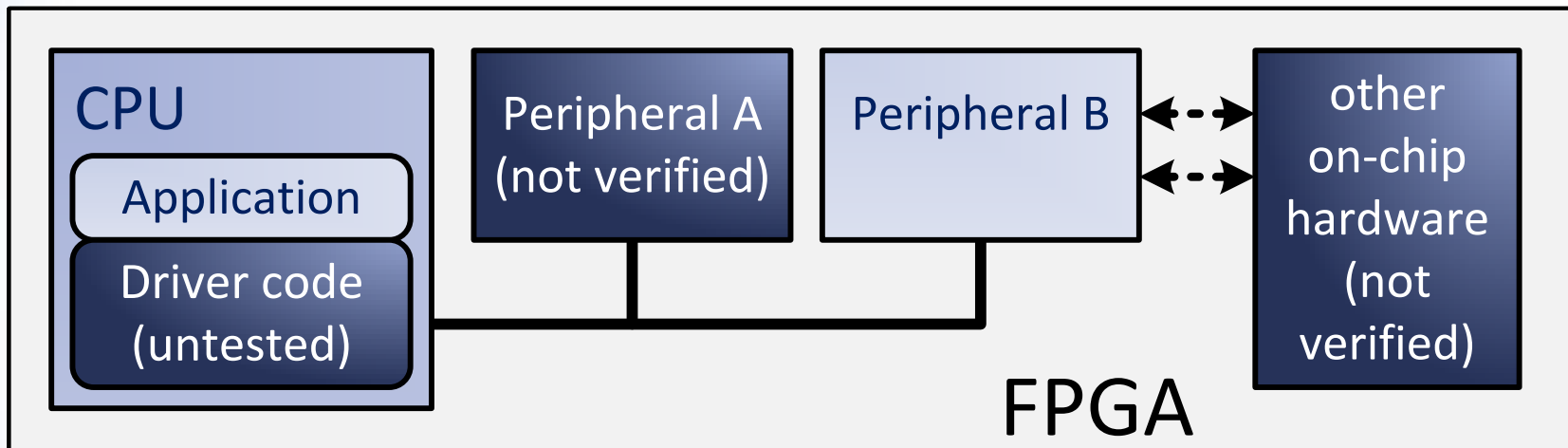
FPGA embedded systems



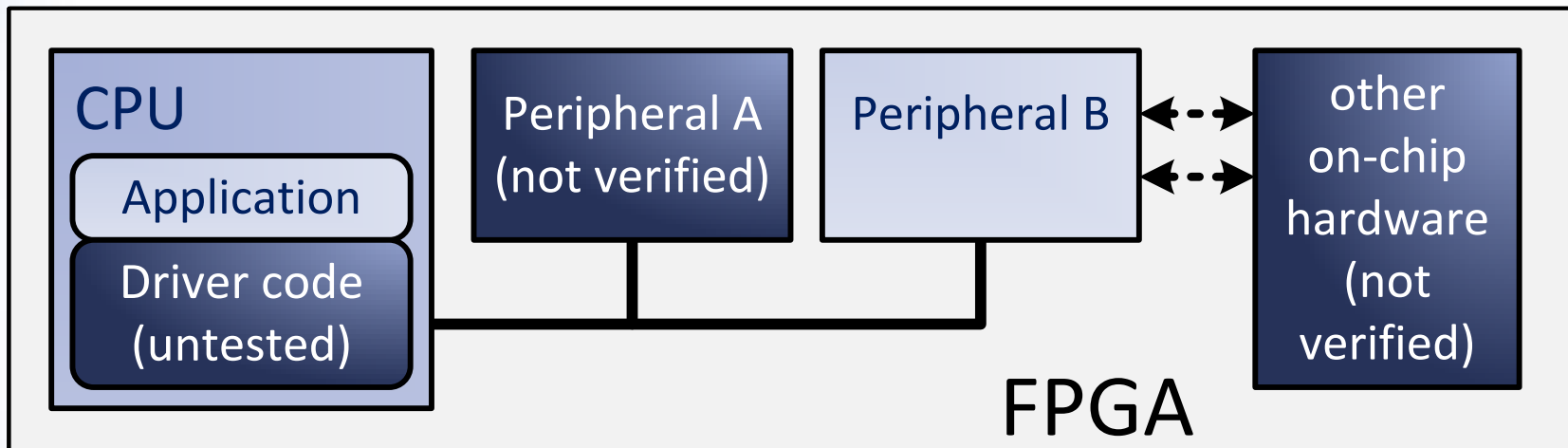
FPGA embedded systems



FPGA embedded systems

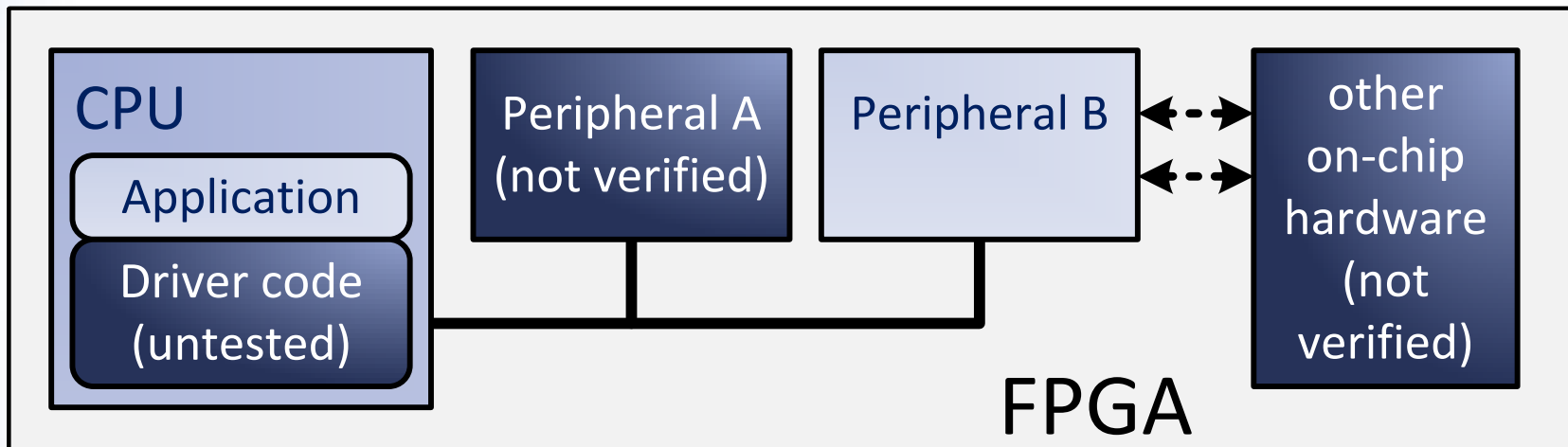


FPGA embedded systems

1
3

Optimal: Debug hardware, software and their
interaction together

FPGA embedded systems



Optimal: Debug hardware, software and their
interaction together

Embedded SW debugging chain



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

The screenshot displays a multi-paneled IDE for embedded system debugging. The top-left pane shows the 'Debug' console with the application 'SimXMD_microblaze_0' and a breakpoint hit in 'main()' at line 49 of 'network_rx_test.c'. The bottom-left pane shows 'Variables' and 'Registers'. The middle-left pane shows 'Memory' with a hex dump. The right pane shows the source code for 'network_rx_test.c'.

Variables:

Name	Value
packet_cnt	5
rx_buffer	0x00008068
buffer_ptr	0x000080b8
t	4

Registers:

Name	Value
r0	0
r1	0x00008040
r2	3842
r3	32952
r4	3
r5	4184
r6	8
r7	57
r8	0
r9	0

Memory (0x8068):

Address	0 - 3	4 - 7	8 - B	C - F
00008060	B8800000	04000000	00000000	2A000000
00008070	2B000000	2C000000	2D000000	00000000
00008080	2E000000	2F000000	30000000	31000000
00008090	00000000	00000000	32000000	33000000
000080A0	35000000	00000000	36000000	37000000
000080B0	38000000	39000000	00000000	00000000
000080C0	00000000	00000000	00000000	00000000
000080D0	00000000	00000000	00000000	00000000
000080E0	00000000	00000000	00000000	00000000
000080F0	00000000	00000000	00000000	00000000

Source Code (network_rx_test.c):

```
#include "xparameters.h"
#include "stdio.h"
#include "fsl.h"
#include "xgpio.h"

#define PACKET_SIZE 4
#define MAX_PACKETS 256

void read_network_packet(unsigned int* packet);
void decrypt_packet(unsigned int key, unsigned int* packet, unsigned int* buffer);

unsigned int cryptokey = 0xDDCCBBAA; // global: not re-initialized on reset
unsigned int rx_packet[PACKET_SIZE];
unsigned int rx_packet_decrypted[PACKET_SIZE];
XGpio LEDs;

int main (void)
{
    //unsigned int cryptokey = 0xDDCCBBAA; // local: re-initialized on reset

    unsigned int packet_cnt = 0;
    unsigned int rx_buffer[PACKET_SIZE * MAX_PACKETS];
    unsigned int *buffer_ptr = rx_buffer;
    unsigned int t;

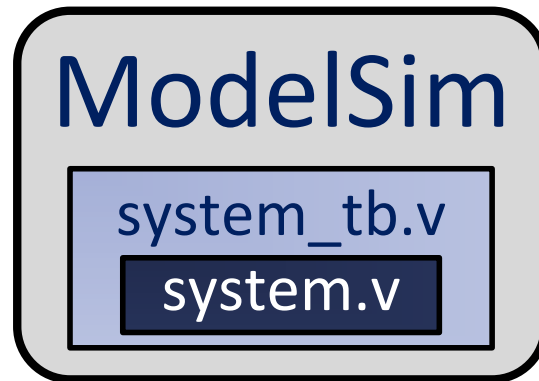
    XGpio_Initialize(&LEDs, XPAR_LEDS_DEVICE_ID);
    XGpio_SetDataDirection(&LEDs, 1, 0x0);

    while(packet_cnt < MAX_PACKETS)
    {
        read_network_packet(rx_packet);
        decrypt_packet(cryptokey, rx_packet, buffer_ptr);
        packet_cnt++;
        buffer_ptr += PACKET_SIZE;
    }
}
```

Console:

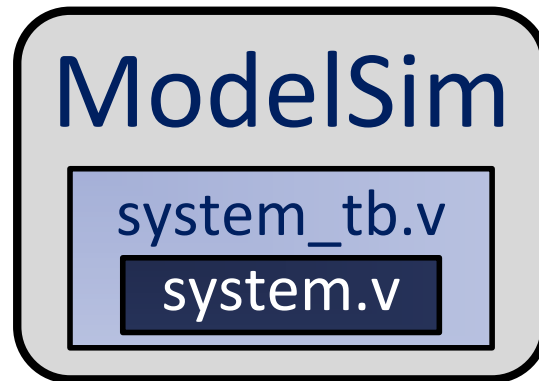
```
C-Build [network_rx_test]
elfcheck network_rx_test.elf -hw ../SimXMD_Demo_v14_hw_platform/system.xml -pe microblaze_0 | tee
"network_rx_test.elf.elfcheck"
elfcheck
Xilinx EDK 14.1 Build EDK_P.15xf
Copyright (c) 1995-2012 Xilinx, Inc. All rights reserved.
Command line: elfcheck -hw ../SimXMD_Demo_v14_hw_platform/system.xml -pe
```

HW debugging



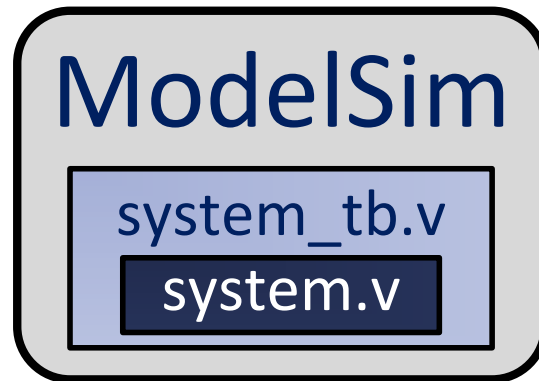
20

HW debugging



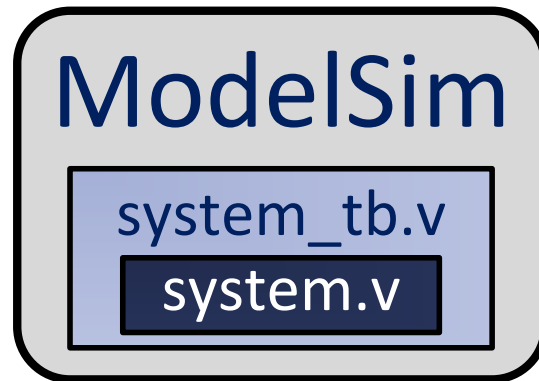
- Cycle-accurate digital simulator

HW debugging



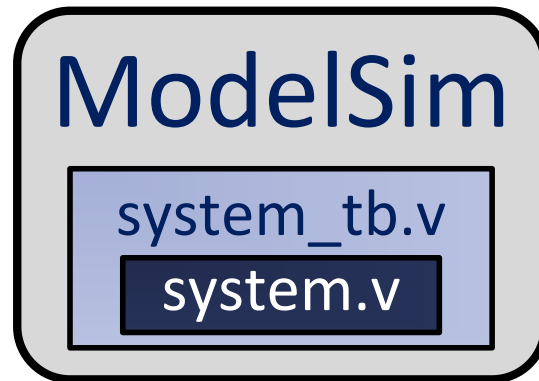
- Cycle-accurate digital simulator
- A system model in HDL

HW debugging



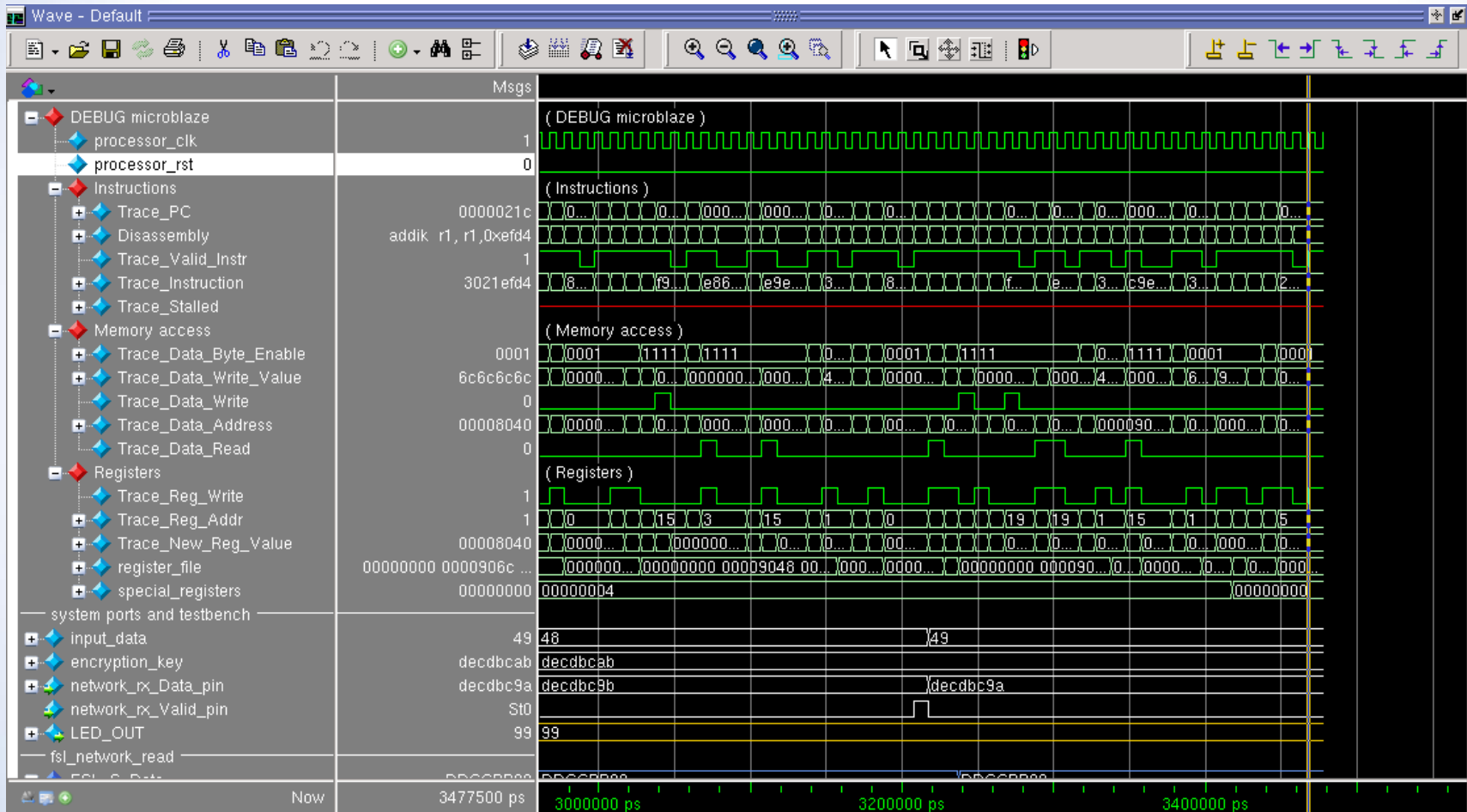
- Cycle-accurate digital simulator
- A system model in HDL
- Testbench instantiates and stimulates model

HW debugging



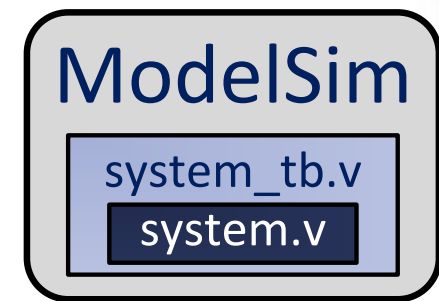
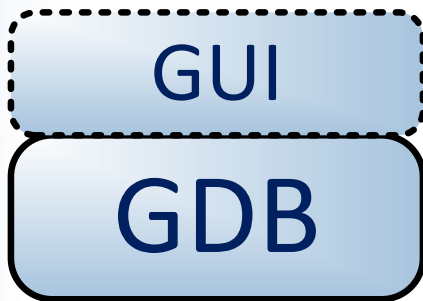
- Cycle-accurate digital simulator
- A system model in HDL
- Testbench instantiates and stimulates model
- All internal signals observable

HW debugging



25

HW/SW Co-Debugging?



26

HW/SW Co-Debugging?

2
7

HW/SW Co-Debugging?



- SimXMD: Simulation-based eXperimental Microprocessor Debugger

HW/SW Co-Debugging?



29

- 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

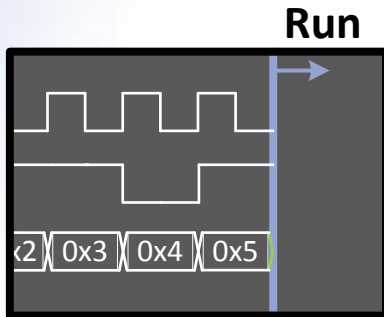
- GDB Remote Serial Protocol
 - Defines requests and replies for:
 - Setting/deleting breakpoints
 - Advancing execution by instruction, line, breakpoint
 - Reading registers or memory state
- Xilinx MicroBlaze Trace Port
 - Reports all information about a finished instruction:
 - Instruction code and 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

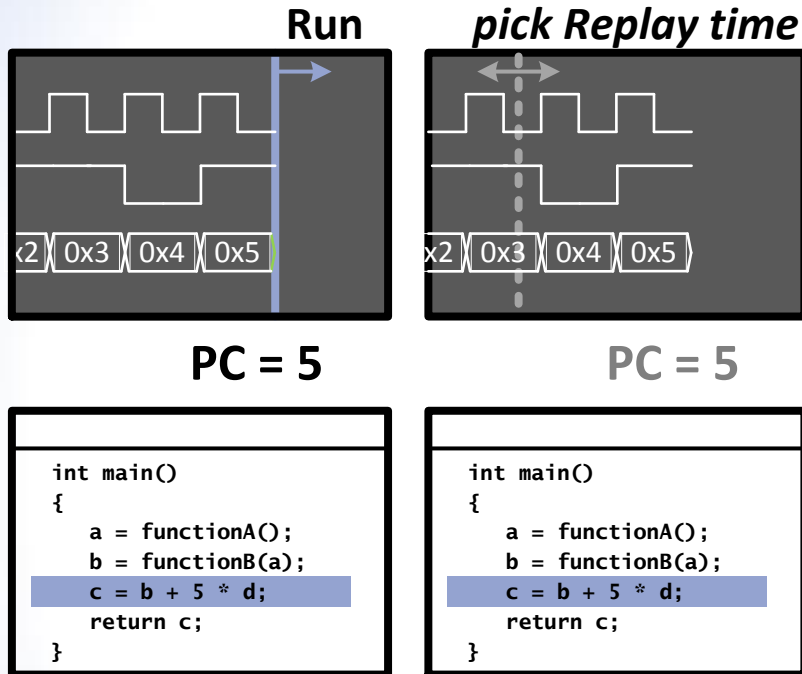


PC = 5

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

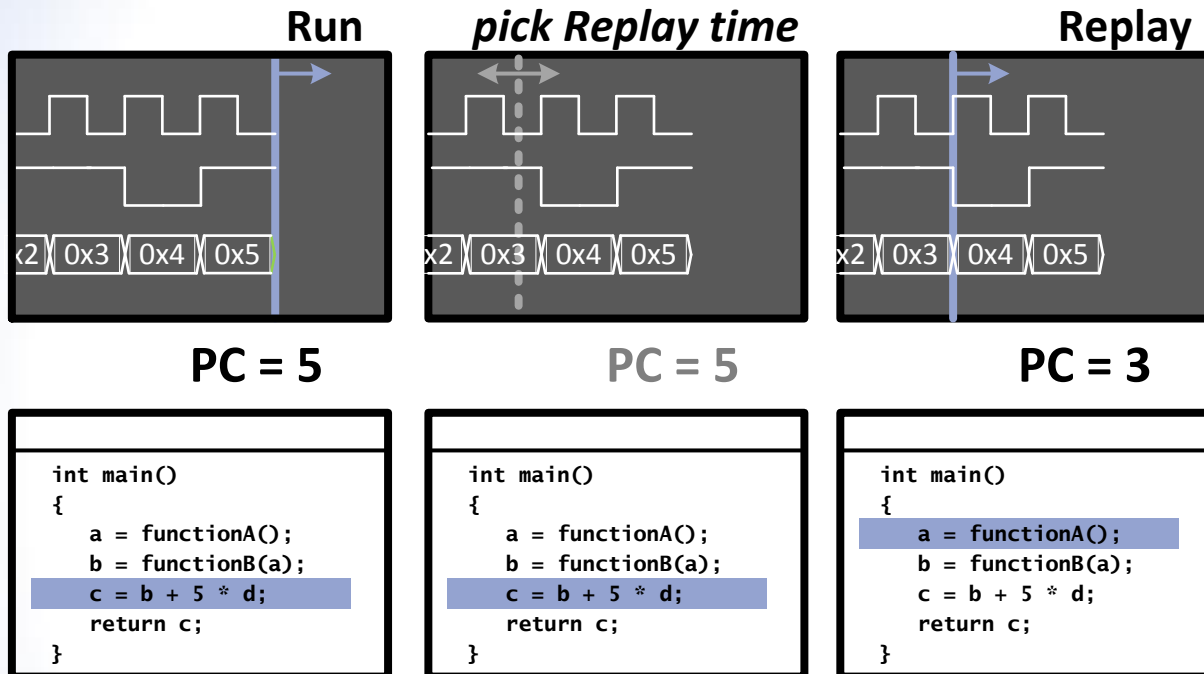
- In *Run mode*, debugging drives the simulation

Operating SimXMD: Modes



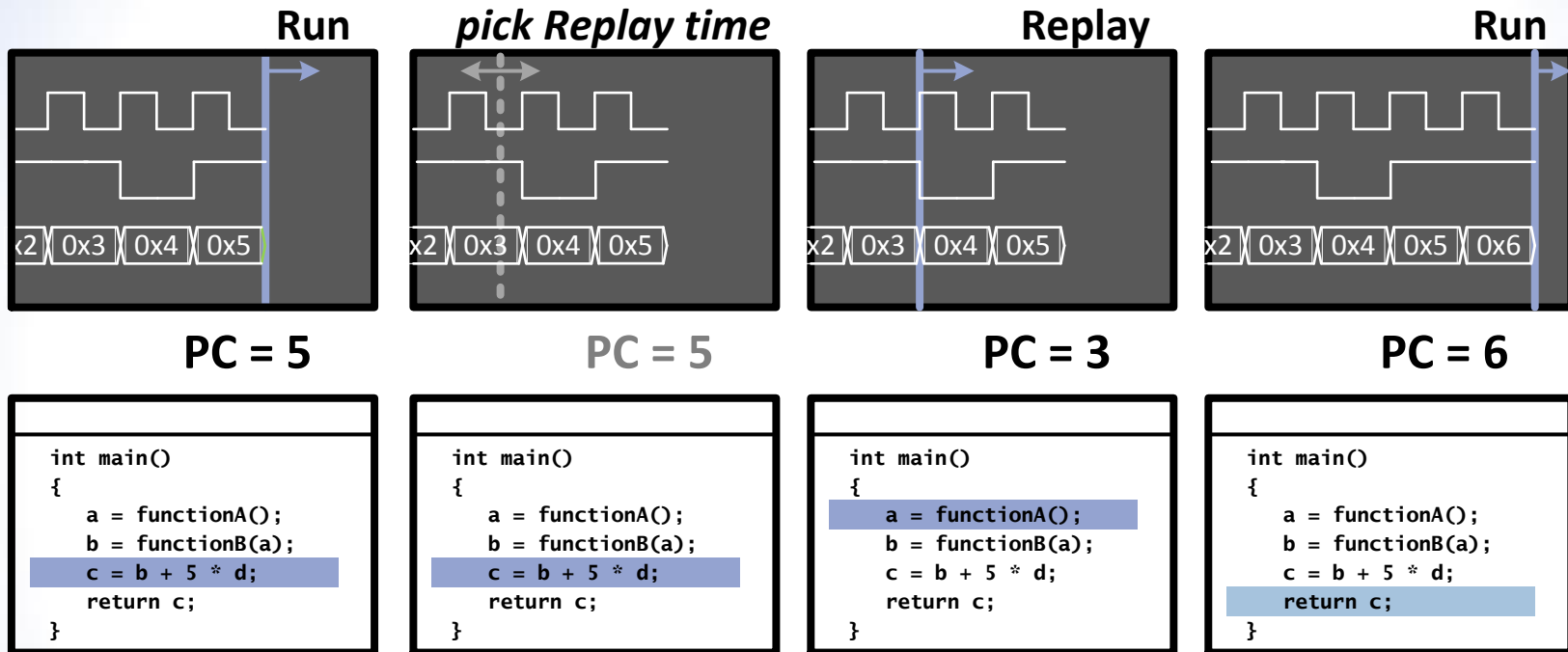
- In *Run mode*, debugging drives the simulation

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

3
8


Debug - network_rx_test/src/network_rx_test.c - Xilinx SDK

Variables

Name	Value
packet_cnt	1
rx_buffer	0x00008068
buffer_ptr	0x00008078
t	0

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

Wave - Default

Msgs

Component	Msgs	Value
DEBUG_microblaze	1	(DEBUG_microblaze)
processor_rst	0	
Instructions		(Instructions)
Trace_PC	00000304	00000300 00000304 00000308
Disassembly	lwi r3,r19,0x0024	swi r4,... lwi r3,...
Trace_Valid_Instr	1	
Trace_Instruction	e8730024	e8840000 f8830000 e8730024
Trace_Stalled		
Memory access		(Memory access)
Registers		(Registers)
system ports and testbench		
input_data	83	82 83
encryption_key	e7d6c5b4	e7d6c5b4
network_rx_Data_pin	e7d6c5e7	e7d6c5e6 e7d6c5e7
network_rx_Valid_pin	St0	
LED_OUT	44	44

Implementation: Debugging memory



3
9



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

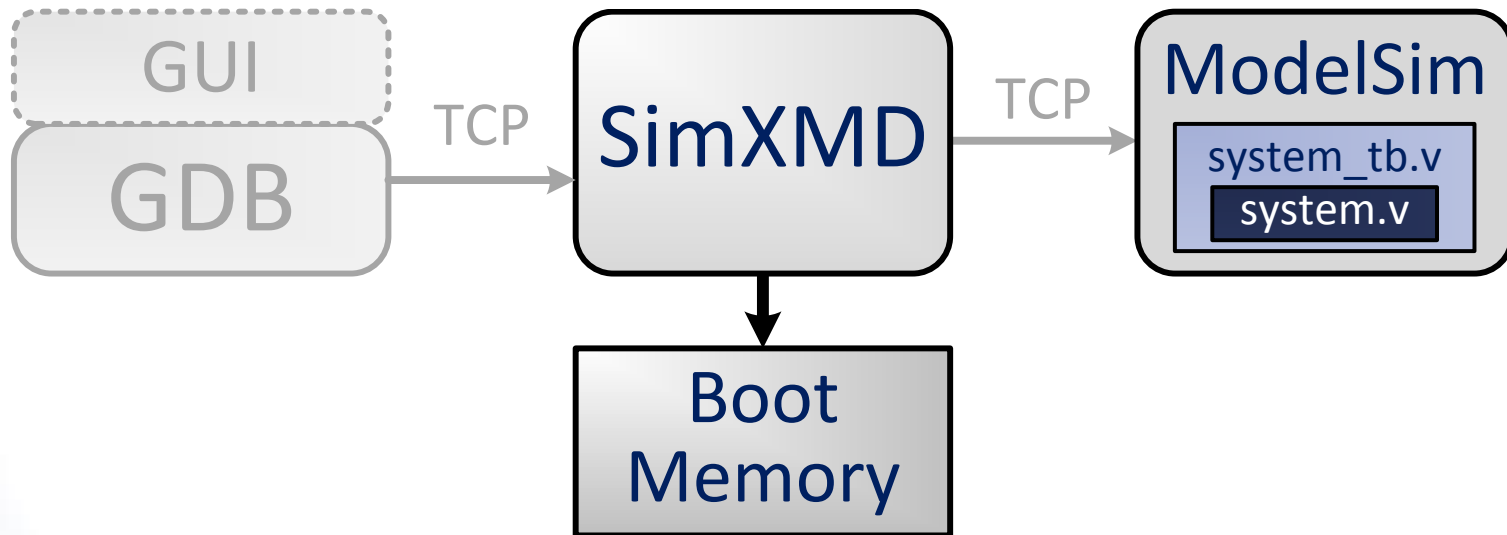
- Digital hardware simulation models the complete memory hierarchy:
 - On-chip and external memory
 - All cache levels
 - Memory-mapped peripherals
- Software debugging uses a flat, linear memory model:
 - The debugger requests a (virtual) memory address
 - The target hardware determines and reads the physical location



SimXMD memory access logging

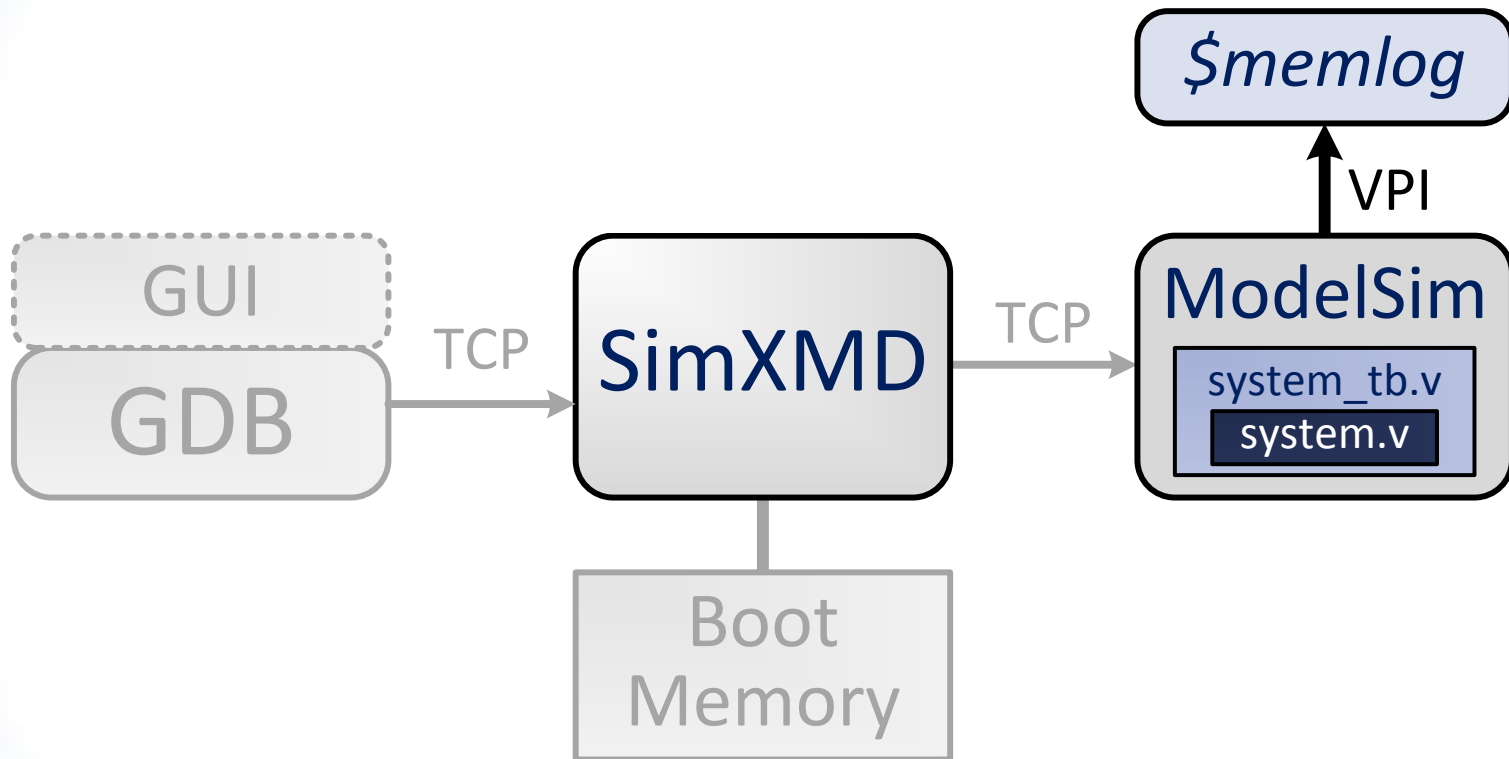


SimXMD memory access logging



43

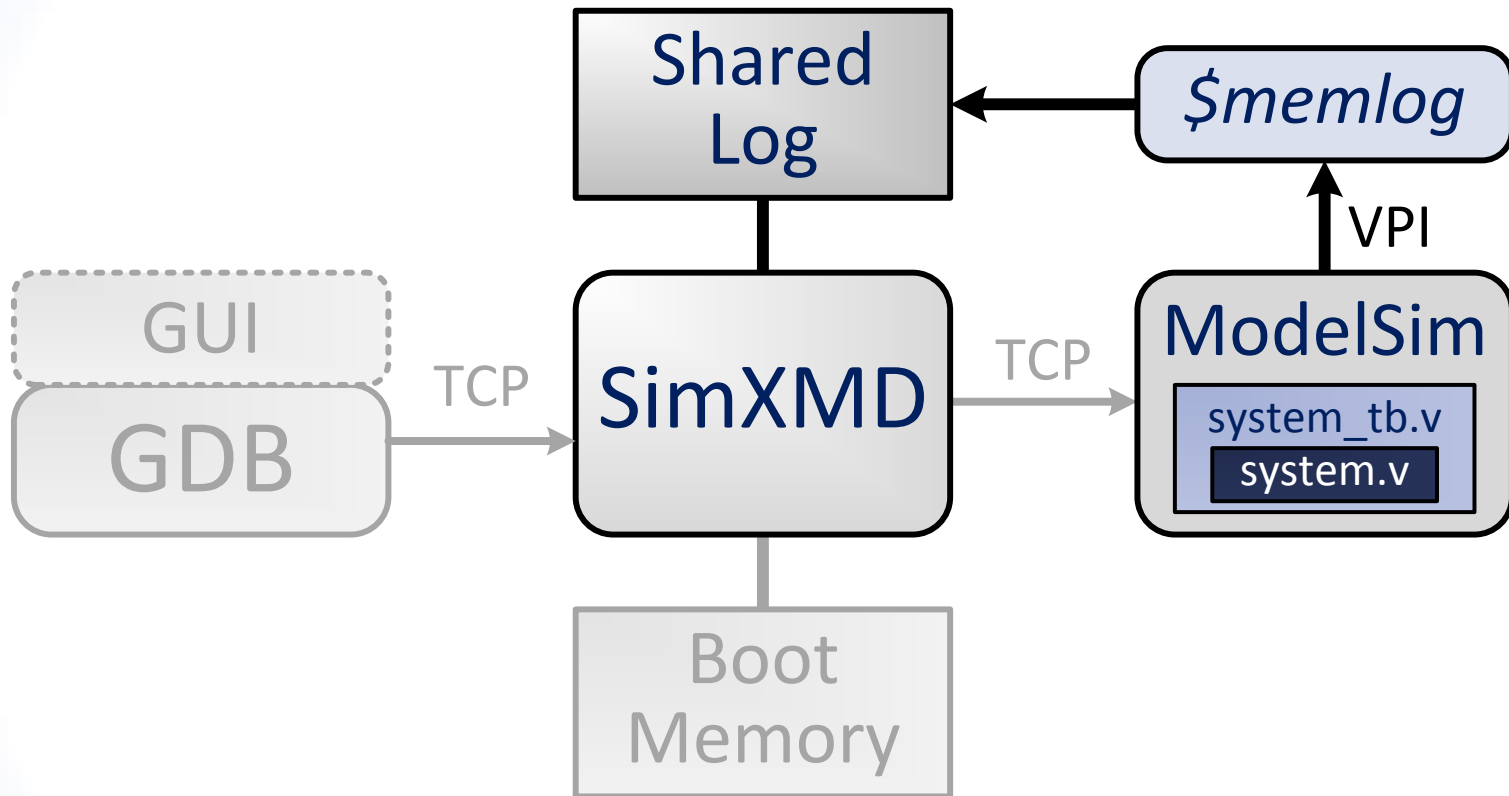
SimXMD memory access logging



44

VPI: Verilog Procedural Interface

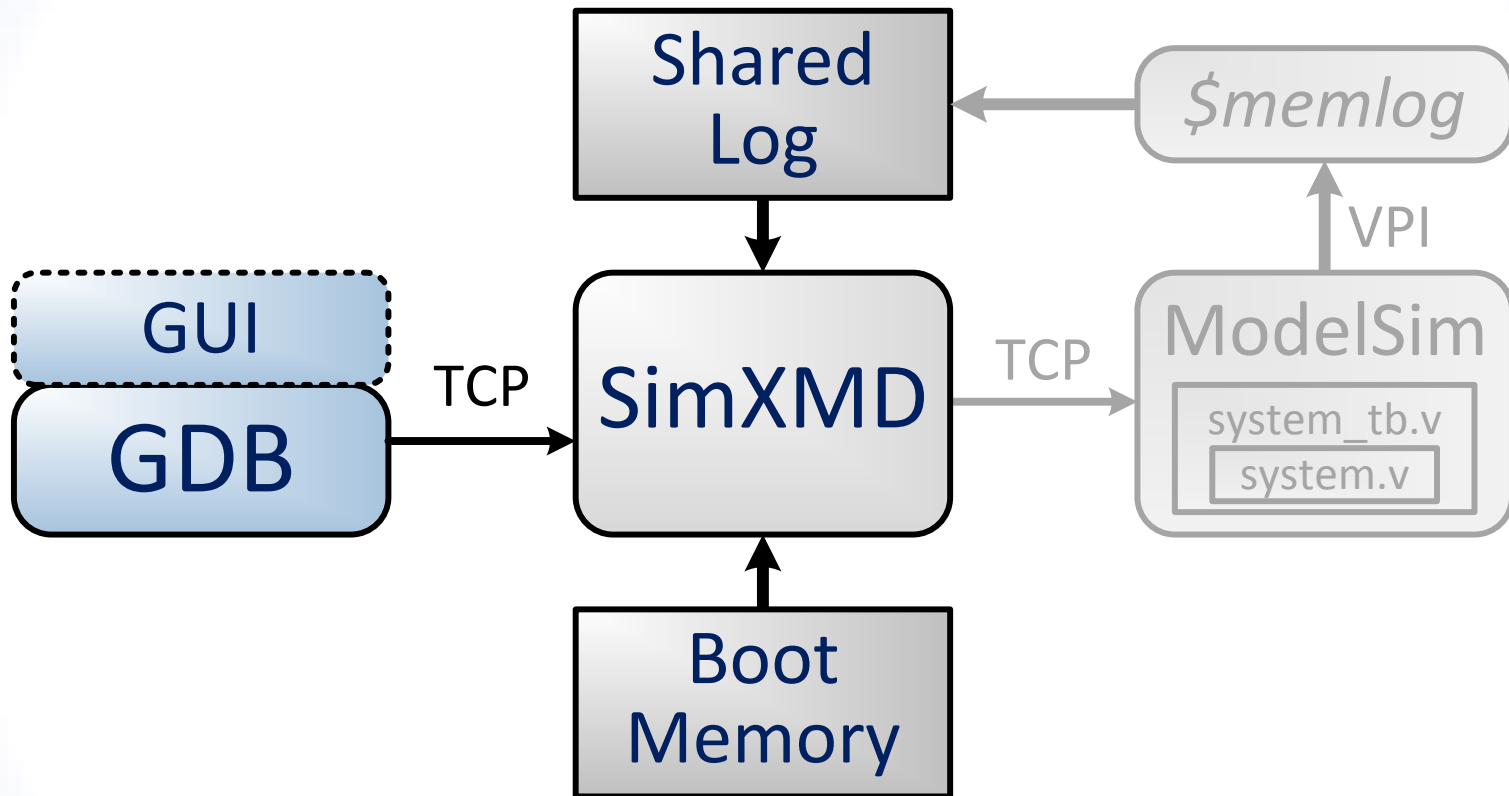
SimXMD memory access logging



45

VPI: Verilog Procedural Interface

SimXMD memory access logging



46

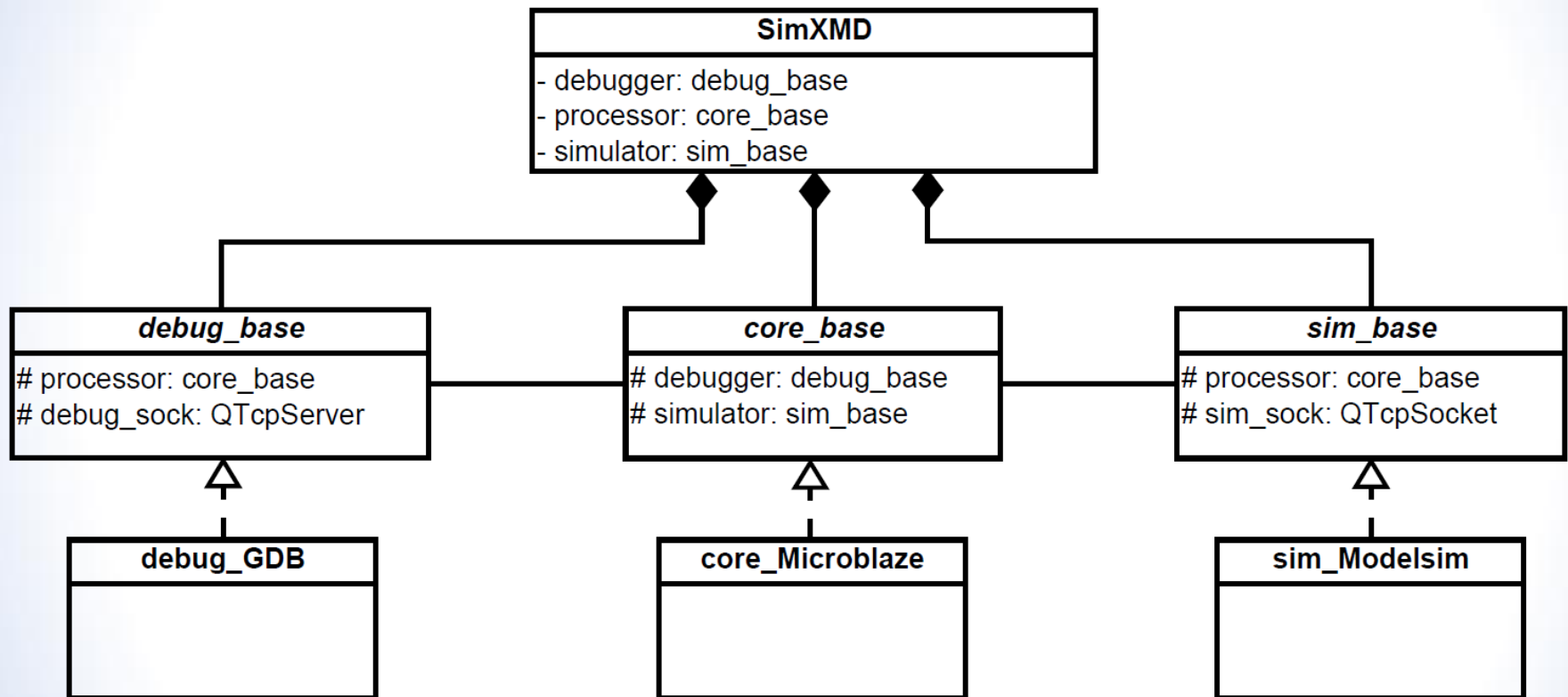
VPI: Verilog Procedural Interface

SimXMD tool support

- Xilinx Embedded Development Kit $\geq 13.x$
 - Xilinx MicroBlaze Processor $\geq 8.x$
- MentorGraphics ModelSim $\geq 6.6g$
- Linux Operating System
- Debuggers
 - Command-line GDB
 - Xilinx SDK (Eclipse)
 - DDD
 - KDbg
 - Nemiver



SimXMD modular architecture



SimXMD limitations



4
9



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 multiprocessing
 - 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?



55



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

Write size	w/o SimXMD	w/ SimXMD
1 kByte	6.9 s	7.3 s
2 kByte	13.8 s	14.5 s
4 kByte	27.3 s	29.0 s
8 kByte	54.9 s	57.7 s
16 kByte	109.0 s	117.1 s
32 kByte	218.9 s	231.7 s



SimXMD overhead

Write size	w/o SimXMD	w/ SimXMD
1 kByte	6.9 s	7.3 s
2 kByte	13.8 s	14.5 s
4 kByte	27.3 s	29.0 s
8 kByte	54.9 s	57.7 s
16 kByte	109.0 s	117.1 s
32 kByte	218.9 s	231.7 s

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:

Hardware with JTAG	1.350 s
SimXMD Run mode	0.850 s
SimXMD Replay mode	0.313 s

Conclusions



6
5



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?