Debugging with JTAG

Anna Dushistova, Alexandre Rusev, John Mehaffey

MontaVista Software Inc.

Agenda

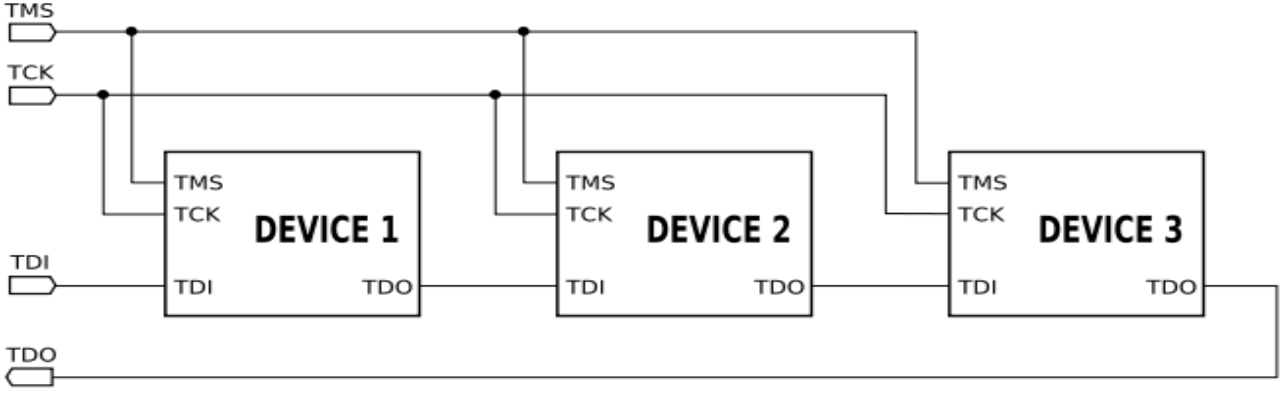
- What is JTAG?
- How does it work?
- JTAG use cases
- JTAG tools overview
- Working examples using Eclipse + CDT Hardware Debug Plugin
- Questions and Answers

JTAG Overview and History

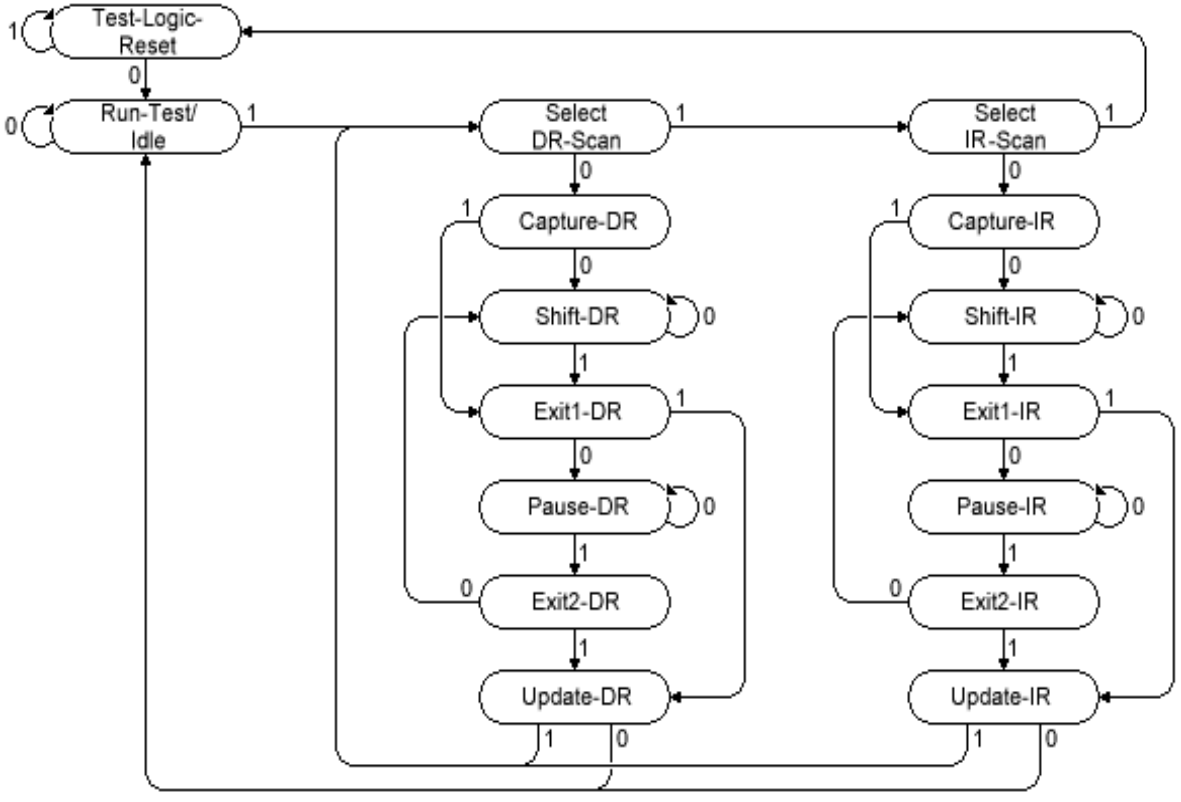
- The Joint Test Action Group (JTAG) name is associated with the IEEE 1149.1 standard entitled *Standard Test Access Port and Boundary-Scan Architecture*
- Started in 1990 as a digital test mechanism
- In 1994, a supplement containing a description of the boundary scan description language (BSDL) was added.
- JTAG is now primarily used for accessing sub-blocks of integrated circuits, but is also useful as a mechanism for debugging embedded systems

JTAG Electrical Characteristics

- A JTAG interface is a special four/five-pin interface
- The connector pins are:
  - TDI (Test Data In)
  - TDO (Test Data Out)
  - TCK (Test Clock)
  - TMS (Test Mode Select)
  - TRST (Test Reset) - optional



TAP State Machine



JTAG commands

- Standard required commands:
  - **BYPASS** 111...1 "all ones" command,
    the TAP command register is filled with ones
  - **EXTTEST** 000 ..0 "all zeros" command
    connects "boundary scan" register between TDI and TDO
  - **SAMPLE** 000...1 "one in last bit" command
    connects 1-bit "bypass" register between TDI and TDO
    TAP behaves as transparent 1-bit shift register

JTAG commands (continued)

- Standard optional commands:
  - **INTEST**

    places the IC in an internal boundary-test mode and selects the boundary-scan register to be connected between TDI and TDO
  - **CLAMP**

    sets the outputs of an IC to logic levels determined by the contents of the boundary-scan register and selects the bypass register to be connected between TDI and TDO
  - **HIGHZ**

    sets all outputs (including two-state and three-state types) of an IC to (high-impedance) state and selects the bypass register connected between TDI and TDO
  - **USERCODE**

    allows functional mode and selects the device register to be connected between TDI and TDO

JTAG commands (usage tips)

- Use BYPASS to count number of devices in the JTAG chain!
  - If each JTAG IC delays the TDI-TDO chain by one clock, we can send some data and check by how long it is delayed. That gives us the number of ICs in the chain.

Boundary scan description language

- Boundary Scan Description Language (BSDL) is a subset of VHDL used to describe how JTAG (IEEE 1149.1) is implemented in a particular device.
- For a device to be JTAG compliant, it must have an associated BSDL file.
- Many IEEE Std 1149.1 tools already on the market support BSDL as a data input format. These tools offer different capabilities to customers implementing IEEE Std 1149.1 into their designs, including board interconnect automatic test-pattern generation (ATPG) and automatic test equipment (ATE).

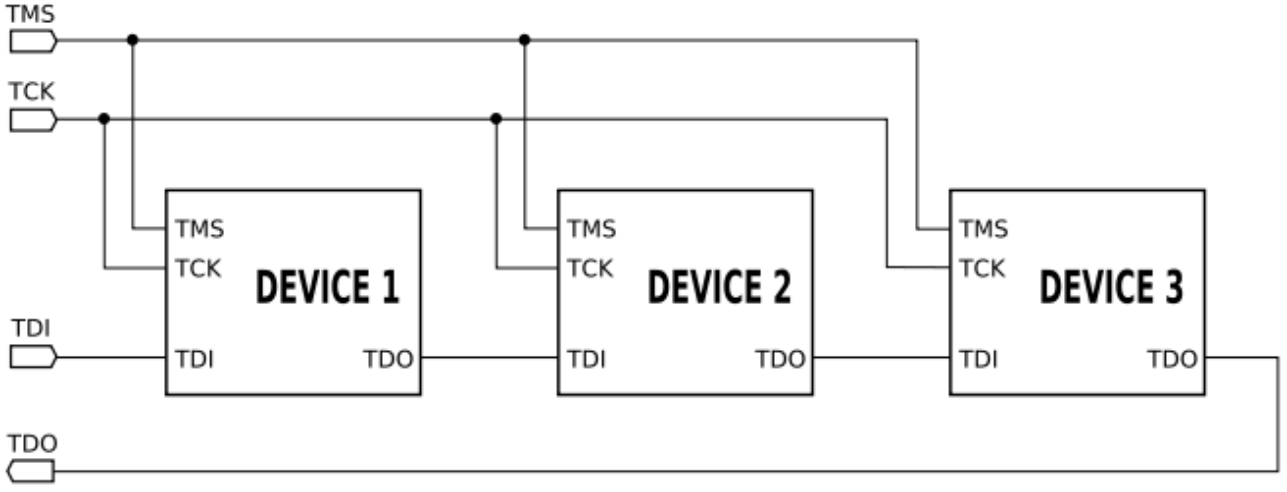
BSDL file format

BSDL files contain the following elements:

- Entity Description
- Generic Parameter
- Port Description
- Use Statements
- Pin Mapping(s)
- Scan Port Identification
- Instruction Register Description
- Register Access Description
- Boundary Register Description

Daisy Chain

Several chips on your Print Circuit Board (PCB) can be connected TDO of previous to TDI of the next creating a “daisy chain".



JTAG enables us to choose which chips in the daisy chain to deal with.

Daisy Chain (BDI configuration file example)

We just need to point out to BDI2000/3000 how many devices (chips) are connected before our CPU and after it along with total length of command registers of these groups of devices.

```
SCANPRED    2 24        ;Two JTAG devices connected before CPU: 8
                        ;(SYSTEM ACE) + 16 (XCF32P):
SCANSUCC    1 14        ;One only JTAG device is  connected after CPU: 6
                        ;(FPGA) + 8 (XC95144xl)
```

JTAG Typical usage scenarios

- Chip external connections test
- Flash programming
- Step-by-step debugging (instruction level)
- Processor IO ports manual testing

Chip external connection test – traditional approach

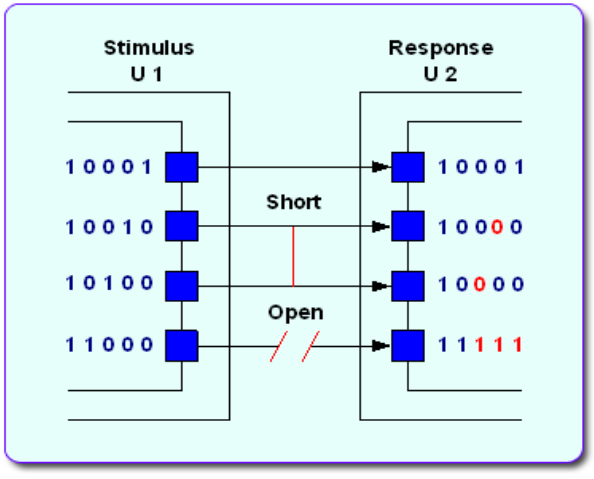
- **Classical approach**: use logic analyzers for debugging the interaction between the processor and peripheral chips
- **Problems**:
  - equipment costs ($10,000 and up)
  - board must have test points at specific places
  - software engineer needs help from electrical engineer to complete testing

Chip external connection test with JTAG

- Assume that that the circuit includes two faults:
  - a short between Nets 2 and 3, and
  - an open on Net 4
- Assume also that a short between two nets behaves as a wired-AND and an open is sensed as logic 1.

To detect and isolate the above defects, the tester shifts the patterns shown in Figure 2 into the U1 boundary-scan register and applies these patterns to the inputs of U2. The input values of the U2 boundary-scan register are shifted out and compared to the expected results.

In this case the results (marked in red) on Nets 2, 3, and 4 do not match the expected values, and therefore the tester detects the faults on Nets 2, 3, and 4.



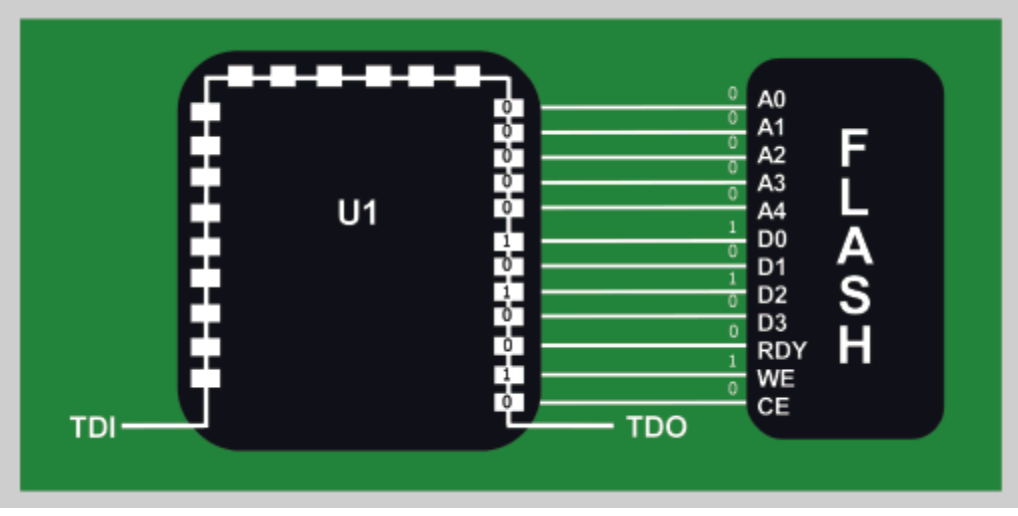
Chip external connection test with JTAG (continued)

www.opencollector.org

- offers lots of software projects targeted at this area

Flash programming

- Flash programming algorithm controls our chip output pin-s state using JTAG EXTTEST mode
- Pins logic state is set to 0,1 or high impedance state according to signal diagrams in FLASH chip datasheet



Flash programming (code snippet from OpenOCD)

```
static void
pxa250_bus_write( bus_t *bus, uint32_t adr, uint32_t data )
{
        part_t *p = PART;
        chain_t *chain = CHAIN;

        if (adr >= 0x04000000)
                return;

        part_set_signal( p, nCS[0], 1, 0 );
        part_set_signal( p, DQM[0], 1, 0 );
        part_set_signal( p, DQM[1], 1, 0 );
        part_set_signal( p, DQM[2], 1, 0 );
```

Flash programming (code snippet from OpenOCD)

```
        part_set_signal( p, DQM[3], 1, 0 );
        part_set_signal( p, RDnWR, 1, 0 );
        part_set_signal( p, nWE, 1, 1 );
        part_set_signal( p, nOE, 1, 1 );
        part_set_signal( p, nSDCAS, 1, 0 );
        setup_address( bus, adr );
        setup_data( bus, adr, data );
        chain_shift_data_registers( chain, 0 );
        part_set_signal( p, nWE, 1, 0 );
        chain_shift_data_registers( chain, 0 );
        part_set_signal( p, nWE, 1, 1 );
        chain_shift_data_registers( chain, 0 );
}
```

Step-by-step debugging

- Usually an internal capability of commercial JTAG devices
- OpenOCD

(http://openhardware.net/Embedded_ARM/OpenOCD_JTAG/) is an implementation of JTAG algorithms

Step-by-step debugging (continued)

Two corner cases of JTAG software debug implementation

- **First case:** software "speaks" with JTAG device in terms of TAP controller FSM states; bit-vectors need to be shifted in and out, to and from registers of TAP controller.
  - Physical connection to TAP controller is made in software bit-banging mode
  - Optimized access to TAP controller when intellectual JTAG cable accelerates JTAG operations at physical levels

Example:

Olimex-USB-OCD + OpenOCD for ARM

Step-by-step debugging (continued)

- **Second case:**
  - software "speaks" with JTAG device in terms of debugged process, such as "next instruction", "step-in", "step-over", "show registers"

Example:

Abatron BDI 2000

Example of Optimized access to TAP controller (JJTAG API)

- https://jjtag.dev.java.net/source/browse/jjtag/
- Using a logical representation of JTAG at this level enables the particular JTAG device implemnenter to make his desision to use bit-banging or HW-optimized acccess to JTAG FSM at physical layer transparent to applications.
- The API is designed this way if it uses "HW-optimized access" while internal implementation may be varied in any way.
- The code using APIs of this type may work in different OS environments; just a "driver" library to low-level part is needed
- In simplest case it could be bit-banging with parallel port cable. Users of PCs with Windows installed, Linux users, and "classical" EDA users (who tend to be Solaris users) may use it the same easy!

The following code illustrates the API usage model:

Code snippet

```
public class ARMICE {
  // Instruction register size
  public static final int IR_SIZE = 4;

  // Instruction register values
  public static final String
    EXTEST = "0000", // External test
    SCAN_N = "0010", // Select scan chain
    SAMPLE_PRELOAD = "0011",
    RESTART = "0100", // Restart core
    CLAMP = "0101", // Clamp pins
    HIGHZ = "0111", // HiZ pins
    CLAMPZ = "1001", // Clamp, HiZ pins
    INTEST = "1100", // Internal test
    IDCODE = "1110", // Read ID code
    BYPASS = "1111"; // Bypass core
```

Code snippet (continued)

```
// Scan chains
  public static final String
    SCAN0 = "0000", // Macrocell scan test
    SCAN1 = "0001", // Debug
    SCAN2 = "0010", // EmbeddedICE-RT registers
    SCAN3 = "0011"; // External boundary-scan

  /** JTAG controller */
  JTAGController controller;

  /** Constructor - we provide an
    implementation-independent controller
  */
  public ARMICE(JTAGController c) {
    controller = c;
  }
```

Code snippet (continued)

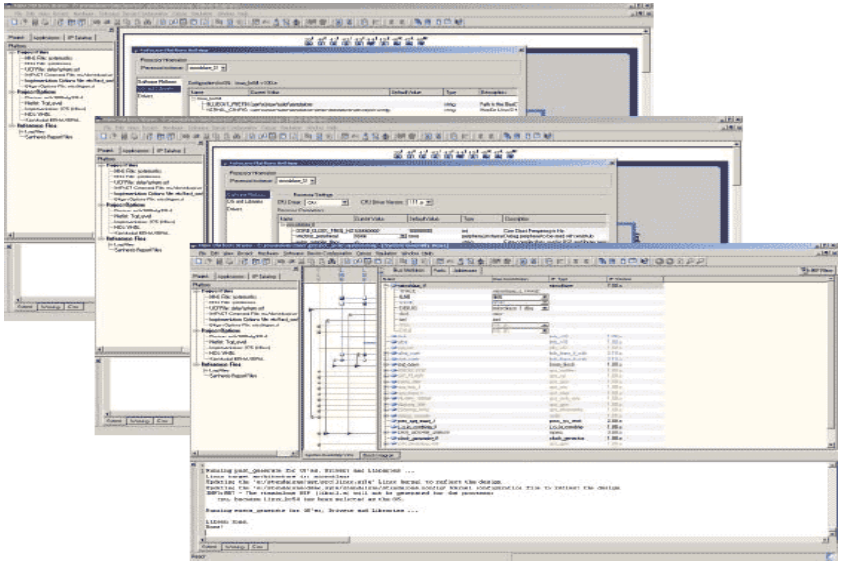
```
/* Some helper functions */
  String shift(byte ir, String dr) {
    controller.jump(ir);
    return controller.tdi(dr);
  }
  String shiftIR(String b) {
    return shift(SHIFT_IR, b);
  }
  String shiftDR(String b) {
    return shift(SHIFT_DR, b);
  }
  void selectScan(String s) {
    shiftIR(SCAN_N);
    shiftDR(s);
    shiftIR(INTEST);
    controller.jump(RUN_TEST_IDLE);
  }
```

Code snippet (continued)

```
String idcode() {
    controller.jump(SHIFT_IR);
    controller.tdi(IDCODE);
    controller.jump(SHIFT_DR);
    return controller.tdi(new Bits(0, 32)).toString();
 }
 public static void main(String[] args) throws Exception {
    ARMICE ai = new ARMICE(new JTAGWiggler(0x378));
    // reset the controller, set idle signal values
    ai.controller.initialize();
    // soft-reset the TAP, just in case
    ai.controller.reset();
    String c = ai.idcode(),
      ch = Integer.toHexString((int)Long.parseLong(c, 2));
    System.out.println("IDCODE: " + c + ", " + ch);
    System.out.println( "Controller is in state: " + description[ai.controller.state]);
  }
}
```

Chip-specific usages

- Today, as custom logic in hardware using FPGA and ASIC becomes more commonly used, the JTAG increases its importance as a tool for debugging interaction of programmable code with custom hardware.
- Final application developers want to see JTAG TAP controllers not only in processor part of the project but also in their custom logic. Xilinx provides primitives making it possible to embed TAP controller registers to custom logic and link them to daisy chain.



Commercial equipment

- There are a lot of offerings with varying functionality and price.

Conclusions

- JTAG is very simple at the hardware level as it is controlled by "trivial FSM"
- JTAG’s real expertise lies with upper-level algorithms hidden inside commercial and open source tools, which provide JTAG tools the ability to control FLASHes, processor machine instruction stepping, memory controller registers, and other facilities needed for the board initial startup (tunning SDRAM timings, GPIO multiplexing, initial console UART, etc.)

Opensource JTAG Tools Overview

- **Open Collector** http://opencollector.org/ **database provides information on a number of opensource JTAG tools**:
- **JTAG-O-MAT** (http://jtagomat.sourceforge.net) is a simple but highly configurable command-line tool for Win32 and Linux based PCs, primarily used to upload initial software to a virgin embedded system via a JTAG interface. In contrast to similar projects, the focus is on running automatic JTAG sequences. The code has been kept intentionally simple to maintain portability and allow modification without the risk of spoiling too many dependant parts.
- **jtag-util 0.02** (http://recycle.lbl.gov/~ldoolitt/jtag.html) is a first attempt to make a software base layer for host-side access to JTAG Test Access Ports. Most people seem to roll their own, but duplicates work, especially when it comes to portability across JTAG access adapters, and debugging or auto-detecting multiple devices on the chain. The community component of this exercise is to settle on a common API for mid-level access to JTAG devices. This way higher level software authors will not have to re-invent the low-level wheel, users will have their choice of adapter to the JTAG pins, etc.

Opensource JTAG Tools Overview (continued)

- **JTAG Tools** (http://openwince.sourceforge.net/jtag/) is a software package that enables working with JTAG-aware (IEEE 1149.1) hardware devices (parts) and boards through JTAG adapter. This package has open and modular architecture with the ability to write miscellaneous extensions (like board testers, flash memory programmers, and so on).
- **EBS 0.1** (http://ebsp.sourceforge.net/) The purpose of the Experimental Boundary Scan project (EBSp) is to provide completely open and flexible software support for commercially available JTAG/IEEE 1149-1 boundary scan master (BSM) controllers. BSM controllers are used to provide JTAG test bus control capability in various hardware platforms. The EBSp aims to provide adequate software support, which will facilitate the integration of these devices into platforms running under the Linux operating system and x86 architecture. Currently, the EBSp provides a Linux device driver for the Texas Instruments SN74ACT8990 BSM. Higher level software tools such as a hardware independent Serial Vector Format (SVF) parser with plug-in modules and Graphical User Interfaces (GUIs) are also available.

Opensource JTAG Tools Overview (continued)

- **ianjtag 1.2** (http://www.inaccessnetworks.com/projects/ianjtag/) is a collection of code and a set of tools for using the JTAG interface (present in most modern microprocessors) to perform hardware tests and to program Flash Memory Devices connected to the processor's bus. It is especially useful in embedded systems development projects for performing initial system tests and for bootstrapping the prototype systems. ianjtag tools run on a host system (e.g., a desktop computer with Linux) and access the target system (e.g., the embedded system's CPU board) through a simple 5-line hardware interface. In the present implementation the host system's parallel port is used as the hardware interface, though other arrangements can very easily be supported.
- **MITOUJTAG 0.0.2** (http://www.tokudenkairo.co.jp/jtag/) aims to provide comprehensive JTAG support software for Linux. This software can perform a graphical boundary scan of any IC with the BSDL files and a package shape description file. (For some standard packages such as DIP , PLCC , QFP and BGA, standard package files are prepared as the default.) Even if you don't have an expensive oscilloscope or logic analyzer, you can see any IC pin’s condition. And, it can be controlled remotely through TCP/IP.

OpenOCD project

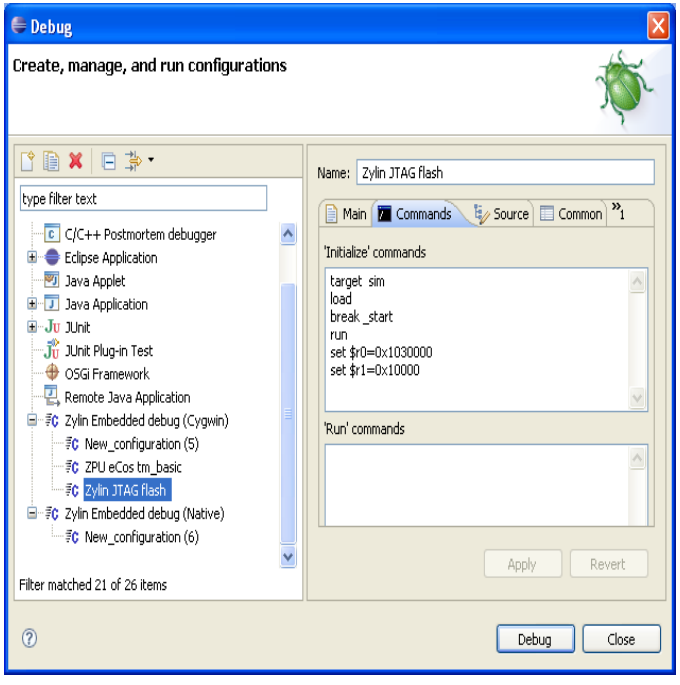
- The Open On-Chip Debugger (openocd) aims to provide debugging, in-system programming, and boundary-scan testing for embedded target devices.
- Openocd currently supports Wiggler (clones), FTDI FT2232-based JTAG interfaces, the Amontec JTAG Accelerator, and the Gateworks GW1602. It allows ARM7 (ARM7TDMI and ARM720t), ARM9 (ARM920t, ARM922t, ARM926ej-s, ARM966e-s), XScale (PXA25x, IXP42x) and Cortex-M3 (Luminary Stellaris LM3 and ST STM32) based cores to be debugged.
- Flash writing is supported for external CFI compatible flashes (Intel and AMD/Spansion command set) and several internal flashes (LPC2000, AT91SAM7, STR7x, STR9x, LM3 and STM32x). Preliminary support for using the LPC3180's NAND flash controller is included.

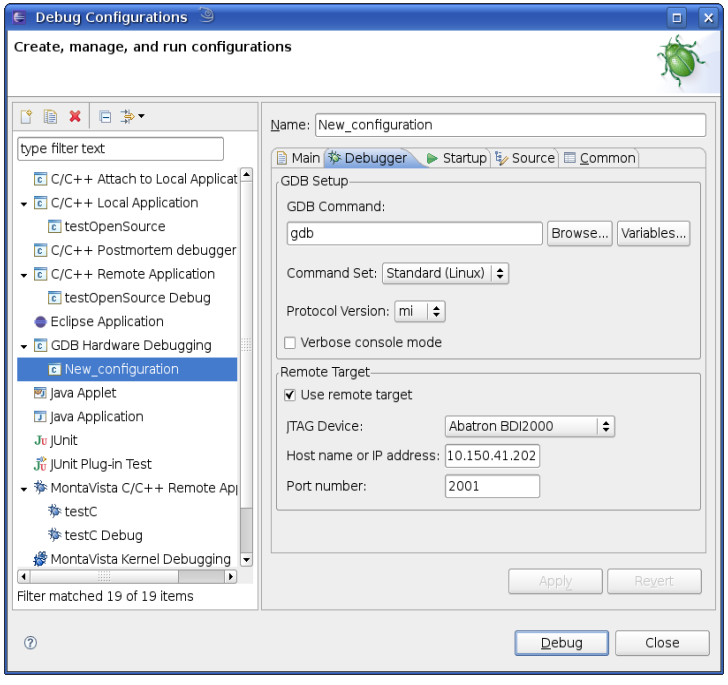
OpenOCD project (continued)

- The OpenOCD runs as a daemon, waiting for connections from clients (Telnet or GDB).
- It reads its configuration by default from the file openocd.cfg located in the current working directory. This may be overwritten with the '-f <configfile>' command-line switch.

Opensource Eclipse Front-ends

- Zylin Embedded CDT http://www.zylin.com/embeddedcdt.html
- CDT Hardware Debugging plugin http://www.eclipse.org/cdt/

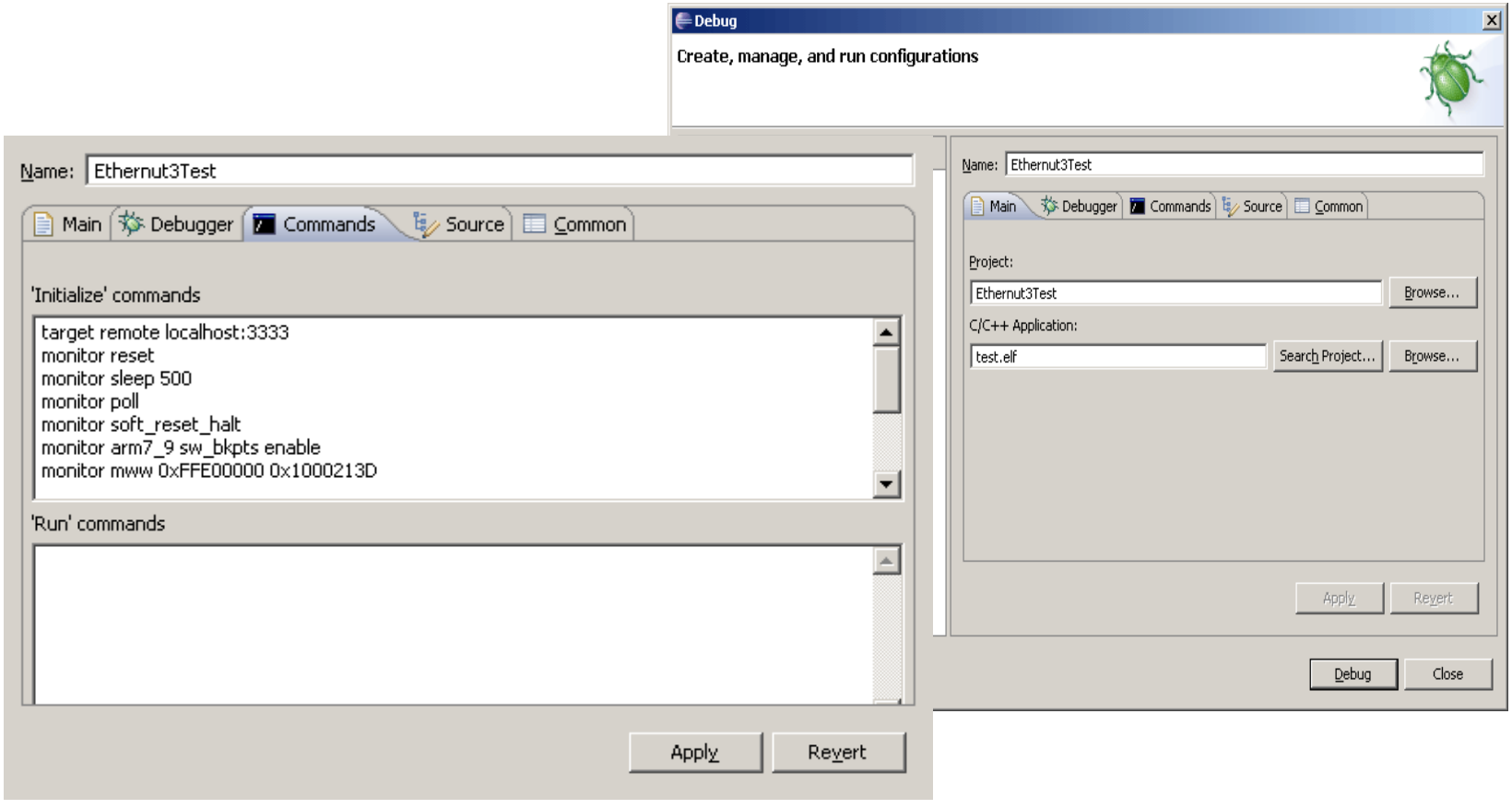




Zylin Embedded CDT

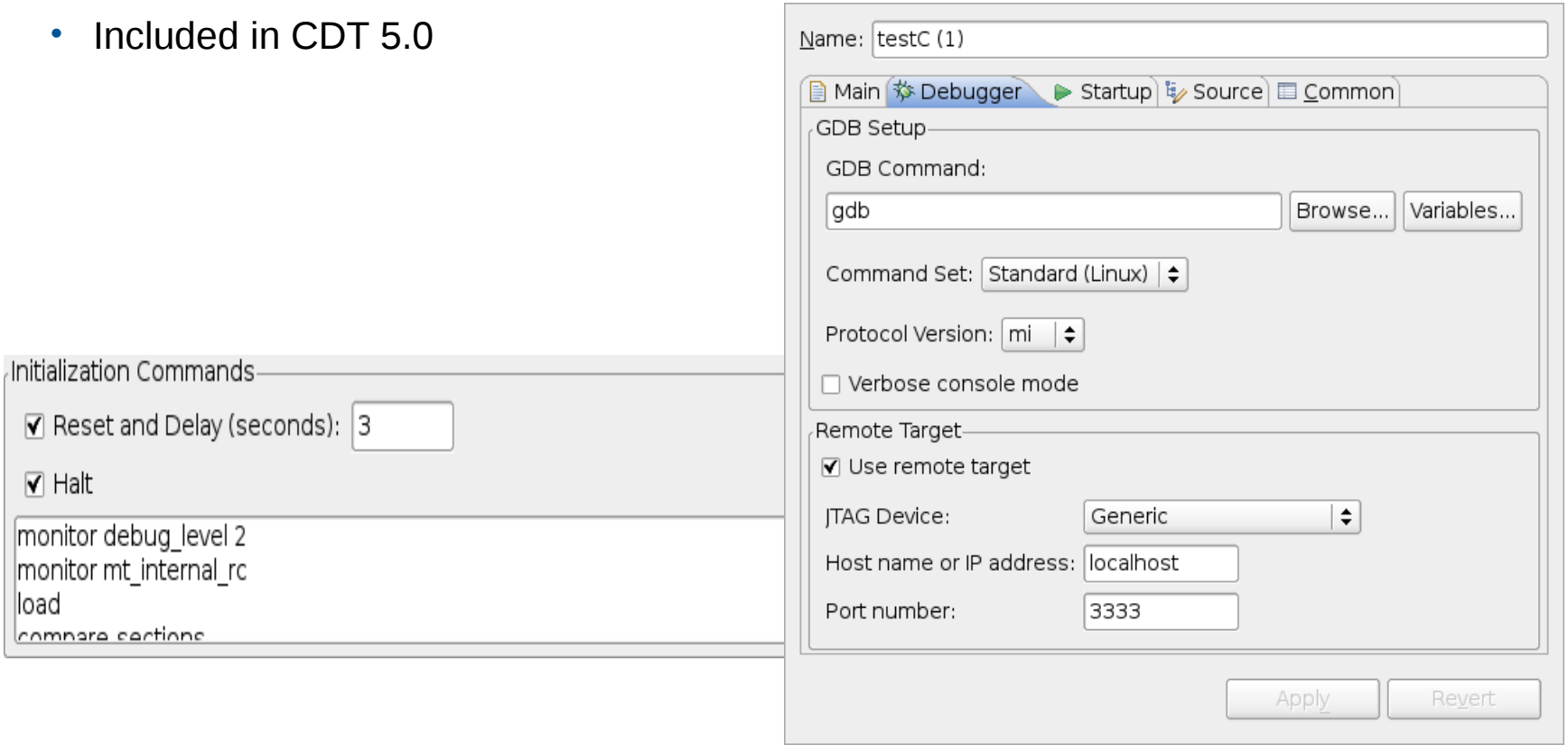
- Appeared first, works with the official Eclipse CDT 4.x or higher
- Actually modifies CDT
- Supports most of JTAG/hardware debuggers: you can issue "monitor XXX" commands to send commands to the JTAG/hardware debuggers in the GDB startup/run scripts to perform such functions as flash programming and resetting the target.
- No fancy flash programming GUIs
- Can be used with any CPU that GDB supports

Zylin Embedded CDT (continued)



CDT Hardware Debugging Plugin

- Created to avoid the need for forking the CDT
- Closely replicates what Zylin has done
- Included in CDT 5.0



- There is no single opensource or commercial solution for all JTAG use cases or any JTAG device.
- Vendors of cables/devices usually provide some software tools and GUI with their product for an additional price.
- Some such as Abatron (BDIx000 devices vendor) provide terminal UI and support for GDB compatible tools communication, so that a number of opensource tools such as Eclipse, OpenOCD, DDD may be used with such devices.
- User may also buy or assemble low-cost LPT JTAG cables, but these have low performance.
- For the purpose of software debugging with JTAG, opensource provides multiple solutions usable with numerous commercial JTAG cables/devices.

Commercial integrated JTAG solutions overview

- Linux Scope (Ultimate Solutions)

  http://www.ultsol.com/mfgs_comp_linuxscope.htm
- PowerView (Lauterbach)

  http://www.lauterbach.com/frames.html?powerview.html
- WindRiver Workbench (WindRiver)

  http://www.windriver.com/products/OCD/workbench_OCD/
- TimeStorm IDE (TimeSys)

  https://timesys.com/products/tools/timestorm
- Arriba Embedded Edition (Viosoft)

  http://www.viosoft.com/index.php?option=content&task=section&id=3&Itemid=30

Questions?

Acknowledgements

We would like to thank

- Dina Kommar (MontaVista)

  for testing Olimex Arm-USB-OCD with openOCD and Eclipse CDT Hardware Debugging plug-in
- Laurette Wharton (MontaVista)

  for language and style of this presentation