Software Driven Test of FPGA Prototype
Methods & Use cases

Krzysztof Szczur, Aldec Inc.
Radosław Nawrot, Aldec Inc.
Motivation - SoC verification challenge

• SoC = Hardware + Software
  – both need verification
  – software on critical-path (schedule risk)

• Complex architecture
  – Hierarchical Network-on-Chip (NoC)
    • Corner case hardware/software bugs
  – Re-usable IPs: CPU, MEM, DMA, …
    • RTL code not available for FPGA prototyping
  – Custom IPs: Co-processors, Function accelerators
    • Often need late hardware debugging
Tutorial Agenda

• Software driven v.s. traditional
• FPGA Technology enabling software driven verification
• Embedded software driven testbench – application examples with HES-US-440
  – Using HES Proto-AXI host interface
  – Bare metal approach
• Software driven testbench in simulation
  – QEMU co-simulation with Riviera-PRO
Traditional Prototyping Bench

• Controllability & Configuration
  – Dip switches
  – Push buttons
  – UART or JTAG ports

• Status & Debug
  – LED & Panels
  – Logic analyzers

• Live data streams
  – PHY components
  – Peripheral devices
Software Driven Testbench (SDT)

- Controllability & Configuration
  - Use CPU
- Status & Debug
  - Use CPU
  - Logic analyzers (external & on-chip)
- Live data streams
  - PHY components
  - Peripheral devices
FPGA Technology Enabling SDT

Xilinx Zynq and the like...

- **Processing System**
  - ARM Cortex CPU
  - Standard I/O peripherals

- **Programmable Logic**
  - Classic FPGA used for:
    - More peripherals
    - Glue logic & Bridge
Aldec HES-US-440

• Main FPGA:
  – Virtex UltraScale™ XCVU44
  – 26 Million ASIC Gates

• Host FPGA
  – Zynq-7000, XC7Z100
  – CPU: ARM Cortex A9
  – FPGA: Kintex-7
HES-US-440 Diagram

• Memory
  – DDR4 (64 GB) : 2 x SO-DIMM
  – RLDRAM-3 (1152 Mb)
  – NAND Flash, SPI Flash
  – 2x MicroSD card slots

• I/O Connectivity
  – Inter-FPGA:
    – 80 DIFF / 160 SE, 4 GTH
  – FMC connectors
    – 320 DIFF / 640 SE, 26 GTH

• Peripherals
  – PCIe x8 gen3 & gen2
  – USB 3.0 & 2.0 & OTG, SATA
  – Ethernet 1Gb, QSFP+ 40Gb
  – BullsEye

• Clocks
  – 7x programmable clock modules
SDT application examples on HES-US-440

**Case-1: Aldec HES-ProtoAXI**
- Zynq image ready to use (PL & PS)
- Use Aldec Proto-AXI IP
- Ready Embedded Linux env.
- Simple Proto-AXI C/C++ API
- Spend all your time to develop SDT

**Case-2: Bare metal / Custom**
- Use Xilinx Vivado & SDK
- Create custom C2C bridge in PL
- Create custom SDT in PS
Use Case-1 with HES Proto-AXI

1. Connect DUT to HES Proto-AXI & implement FPGA bitstream
2. Develop & build software TB
3. Run test
Step-1: Connecting DUT to HES-Proto-AXI

- Connect HES Proto-AXI
  - Use any HDL Editor
  - VHDL or Verilog templates available
  - Fixed size AXI4 memory mapped interface
  - Master interface is optional
- Configure address translation parameters (Address Remapper)
- DUT gets access to on-board DDR4 through HES Proto-AXI
- Compile and implement project in Xilinx Vivado
Step-2: Developing & building testbench

- Develop testbench as C/C++ code
  - Use HES Proto-AXI C API
  - `#include "hesprotoaxiapi.h"`
  - Simple, user app level API
    - needn’t know Zynq or Linux kernel
- Compile and build
  - Cross-compilation on host PC
  - Use GCC toolchain from Xilinx SDK
    - `arm-Xilinx-linux-gnuabi-g++`

```c
// Get Proto-AXI handler
HesProtoAxiHandler HesProtoGetAxiInterface(
    const char *axisimconfig = 0);

// AXI write (blocking transaction)
HesProtoAxiTrans HesProtoAxiWrite(
    HesProtoAxiHandler _handler,
    HesProtoAxiLocation _location,
    HesProtoAxiU64 _address32,
    HesProtoAxiBuffer _buffer,
    HesProtoAxiU64 _bufferSize32);

// AXI read (non blocking transaction)
bool HesProtoAxiReadNoCheck(
    HesProtoAxiHandler _handler,
    HesProtoAxiLocation _location,
    HesProtoAxiU64 _address32,
    HesProtoAxiBuffer _buffer,
    HesProtoAxiU64 _bufferSize32);

// GPI write
HesProtoAxiStatus HesProtoAxiWriteGpi(
    HesProtoAxiHandler _handler,
    HesProtoAxiU16 _value);

// GPIO read
HesProtoAxiStatus HesProtoAxiReadGpio(
    HesProtoAxiHandler _handler,
    HesProtoAxiU16 * _out_value);
```
Step-3: Running Test

Proto-AXI Zynq Images Linux & FPGA Config

DUT Bitstream XCVU440

Software TB APP tb.elf

Ethernet UART/USB

Welcome to minicom 2.3
OPTIONS: 118n
Compiled on Mar 22 2017, 00:13:25.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys
aldec_hes/media/card/v1.1/bin# AT S7-45 50=0 L1 V1 X4 &c1 E1 Q0
[1] 1283
-c: AT, command not found
-c: c1: command not found
[1]+ Done(127)
atdec_hes/media/card/v1.1/bind
HES Proto-AXI Highlights

Data for HES-US-440
- AMBA AXI4 memory mapped
- 8x Master & Slave interfaces
- Data width: 256, Burst length: 256
- Local clock: 160 MHz
- RAM space: 16GB DDR4 + 64MB RLD
- Customizable address translation
- 16x GPI and GPIO additional lines
- Simple C API
- HES Proto-AXI simulation model available
- Embedded & PC Host modes

© Accellera Systems Initiative
Questions

Next: Case-2: Bare metal approach
Use Case-2 – Bare metal approach

1. Test controller (Zynq)
   A. Hardware design
   B. Embedded Linux development
   C. Software development

2. Design connectivity (U440)
   A. Custom interconnect
   B. FPGA implementation
Step 1.A: Hardware design

- Short path: start with reference design from Aldec
- Customize reference design
  - Add your custom chip-2-chip connectivity
- Use Aldec Riviera-PRO with QEMU to verify your changes
- Run Vivado to implement bitfiles and export hardware description
Step 1.B: Embedded Linux development

- Embedded Linux projects available:
  - Yocto
  - Petalinux

- Other than Linux:
  - Free RTOS
  - Bare metal – no OS
  - Xilinx SDK provides toolchain for such mode

Embedded Linux Development

1. Bootloader u-boot.elf
2. Kernel image ulimage
3. Devicetree devicetree.dtb
4. Filesystem uramdisk.image.gz

Manual compilation from sources or Linux build tools like Buildroot, Yocto or Xilinx Petalinux
Step 1.C: Software development

- Use Xilinx SDK
  - to develop test application
  - create bootable Linux image

- Copy files to Micro-SD
  - Bitstream for FPGA (PL configuration)
  - Bootloader for ARM (PS configuration)
  - Linux image & device tree
  - User application (app.elf)
Step 2: Design connectivity

- Use Xilinx Vivado to implement XCVU440
  - Create custom interconnect
    - Coupling with one created in Zynq
    - Use any standard (AXI, PCIe)
    - Use any signaling (LVDS, GTX)
  - Connect with the DUT
  - Run FPGA implementation
Questions

Next:
Software driven testbench in simulation:
QEMU co-simulation with Riviera-PRO
QEMU CO-SIMULATION WITH RIVIERA-PRO

Radosław Nawrot
Software Products Manager, Aldec Inc.
QEMU co-simulation platform

• Parts of the flow:
  – Riviera-PRO Advance Verification Platform
  – Aldec AXI BFM
  – Aldec QEMU Bridge
  – QEMU Emulator
QEMU co-simulation platform

- Full debug capabilities of RTL IP Core in Riviera-PRO simulator:
  - Waveforms
  - Hardware Breakpoints
  - Hardware steps
  - Transaction based verification and debug
- Kernel and driver debug via GDB
  - Software Breakpoints
  - Variable probing
- Zynq Linux OS ready to use on QEMU without modifications
Riviera-PRO Verification Platform

A Verification Platform that grows with your requirements.

System on Chip
Internet of Things

System Interfaces
MATLAB®, SystemVue®

Debugging
Mixed-Language IDE, Code Tracing, Waveform, Dataflow, Image Processing, DSP

Metrics
Coverage, Test Rank, Assertions, Code and Functional Coverage, SVA/PSL/OVM

Test Automation
TLM, RTL, Gate-Level, SystemVerilog, SystemC, Python, UVM, C/C++

Simulation
Mixed-Language, Mixed-Signal, Hardware/Software, Verilog, Verilog-AMS, VHDL

© Accellera Systems Initiative 2017
Riviera-PRO Highlights

• High Performance Simulation
  – Extensive simulation optimization algorithms
  – Support for latest Verification Libraries: UVM, OSVVM, UVVM, CocoTB and more

• Advanced Debugging
  – Transaction Level simulation end debug
  – Multi-language debug environment (Verilog, VHDL, SystemVerilog, SystemC, Verilog-AMS)
  – Support for MATLAB and Simulink
  – C/C++ debug environment
  – Support for external C/C++ compilers (GCC, Visual C++)
  – UVM Toolbox, Graph and Class Viewer
  – Code tracing, Waveform, Dataflow, FSM window, Coverage, assertions, memory visualization
  – Comprehensive Assertions-Based Verification (SVA and PSL)
  – Advanced Code and Functional Coverage
  – User-defined test plan linking with coverage database
  – Plot Viewer and Image Viewer
QEMU – Virtual Platform

• Free and open-source
• Fast emulator
• Supports many machines:
  – Xilinx-zynq-a9
• Supports many ARM processors:
  – ARM926
  – ARM946
  – Cortex-A8
  – Cortex-A9
  – Cortex-A15
  – Cortex-A53
  – Cortex-A57
Riviera-PRO and QEMU

QEMU Emulator
(Linux for TYSOM board)
with
Xilinx SystemC SoC Library
ARM Cortex-A9 model

ARM application
running on Linux

Aldec Riviera-PRO

Aldec QEMU Bridge

Aldec AXI BFM

Custom IP Core

© Accellera Systems Initiative
Aldec AXI BFM usage

Design stage
- Distributed with precompiled libraries
- Instantiation in HDL part of the project

Compilation and Configuration stage
- Configuration set by parameters
- Turn on build-in transaction recorder

Simulation stage
- Automatic connection with Aldec QEMU Bridge process done at simulation stage
Aldec AXI BFM

- AXI 3 Master
- AXI 3 Slave
- AXI 4 Master
- AXI 4 Slave
- AXI 4 Lite Master
- AXI 4 Lite Slave
- AXI 4 Stream Master
- AXI 4 Stream Slave
AXI BFM instance in HDL Code
AXI BFM instance in HDL Code
Built-in Transaction Recorder
Riviera-PRO QEMU environment

© Accellera Systems Initiative
Hardware and Software co-sim
Transaction based debug
Transaction logs
Riviera-PRO and QEMU – project flow
# TySOM – Zynq Based EDK

<table>
<thead>
<tr>
<th>Specs</th>
<th>TySOM-1</th>
<th>TySOM-2</th>
<th>TySOM-2A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soc</td>
<td>XC7Z030</td>
<td>XC7Z045/100</td>
<td>XC7Z030</td>
</tr>
<tr>
<td>DDR3</td>
<td>512MB</td>
<td>168</td>
<td>166MB</td>
</tr>
<tr>
<td>SPI Flash</td>
<td>4MB</td>
<td>16MB</td>
<td>16MB</td>
</tr>
<tr>
<td>SD/MMC</td>
<td>uSD</td>
<td>uSD</td>
<td>uSD</td>
</tr>
<tr>
<td>E2C EEPROM</td>
<td>64kB</td>
<td>64kB</td>
<td>64kB</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>USB 3.0</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UART/USB-UART</td>
<td>2</td>
<td>1 USB → UART</td>
<td>1 USB → UART</td>
</tr>
<tr>
<td>JTAG PL Header</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GPIO Ethernet</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Ethernet IEEE1588</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Audio IN/OUT</td>
<td>V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HDMI/VGA</td>
<td>HDMI</td>
<td>HDMI</td>
<td>HDMI</td>
</tr>
<tr>
<td>Camera Con</td>
<td>V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mPCIe</td>
<td>V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FMC</td>
<td>-</td>
<td>× 2</td>
<td>× 1</td>
</tr>
<tr>
<td>PCIe</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GTX</td>
<td>1 on MMCX con</td>
<td>16 on FMC con</td>
<td>4 on FMC con</td>
</tr>
<tr>
<td>ADC</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>WiFi/BT</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RTC</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Touch Panel/LCD Interface</td>
<td>T.P &amp; LCD</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>User DIP switch</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>User LED</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>FMC - IoT</th>
<th>FMC - Vision</th>
<th>FMC - ADAS</th>
<th>FMC - INTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boards</td>
<td>FMC, 1× Ethernet, IEEE1588, Wi-Fi, Bluetooth 4.1, ZigBee, 60 GHz, RFFE, 802.11ad, WirelessHART, Sigfox, WiMBus, mPCIe, USB 2.0, Micro SD, RS-485, 2×6 Digilent FMC</td>
<td>FMC, 1× HDMI IN, 1× HDMI OUT, 1× Display Port IN, 1× Display Port OUT, MIPI DSI, MIPI CSI, Audio Codec, 2× USB 3.0</td>
<td>FMC, 5× FBPD-Link III (LVDS interface), 1× URM37_V4.0 Ultrasonic sensor connector, 1× UDMAR-Lite_V2 Connector</td>
<td>FMC, 1× Ethernet, PCIe Express connector, QSFP+ connector, 1× USB 2.0, 1× UART to USB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication interfaces</th>
<th>FMC - IoT</th>
<th>FMC - Vision</th>
<th>FMC - ADAS</th>
<th>FMC - INTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boards</td>
<td>FMC, 1× Ethernet, IEEE1588, Wi-Fi, Bluetooth 4.1, ZigBee, 60 GHz, RFFE, 802.11ad, WirelessHART, Sigfox, WiMBus, mPCIe, USB 2.0, Micro SD, RS-485, 2×6 Digilent FMC</td>
<td>FMC, 1× HDMI IN, 1× HDMI OUT, 1× Display Port IN, 1× Display Port OUT, MIPI DSI, MIPI CSI, Audio Codec, 2× USB 3.0</td>
<td>FMC, 5× FBPD-Link III (LVDS interface), 1× URM37_V4.0 Ultrasonic sensor connector, 1× UDMAR-Lite_V2 Connector</td>
<td>FMC, 1× Ethernet, PCIe Express connector, QSFP+ connector, 1× USB 2.0, 1× UART to USB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous</th>
<th>FMC - IoT</th>
<th>FMC - Vision</th>
<th>FMC - ADAS</th>
<th>FMC - INTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boards</td>
<td>2× user clock, 1× clock for SATA, Dedicated clock to PCIe</td>
<td>2× Oscillators, 2× Clock for transceivers</td>
<td>Buzzer, 2× Oscillator</td>
<td>3× Oscillator, 8× User Dip switch, 8× User LED, GPIO (20 I/Os, 2×20 gold pins connector)</td>
</tr>
</tbody>
</table>
Summary

• SoC FPGA Verification Platform
• Processor modeling based on QEMU emulator (instead of simplified BFM model)
• Hardware and Software debugging
• Fast RTL IP Core Verification and bugs fixing
• Driver and Firmware Verification
• Built-in Transaction Recorder
• Support for many platforms and processors
• Support for different Linux versions
  – Yocto project
Questions