EMBECOSM®

Open Source Software Meets Open Source Hardware OpenCores and the OpenRISC 1000

Jeremy Bennett, CEO Embecosm
Julius Baxter, OpenRISC Program Manager, OpenCores.org

17 October 2011

Agenda

- About OpenCores
  - licensing
- OpenRISC 1000 architecture
  - OpenRISC 1200 implementation
  - verification
- OpenRISC 1000 tool chain
  - tools, libraries and operating systems
  - software development on OpenRISC
- 20 minute break (refreshments outside)
- Hands-on session
  - installing the software
  - bare metal and Linux
- Find out more...

About OpenCores

Overview of OpenCores

- 131,382 registered users reported as of 17 October 2011
- 870 projects as of 17 October 2011

Founded by Damjan Lampret

OpenRISC 1000 developed

OpenRISC 1000 commercial deployment

1999

2007

2011

Primary support Flextronics

Owned by ORSoC AB

Web: **www.opencores.org** and **www.openrisc.net**
IRC: **freenode.net**, channel **#opencores**

Licensing Open Source Hardware

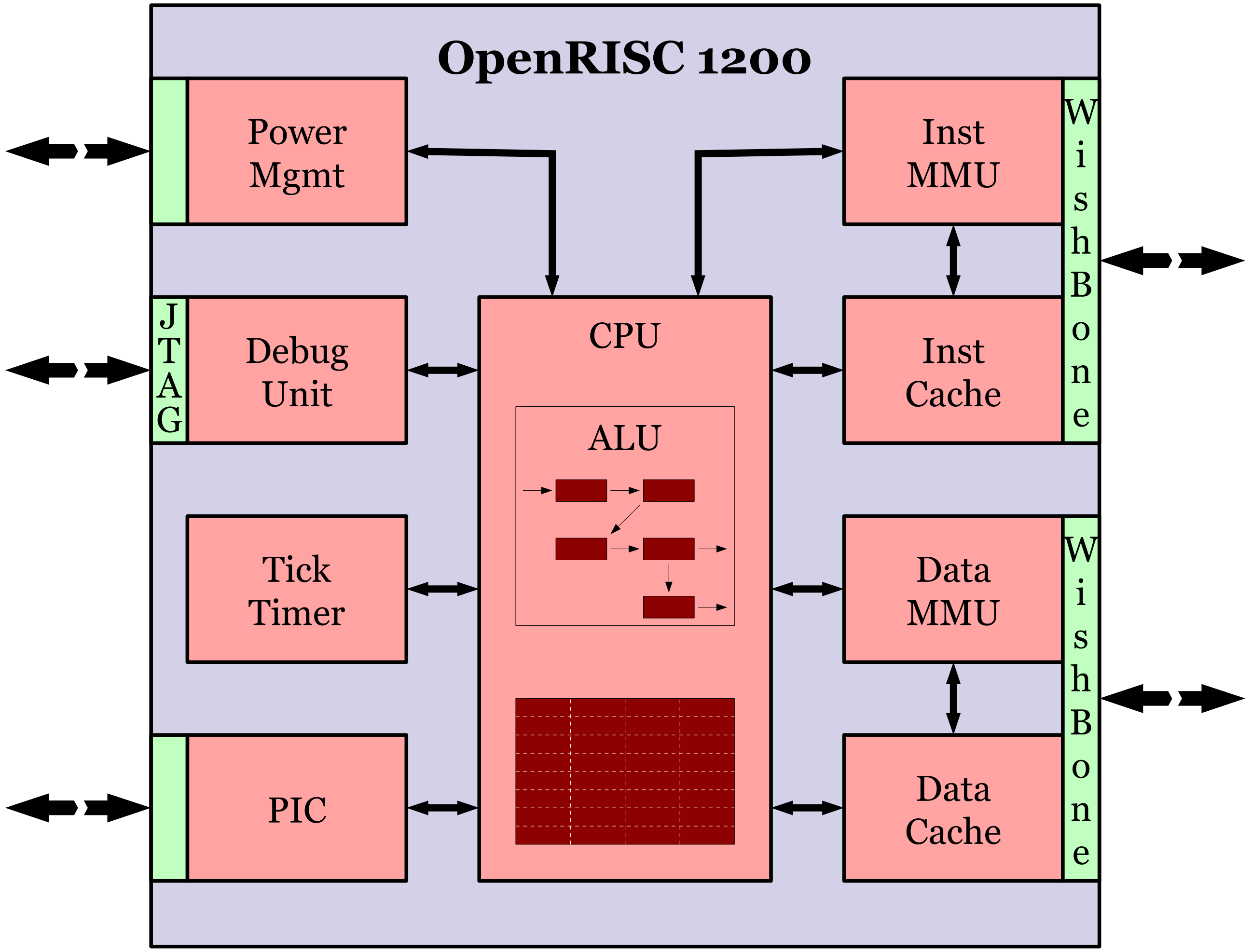
- Most OpenCores hardware is LGPL or BSD
  - software tools GPL, documentation also GPL (!)
- What does LGPL mean for hardware
  - open source obligations apply only to this hardware block
  - what is the hardware equivalent of linking?
- OSS licenses don't really work
  - consider:
    - I take an LGPL hardware design and modify it
    - I manufacture a chip from that design
    - I sell you one of those chips
  - Am I then obliged to give you a copy of the modified Verilog I used?
- There are alternatives based on contract law
  - best known are the TAPR OHL and CERN OHL
  - tend to be jurisdiction specific
- This is a major barrier to open source hardware growth

The OpenRISC 1000 Architecture

The OpenRISC 1000 Project

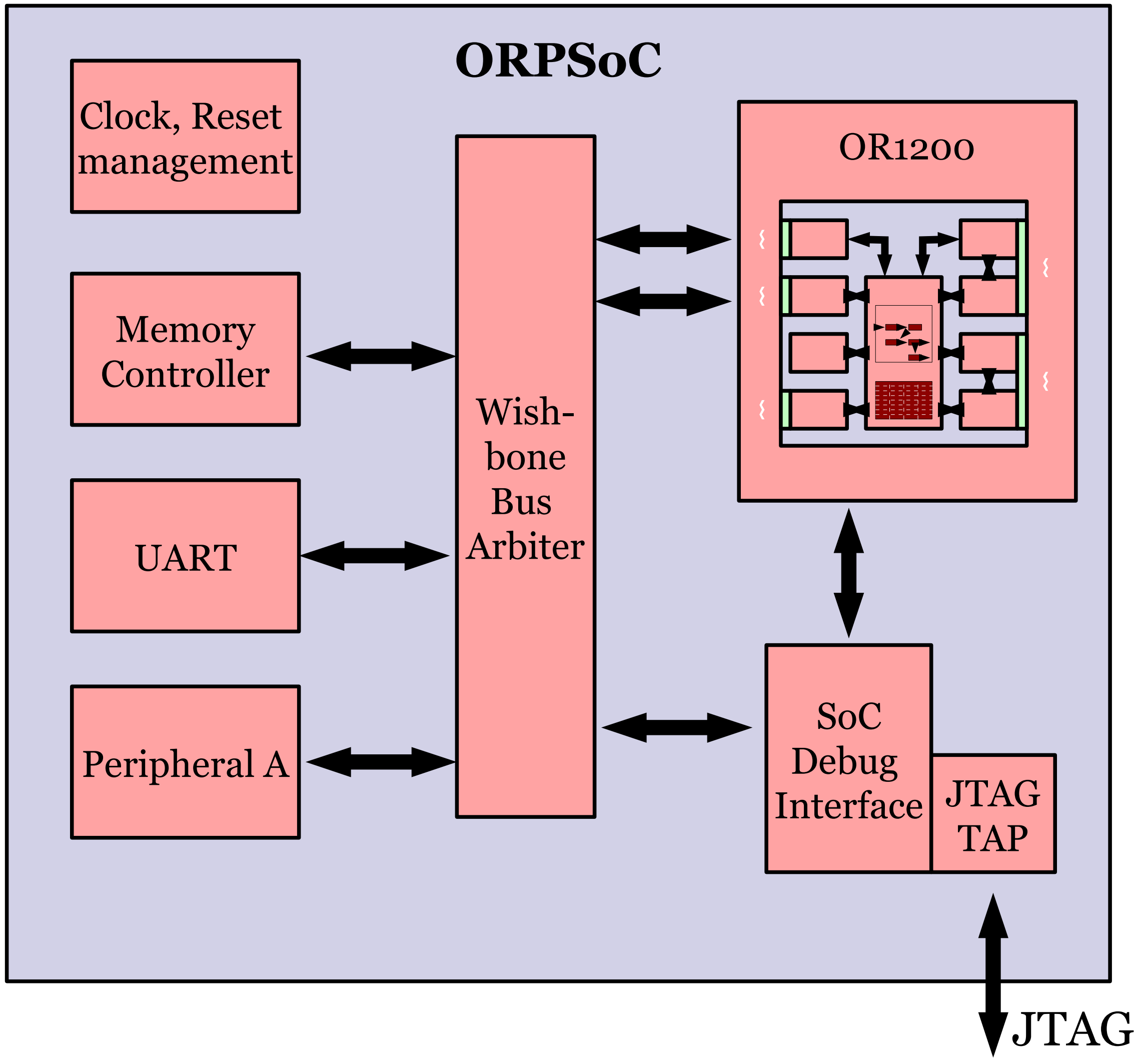
- Objective to develop a family of open source RISC designs
  - 32 and 64-bit architectures
  - floating point support
  - vector operation support
- Key features
  - fully free and open source
  - linear address space
  - register-to-register ALU operations
  - two addressing modes
  - delayed branches
  - Harvard or Stanford memory MMU/cache architecture
  - fast context switch
- Looks rather like MIPS or DLX

The OpenRISC 1200



- 32-bit Harvard RISC architecture
  - MIPS/DLX like instruction set
  - first in OpenRISC 1000 family
  - originally developed 1999-2001
- Open source under the
  - GNU Lesser General Public License
  - allows reuse as a component
- Configurable design
  - caches and MMUs optional
  - core instruction set
- Source code Verilog 2001
  - approx 32k lines of code
- Full GNU tool chain and Linux port
  - various RTOS ported as well

ORPSoC: OpenRISC Reference Platform System-on-Chip



- Combined reference implementation and board adaptations
- Reference implementation – minimal SoC for processor testing, development
  - compilable into cycle-accurate model
- Boards ports target multiple technologies
- Lowers barrier to entry for OpenRISC-based SoC design
  - Push-button compile flow
  - Largely utilises open-source EDA tools

Hardware Development

- Objective is to use an open source EDA tool chain
  - back end tools for FGPA all proprietary
    - free (as in beer) versions available
  - front end tools now have open source alternatives
- OpenRISC 1000 simulation models
  - Or1ksim: golden reference ISS
    - C/SystemC interpreting ISS, 2-5 MIPS
  - Verilator cycle accurate model from the Verilog RTL
    - 130kHz in C++ or SystemC
  - Icarus Verilog event driven simulation
    - 1.4kHz, 50x slower than commercial alternatives
- All OpenRISC 1000 simulation models suitable for SW use
  - all support GDB debug interface

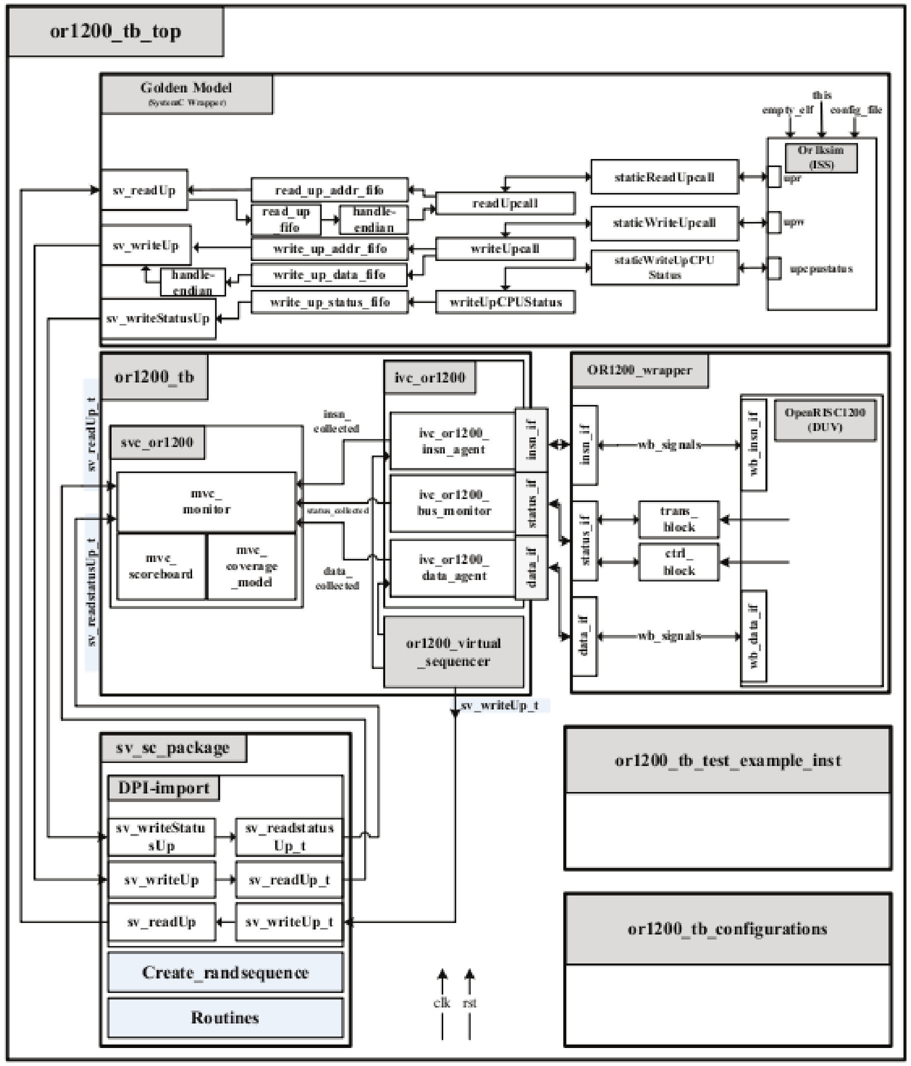
OR1200 Verification 1999-2001

- Test bench and DUT simulated using *Icarus Verilog*
  - Open source event driven simulator
  - Approximately 20-50 times slower than NC/VCS/ModelSim
  - 1.4kHz on a basic Core2 Duo
- Test by running programs in C and assembler
  - 13 different test programs
- Limitations
  - not exhaustive
  - no coverage metrics
  - not consistent with the golden reference model

Needs Improvement

OVM Verification of the OR1200

- MSc by Waqas Ahmed, KTH, Stockholm
- Criteria
  - does PC update correctly?
  - does status update correctly?
  - does exception save context correctly?
  - is data stored to the correct memory address?
  - are results stored correctly in registers?
- Compare behavior of RTL against Or1ksim
  - constrained random generation of scenarios
- Provide SystemVerilog wrappers
  - for Or1ksim use DPI to SystemC interface
- Results:
  - `l.jr`/`l.jalr` to unaligned address
  - `l.addic` and `l.lws` missing in DUV
  - `l.macrc`/`l.ror`/`l.rori` missing from Or1ksim
  - 20 instructions had errors
  - most coverage targets achieved



More Hardware Verification

There is another way...

(to be continued later)

The OpenRISC 1000 Tool Chain

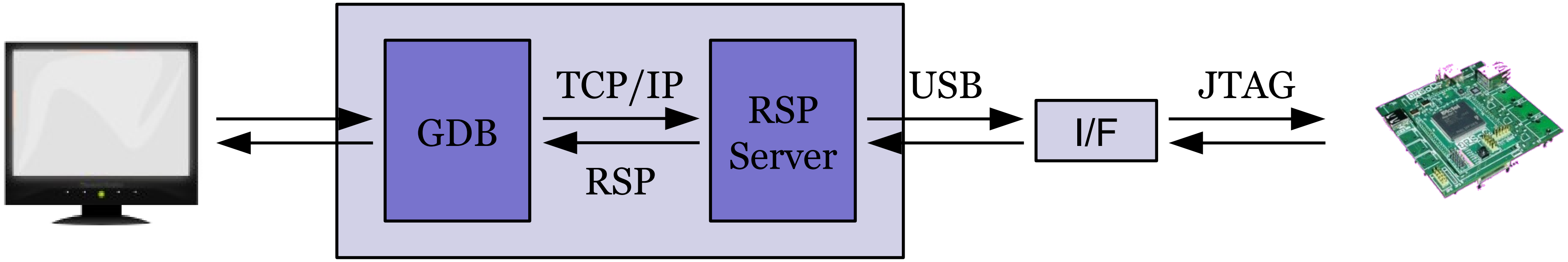
The Software Tool Chain

- A standard GNU tool chain
  - binutils 2.20.1
  - gcc 4.5.1
  - gdb 7.3 (for BCS use only!)
  - C and C++ language support
- Library support
  - static libraries only
  - newlib 1.18.0 for bare metal (**`or32-elf-*`**)
  - uClibc 0.9.32 for Linux applications (**`or32-linux-*`**)
- Testing
  - regression tested using Or1ksim (both tool chains)
  - **`or32-linux-*`** regression tested on hardware
  - **`or32-elf-*`** regression tested on a Verilator model

Board and OS Support

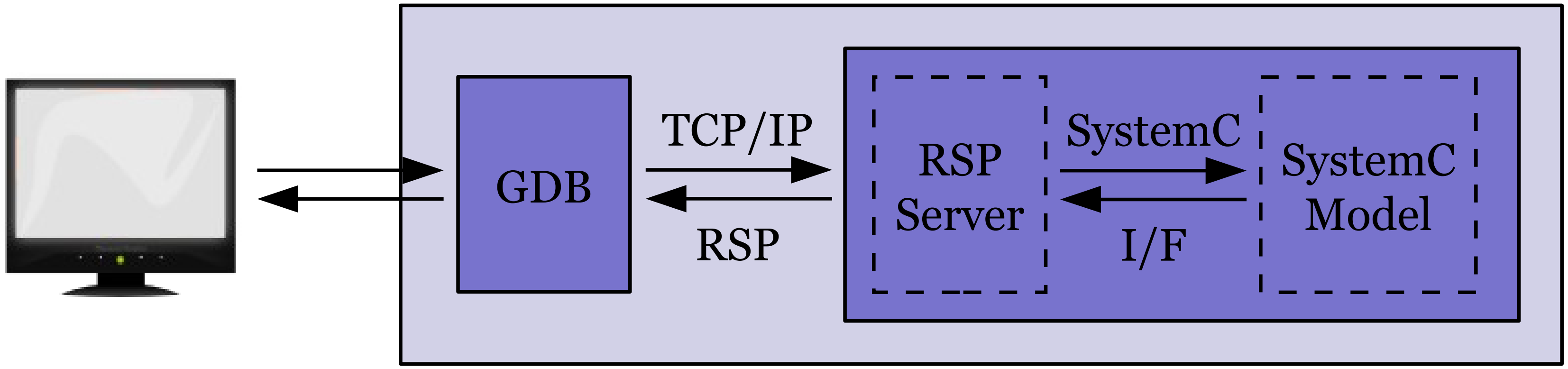
- Boards with BSP implementations
  - Or1ksim
  - DE-nano
  - Terasic DE-2
- RTOS support
  - FreeRTOS, RTEMS and eCos all ported
- Linux support
  - adopted into Linux 3.1 kernel mainline
  - some limitations (kernel debug, ptrace)
  - BusyBox as application environment
- Debug interfaces
  - JTAG for bare metal
  - *gdbserver* over Ethernet for Linux applications

Software Development Remote Connection to GDB



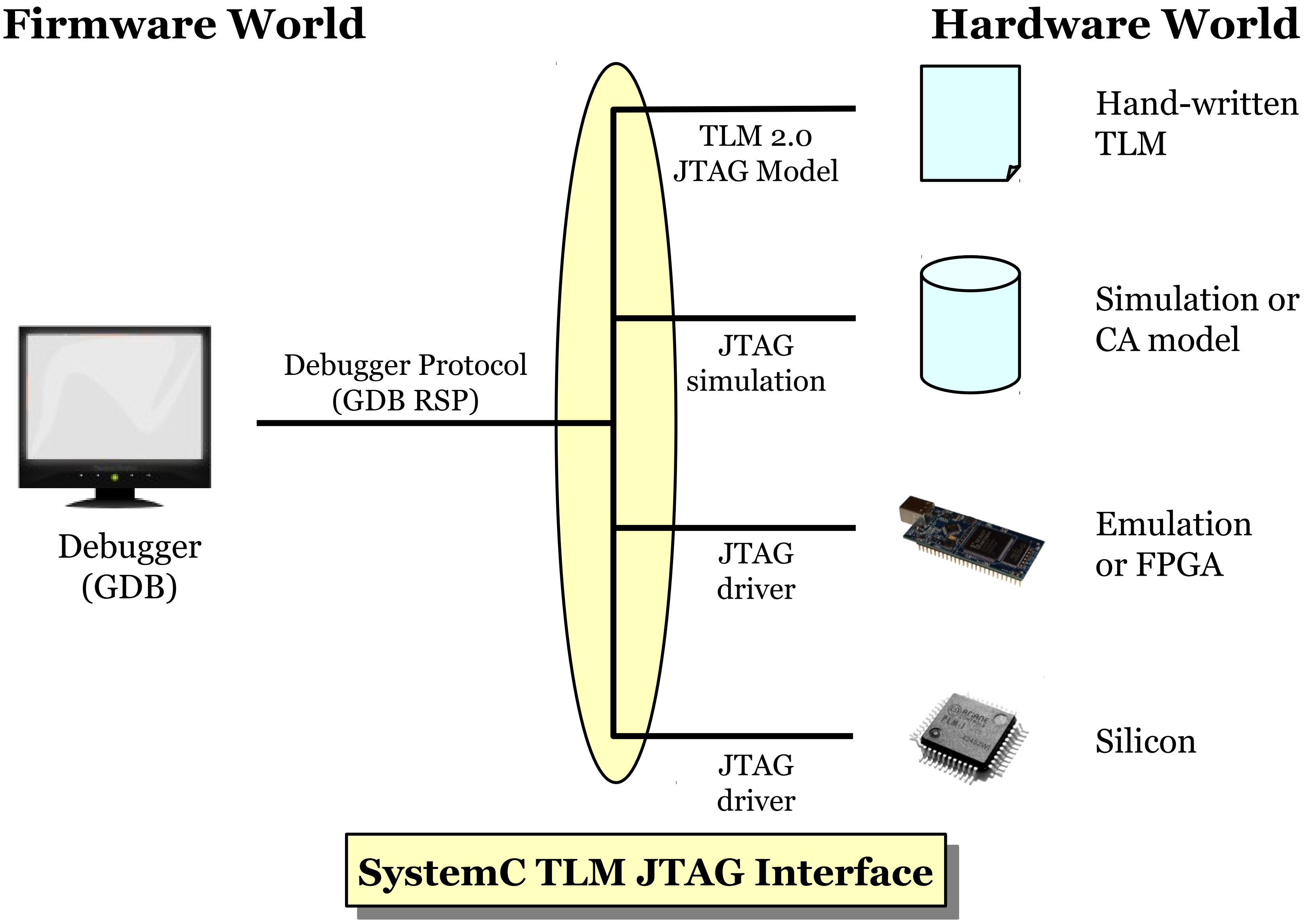
```
(gdb) target remote :51000
```

Remote Connection to SystemC



```
(gdb) target remote :51000
```

Unified SystemC Debug



More Hardware Verification

There is another way...

(continued)

Golden SystemC TLM Model

```
                === gcc Summary ===

# of expected passes            52753
# of unexpected failures        152
# of expected failures          77
# of unresolved testcases       122
# of unsupported tests          716
```

Comparative Regression Testing of the OpenRISC 1200

Golden SystemC TLM Model

```
                    === gcc Summary ===

# of expected passes                52753
# of unexpected failures            152
# of expected failures              77
# of unresolved testcases           122
# of unsupported tests              716
```

Verilator SystemC RTL Model

```
                    === gcc Summary ===

# of expected passes                52677
# of unexpected failures            228
# of expected failures              77
# of unresolved testcases           122
# of unsupported tests              716
```

Comparative Regression Testing of the OpenRISC 1200

Golden SystemC TLM Model

```
                === gcc Summary ===

# of expected passes            52753
# of unexpected failures        152
# of expected failures          77
# of unresolved testcases       122
# of unsupported tests          716
```

Verilator SystemC RTL Model

```
                === gcc Summary ===

# of expected passes            52677
# of unexpected failures        228
# of expected failures          77
# of unresolved testcases       122
# of unsupported tests          716
```

- We can identify two types of problem
  - tests which fail due to timing out with RTL, but not due to slower model
  - tests which give a different result with RTL
- These are candidates for possible RTL errors
- Used commercially by Adapteva Inc
  - 50-60 RTL errors eliminated pre-tape out

Test Failure Examples

- `gcc.c-torture/execute/20011008-3.c execution,  -O0`
  - this test just needs more time
- `gcc.c-torture/execute/20020402-3.c execution,  -Os`
  - this test doesn't complete in 2 hours
  - sibling tests take 200k cycles
  - BUG???
- `gcc.c-torture/execute/20090113-1.c execution,  -O2`
  - RTL test fails with bus error exception
  - BUG???

Practical Demonstration

Practical Demonstrations

- Three strands
- Using OpenRISC bare metal on an FPGA
  - Jeremy Bennett
- Using OpenRISC with Linux on an FPGA
  - Julius Baxter
- Using OpenRISC on Or1ksim
  - Everyone!
  - both native and Linux

Installation: Preparation

- DVD/USB contains source code for
  - Verilog RTL and testbench for OpenRISC 1200 and ORPSoC
  - Or1ksim, the architectural simulator
  - The GNU tool chain (binutils, gcc and gdb)
  - *newlib* and *uClibc* libraries
  - Linux kernel and BusyBox
  - Assorted scripts
  - Example program (**`hello.c`**)
- Ensure you have the following installed
  - native GNU tool chain (usually standard)
  - install the following packages if not already present

```
sudo yum install gmp-devel
sudo yum install mpfr-devel
sudo yum install libmpc-devel
```

Installation: GNU Tool Chains

- Preparation
  - copy everything from the DVD/USB to your local disc
  - untar the packages in the **`toolchain`** directory
  - decide where to install Or1ksim and the GNU toolchain
  - default **`/opt/or1ksim-new`** and **`/opt/or32-new`**
- Build or1ksim and GNU tool chains

```
./bld-or1ksim.sh --prefix /opt/or1ksim-new
./bld-all.sh --prefix /opt/or32-new \
             --or1ksim-dir /opt/or1ksim-new
```

- Add tools to the PATH

```
export PATH=/opt/or32-new/bin:/opt/or1ksim-new/bin:$PATH
```

  - or

```
setenv PATH /opt/or32-new/bin:/opt/or1ksim-new/bin:$PATH
```

- You may also need

```
sudo ln -s /lib/libncurses.so.5.7 /lib/libtinfo.so.5
```

Installation: Linux and BusyBox

- Unpack the linux initramfs

```
cd linux-2.6.39
tar jxf initramfs.tar.bz2
cd ..
```

- Configure BusyBox

```
cd busybox-1.17.3
make menuconfig
```

  - under BusyBox Settings → Build Options
    - Build BusyBox as static binary setting: **`[*] (Y)`**
    - Cross Compiler prefix setting: **`or32-linux-`**
  - under BusyBox Settings → Installation Options
    - BusyBox installation prefix setting: *installation_root*`/linux-2.6.39/arch/openrisc/support/initramfs`

```
cd ..
```

- Build Linux and BusyBox

```
./bld-bb.sh
```

Bare metal example

- Build the example

```
cd examples
or32-elf-gcc hello.c -o hello
```

- Run the example

```
or32-elf-sim --memory=8M hello
```

- Run the example using GDB. In first window

```
or32-elf-sim --memory=8M --srv=51000
```

- In second window

```
cd examples
or32-elf-gdb hello
(gdb) target remote :51000
(gdb) load
(gdb) break main
(gdb) continue
(gdb) backtrace
(gdb) continue
```

Bare metal debugging

- Simulation as remote debugging target
  - Run the example using GDB. In first window

```
or32-elf-sim --memory=8M --srv=51000
```

  - In second window

```
cd examples
or32-elf-gdb hello
(gdb) target remote :51000
(gdb) load
(gdb) break main
(gdb) continue
(gdb) backtrace
(gdb) continue
```

- Using simulator built into GDB

```
or32-elf-gdb hello
(gdb) target sim
(gdb) load
(gdb) etc…
```

Running Linux

- Edit arch/openrisc/or1ksim.cfg
  - find **`section ethernet`**
  - set **`enabled`** to **`0`**
  - comment out the **`sockif`** line
- Run kernel and BusyBox

```
cd linux-2.6.39
or32-elf-sim -f arch/openrisc/or1ksim.cfg vmlinux
```

- Run kernel with Ethernet
  - use TUN/TAP to create a virtual IP interface, which Or1ksim can use
  - requires typically *openvpn* and/or *bridge-utils* installed
  - scripts **`brstart.sh`** or **`brstart_static.sh`** can be used to set up bridge.

```
or32-elf-sim -f arch/openrisc/or1ksim.cfg vmlinux
```

  - demonstration only today, homework for the audience ☺

FPGA Bare Metal Demonstration

- System details:
  - ORSoC AB board
    - Actel A3P1000 FPGA
  - debug interface
    - ORSoC USB debug cable
    - or-debug-proxy
- Software:
  - bootloader from flash
  - "Hello World" program
    - download using GDB
- Run **`or-debug-proxy`**
  - **`/dev/ttyUSB0`** for JTAG
  - **`/dev/ttyUSB1`** for UART
- Connect **`picocom`** to **`/dev/ttyUSB0`**
  - **`picocom --b 115200 --p n --d 8 /dev/ttyUSB0`**

FPGA With Linux Demonstration

- System details:
  - ORPSoC Xilinx ML501 (Virtex 5) board port
    - OR1200
      - 32KB Insn/Data cache
      - 66MHz
      - FPU, MAC enabled
    - DDR2, CFI flash
    - 10/100 Ethernet
    - UART, SPI, I2C, GPIO
    - Uses about 50% of LX50's resources
    - All RTL source open and available
  - Debug interface:
    - ORSoC USB debug cable
    - OpenOCD as proxy

- Software:
  - U-boot bootloader from flash
    - 2011.09-rc2
    - Boot kernel from flash
    - Boot with TFTP download
  - Linux kernel port
    - 2.6.39
    - BusyBox
    - Currently a little bit limited due to lack of shared library/dynamic linking support.

Find out More

Find out More

- The OpenCores website
  - www.opencores.org
- The Open Source Hardware User Group (OSHUG)
  - www.oshug.org
  - evening meetings each month in London
  - first UK Open Source Hardware camp, 27 October
- FSCONS
  - keynote by Richard Stallman
  - OpenRISC workshop
- Commercial support
  - Embecosm – www.embecosm.com

Thank You

www.embecosm.com

www.opencores.org