A Deep Dive into HW/SW Development with PULP

Robert Balas <balasr@iis.ee.ethz.ch>
Manuel Eggimann <meggimann@iis.ee.ethz.ch>

http://pulp-platform.org  @pulp_platform  https://www.youtube.com/pulp_platform
Introduction Round

Robert Balas

Manuel Eggimann
Introduction – Organization of this Training

Day 1
- PULPissimo SoC Architecture
- Software Environment
- RTL Development Flow
- RTL Simulation/Debug Environment

Day 2
- FPGA Port
- PULP IP Landscape
- Hands-on Full-stack IP Integration Exercise
- PULPissimo Memory Layout Modification
SystemVerilog Atrocities
import pkg_some_other_ip::*;

module top
  #(
    parameter OUT_WIDTH
  )

  input logic [IP_BITWIDTH-1:0] data_i;
  output logic [OUT_WIDTH-1:0] data_o;

endmodule
module top

import pkg_some_other_ip::IP_BITWIDTH;

#(

    parameter OUT_WIDTH
)

(input logic [IP_BITWIDTH-1:0] data_i;
output logic [OUT_WIDTH-1:0] data_o ;
);
endmodule
module my_ip

#(
  parameter WIDTH = 32,
  parameter BE_WIDTH = WIDTH/8, // could be changed
)

input logic [WIDTH-1:0] data_i;
input logic [BE_WIDTH-1:0] be_i;

parameter MY_CONSTANT = 42; // I'm not a constant :-(
endmodule
module my_ip

  #(
    parameter WIDTH = 32,
    localparam BE_WIDTH = WIDTH/8  //Cannot be changed
  )

  input logic [WIDTH-1:0]   data_i;
  input logic [BE_WIDTH-1:0] be_i;

);  
localparam MY_CONSTANT = 42;  //I'm a constant :-)
endmodule
Elaboration SystemTasks
(supported since SV-2012)

module my_ip

#(

    parameter NR_CORES = 32
)
(

    input logic [WIDTH-1:0]  data_i;

    input logic [BE_WIDTH-1:0] be_i;

)

endmodule // my_ip
Elaboration SystemTasks
(supported since SV-2012)

module my_ip

    #(  
        parameter NR_CORES = 32 //Must be power of 2! 
    )(

        input logic [WIDTH-1:0] data_i;
        input logic [BE_WIDTH-1:0] be_i;
    );

endmodule // my_ip
module my_ip

    #(
        parameter NR_CORES = 32 //Must be power of 2!
    )

    input logic [WIDTH-1:0] data_i;
    input logic [BE_WIDTH-1:0] be_i;

    if (NR_CORES == 0 || (NR_CORES & (NR_CORES-1)) != 0)
        $error("NR_CORES must");

endmodule // my_ip

Even better
 Includes

```
`include "macros.svh"
module my_ip
  (input logic clk_i);
endmodule : my_ip
```

Don’t use generic names!
`include "my_ip_macros.svh"

module my_ip

  (input logic clk_i);

endmodule : my_ip

Prefix all header file names and defines with to avoid naming colisions and redefinitions
Generate Statements

genvar i;

generate
    for ( i = 0; i < 10; i++ ) begin
        my_subip i_subip(...)
    end
endgenerate
Generate Statements

```verilog
for (genvar i = 0; i < 10; i++) begin : gen_sub_ips
    my_subip i_subip...
end
```

- Don’t use generate regions. They are redundant in SystemVerilog (and Verilog 2005).
- Always label your generate blocks. Otherwise the hierarchical name is toole-dependent!
An Overview on PULPissimo/ PULP SoC
PULPissimo

soc_domain
(soc_domain.csv)

provides data for pad_control

Safe Domain
(safe_domain.csv)

controls muxing

Padframe
(pad_frame.csv)
PULPissimo Special Toplevel Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pad_xtal_in</td>
<td>32 kHz input clock (no internal crystal oscillator IP). Connects to internal FLLS (ASIC) or clock managers (FPGA port).</td>
</tr>
<tr>
<td>pad_reset_n</td>
<td>Active-low asynchronous reset (internally synchronized)</td>
</tr>
<tr>
<td>pad_bootsel</td>
<td>Affects boot behavior according to program in bootrom.</td>
</tr>
<tr>
<td>pad_jtag Xxx</td>
<td>Debug Access port for bus access and core debugging (single step, SW breakpoints)</td>
</tr>
</tbody>
</table>

- Be careful about parameters! They are not always supposed to be changed or do not work anymore.
- There is a lot of dead code (remainings from tapeout specific fixes and legacy code).
Padframe

- Contains technology independent wrappers of IO pads
- Signals for each IO pad:

<table>
<thead>
<tr>
<th>Direction (padframe perspective)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>oe_&lt;padname&gt;_i</td>
<td>Active high output driver enable</td>
</tr>
<tr>
<td>Output</td>
<td>in_&lt;padname&gt;_o</td>
<td>Logic signal from pad to SoC</td>
</tr>
<tr>
<td>Input</td>
<td>out_&lt;padname&gt;_i</td>
<td>Logic signal from SoC to pad</td>
</tr>
<tr>
<td>Inout</td>
<td>pad_&lt;padname&gt;</td>
<td>The actual pad signal that is connected to the toplevel</td>
</tr>
<tr>
<td>Input</td>
<td>pad_cfg_i</td>
<td>Additional config signals for pad (e.g. pulldown enable)</td>
</tr>
</tbody>
</table>
Safe Domain

- Contains logic that must not be power gated
- Lives in a separate module for simplified power intent specification in CPF or UPF

Modules in PULPissimo:

- pad_control: Multiplexes functionalities of io pads between (e.g. spi sck or gpio)
- Rst_gen: Synchronizes the reset signal to reference clock. Only used for modules within safe domain that are directly clock with ref_clk.
soc_domain/pulp_soc

- Wraps the actual heart of the SoC; The pulp_soc IP.
- Pulp_soc was designed to be the main soc fabric of all our larger 32-bit PULP chips
- Contains many signals that are only used when there is an additional multi-core cluster present
soc_domain/pulp_soc

PULPissimo

Multicore PULP

SoC

Cluster

Tightly Coupled Data Memory

TCMD Interconnect

DMA

PE#0

PE#1

PE#2

PE#3

Instruction Cache

CIM

Accel.
# PULPissimo Clock Domains

<table>
<thead>
<tr>
<th>Clock Name</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref_clk_i</td>
<td>Signal from directly taken from pad_xtal_in.</td>
<td>Connects to internal FLLs/PLLs</td>
</tr>
</tbody>
</table>
| slow_clk_i   | In ASIC version, identical to ref_clk_i. For FPGA version, passes through glitch free clock divider since certain boards (e.g. Genesys2) have very fast external oscillators. This one must always be 32kHz | • Timers  
• Directly used as interrupt source |
| periph_clk_i | One of the two fast clock. Generated by internal FLL/PLL from ref_clk_i.   | • Drives IO facing peripherals (e.g. UART, SPI, I2C)                   |
| soc_clk_i    | Fastest clock in the system. Generated by second internal FLL/PLL from ref_clk_i. | • Drives everything else (Core, memory, interconnect) in the SoC.     |
SoC Peripherals
APB SoC Control

- APB Register File with Global configuration signals for SoC

<table>
<thead>
<tr>
<th>Regname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Address</td>
<td>Contains the boot address of the core.</td>
</tr>
<tr>
<td>Fetch Enable</td>
<td>Enables instruction fetching in the core. By default controlled with an external signal (default 1)</td>
</tr>
<tr>
<td>Padvux</td>
<td>Signals used by pad_control to multiplex between dual usage of pads (GPIO or peripheral)</td>
</tr>
<tr>
<td>Pad Configuration</td>
<td>Controls the special pad control signals when the pad is in GPIO mode</td>
</tr>
<tr>
<td>JTAG Register</td>
<td></td>
</tr>
</tbody>
</table>
L2 RAM Multibank

- Contains Wrappers for SRAM (or block memory) macros

- Internal address conversion
  - Address bit truncation
  - Offset subtraction if necessary and
  - conversion to 32-bit word addressing (wordwidth of SRAMs is 32-bit, core takes care of misaligned load/stores in hw)

- Protocol converter
  - assign gnt = request
  - r_valid delayed by one cycle
SoC Interconnect
TCDM Protocol

- Single cycle latency protocol
- Used for communication between core and memories
- Does not allow multiple outstanding transactions!
- Req must not depend on gnt, but gnt typically does combinatorially depend on req.
TCDM Protocol

Read Transaction

Write Transaction
SoC Interconnect
Training resumes at 13:00

Update: Resuming at 13:15 (food didn’t arrive in time:-)
PULP-SDK vs PULP-RUNTIME

PULP-SDK
- Fully-featured SDK
- Drivers
- Complex
- FPGA support

PULP-RUNTIME
- Minimal bare-metal runtime
- Boot-to-main
- Only uart driver
- FPGA support
- Active
PULP-RUNTIME

- `crt0.S` to `main()` with minimal initialization
- Only uart driver available
- HAL available
PULP-RUNTIME - Overview

PULP-RUNTIME

- bin
- configs
- drivers
- include
- kernel
- lib
- rules
- scripts
PULP-RUNTIME – Build Flow

runtime

src.mk

configs/default.mk  default_rules.mk

$(target).mk

Compatibility

pulp_rt.mk

application

Makefile
PULP-RUNTIME – Hello World - Setup

• Directory structure
  • training/pulpissimo (v6.0.0) git clone https://www.github.com/pulp-platform/pulpissimo
  • training/pulp-runtime (v0.0.6) git clone https://www.github.com/pulp-platform/pulp-runtime
  • training/sw git clone https://www.github.com/pulp-training/sw

• Compiler (pulp-gcc)
  • https://github.com/pulp-platform/pulp-riscv-gnu-toolchain
PULP-RUNTIME – Hello World - Demonstration

1. Simulator location
   $ source setup/vsim.sh

2. Configuration
   $ source configs/pulpissimo.sh

3. Compiler
   $ export PULP_RISCV_GCC_TOOLCHAIN=...
PULP-RUNTIME – FPGA specific

1. ARCHI_FPGA_FREQUENCY
2. ARCHI_FPGA_FC_FREQUENCY
3. configs/fpgas/pulpissimo/*.sh
PULP-RUNTIME – Trivial Driver

1. We add a trivial driver to the pulp-runtime

$ cd sw/runtime-trivial-driver
PULP-RUNTIME – Trivial Driver - Exercise

1. Pass the second test in runtime-trivial-driver. Put the required macro and function in a separate .c/.h file.

$ cd sw/runtime-trivial-driver
Exercise Time

- Join a breakout room of your choice
- In case you need help or have a question, visit: https://bit.ly/37nnI8P and enqueue yourself
- If you cannot use Zoom to share your screen or have issues with it: https://fisch.ddns.net/call/mhq9864w
- Consult https://fisch.ddns.net/sites/pulp_training for SoC schematics and FAQ (hopefully we have time to update it adhoc)
PULP-RUNTIME - Reggen

- Open-Source
- Used in tapeout
- Single source of truth
- Easier hw/sw co-design
- Lowrisc IP supported
We at ETH added some patches

- Tilelink is rather complicated and we don’t use it
- Add support for register_interface (simple protocol to access register)
- Lots of protocol converters (AXI, APB, TCDM (partial)) and CDC
- «reg» keyword to hjson
PULP-RUNTIME – Reggen - Demonstration

1. We show how the hjson description looks like
2. We generate a header file and SystemVerilog code from it
3. We use it a small program

$ cd sw/runtime-reggen
PULP-RUNTIME – Reggen - Exercise

1. Integrate the generated header file into pulp-runtime
2. Try to generate documentation from the hjson description
3. Explore the --help options
PULPissimo – Booting

1. Boot procedure
2. Introduction to Linkerscripts
3. Boot code, compile and link
PULPissimo – Boot procedure

- **Reset**
  - Hardcoded:
    - Boot ROM:
      - 0x1A00_0000
      - ...
      - 0x1A00_2000
      - Bootsel = 0
    - Boot From JTAG = busy loop
    - Boot From SPI

- **Debug Module**

- **L2**: 0x1C00_0000
  - ...
  - 0x1C08_0000
PULPissimo – Introduction to Linkerscripts

• Compiler groups instructions and data in sections
  • .text = instructions
  • .data = initialized variables
  • .bss = zero initialized variables
  • .rodata = read only data

• Linkerscript = Set of rules on how to map sections to memory
PULPissimo – Bootcode - Demonstration

$ cd boot_code/
RTL Development Flow / Tools
IP Dependency Management (IPApprox)

- Transitively resolves Dependencies between IPs
- Automatically checks out sub IP repositories
- Manages tool and target specific file sets
- Generates analyzes and elaborate scripts for simulation, ASIC & FPGA Synthesis
- Called by two python scripts (in PULPissimo they are called update_ips & generate_scripts)
**IP Approx**

**IP Package**

- **src_files.yml (required)**
  - Source files
  - Preproc. Macros
  - Tool specific flags

- **ips_list.yml (optional)**
  - Dependency Declaration
  - Version specification

- **rtl_list.yml (optional)**
  - Local sub IP declaration (src_files.yml) in an subdirectory of the current IP instead of external repository
IPApprox Questasim Output

- **IPdb.export_make**
- **IPdb.generate_makefile**
- **IPdb.generate_vsim_tcl**

**IP Makefiles**
- Compiles source files to Questasim Libs

**toplevel Makefile**
- Compiles all IPs in project
  - Lists output libraries
  - Used to start vsim with right arguments

**IPApprox**
IPApprox FPGA Output

- ipdb.export_vivado
  - contains TCL variables with list of source files for every IP
  - references variables

- ipdb.generate_vivado_add_files
  - calls add file for every IP using the variables

- ipdb.generate_vivado_inc_dirs
  - variable definition for all include directories
IPApprox Development Flow

Toplevel Modifications

1. If modification is in the toplevel, just update src_files.yml in RTL directory.

Independent Sub-IP Modifications

1. modify the IP in the checked out IPs directory on a new feature branch
2. Change the version in the dependent IPs to the new feature branch
3. If you add new dependencies, you have to commit and push the changes to the ips_list.yml
4. Run update-ips to resolve newly added dependencies and generate the new tcl files for simulation and synthesis
5. Once your changes are stable, commit and tag them and change version in all dependent packages to the new commit/release tag
IPApprox Exercise – Integration of a Dummy VIP

Time to try it yourself:

- In this little exercise you are going to practice the IP integration flow using IPApprox

- Source files and Exercise description on: https://github.com/pulp-training/dummy_vip
IP Dependency Management (Bender)

- Transition planned in the next couple of weeks
- Written in Rust
- Better Documentation
- More stable dependency resolution and conflict management
- (Yet) not flexible enough for subrepo flow used for pulp_soc
Simulation and Debug Flow
Build Flow of RTL Platform

make build VERBOSE=1

cd sim
make lib build opt

execute as tcl shell script

tcl_files/rtl_vopt.tcl

compiled source file into individual libraries

vcompile/ips.mk

vlib & vmap (create new libraries)

vopt_tb

optimized into

generated by IPApprox

references IP library list

print vlog commands for debugging
Simulation Invocation

make run gui=1 io=uart

VSIM_PATH
(points to pulpissimo/sim)

VSIM_FLAGS
VSIM_RUNNER_FLAGS
(read env variables)

sim/tcl_files/config/run_and_exit.tcl
(contains TCL function to check exit code of simulation)

sim/tcl_files/run.tcl
(source)

vsim

-invokes vsim (depends on SDK used)

optionaly start simulation in GUI mode

optionaly use UART output instead of virtual printf

set flags for simulation and generate TB args

start actual simulation (vsim vopt_tb <args>
Final Exercise/Homework 😊

- Development of pulp-runtime application (driver interaction)
- Training of RTL debugging skills around PULPissimo
- This is a more involved exercise and requires some code exploration skills. Don’t hesitate to ask if you have troubles.
We would appreciate your Feedback!

- Please let us know what you thought of this first training day by filling the feedback form below:

PULP Training Day 2

Robert Balas <balasr@iis.ee.ethz.ch>
Manuel Eggimann <meggimann@iis.ee.ethz.ch>
Programm of Day 2

Day 2

- FPGA Port
- PULP IP Landscape
- PULPIssimo Memory Layout Modification
- Hands-on Full-stack IP Integration Exercise
FPGA Port
Folder Structure

- fpga
  - pulpissimo (contains auto generated tcl script form IPApprox)
  - pulpissimo-<fpga-board>
    - rtl (port specific source files, i.e. wrappers for block RAM or clock managers)
    - ips
      - <ips_instantied by wrappers in rtl dir>
        - tcl (contains script to generate IP independent of PULPissimo)
FPGA Bitstream Generation

Contains variables for IP Generation e.g. PLL output frequency, block RAM Size.
Must match description in RTL and SDK!

- PLL for peripheral/SoC clock
- block RAM for memory banks

Generated by IPApprox
ips_add_files.tcl

build IPs needed for PULPissimo
ips/xxx/Makefile

generate IP
read variables

generate bitstream
tcl/run.tcl

read constraints (I/O mapping, multi-cycle paths)
constraintd/ <board>.sdc

board specific

PULP Training
## FPGA Synthesis Flow - Wrappers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xilinx_pulpissimo.sv</td>
<td>Toplevel wrapper of the whole pulpissimo. Converts differential clock input to single ended clock for pad_xtal_in.</td>
</tr>
<tr>
<td>pulp_clock_gatinx_xilinx.sv</td>
<td>Dummy clock gating cell. <strong>Must be replaced with a correct implementation for peripherals to work.</strong></td>
</tr>
<tr>
<td>pad_functional_xilinx.sv</td>
<td>Wrapper for IOBUF cells</td>
</tr>
<tr>
<td>fpga_bootrom.sv</td>
<td>Dummy implementation. Hardwired to always respond with jal x0,0</td>
</tr>
<tr>
<td>fpga_clk_gen.sv</td>
<td>Wrapper for Xilinx clock manager generates soc_clk and periph_clk from reference clock. <strong>Not at-runtime configurable</strong></td>
</tr>
<tr>
<td>fpga_interleaved_ram.sv</td>
<td>Wrapper for blockram macro generated by IP make targets</td>
</tr>
<tr>
<td>fpga_private.sv</td>
<td>Same as above</td>
</tr>
<tr>
<td>fpga_slow_clk_gen.sv</td>
<td>Certain FPGA boards have extremely high input frequencies (e.g. Genesys2). This wrapper instantiates PLL to slow down to intermediate freq (256*32768Hz) and feeds it to divide by 256 clock divider. Xilinx clock managers cannot go slower than 4.69MHz, that’s why.</td>
</tr>
</tbody>
</table>
FPGA Simulation Tipps

- Use ILA Cores on bus signals
- Problems of ILA cores and Genesys2 board (jtag frequency)
Techcells

- Contain technology dependent cells like clock gates for manual instantiation in design
- Must be replaced with tech-specific module implementations that internally instantiate the library cells
- I.e. create a new IP with the replaced modules that depends on techcells_generic (this forces correct compile-order and module override behavior)
Techcells

- Generic SRAM
- Clk AND
- Clk Buffer
- Glitch-free Clk Gate
- Clk Inverter
- Clk Mux (clk bypass)
- Clk Xor (freq. doubler)
- Programmable Clk Delay (glitch filtering)
Common Cells

- Contains commonly used high-level modules that are independent of technology
- Contains Verilog Macros for uniform declaration of registers
- Contains a couple of commonly used assertion macros
Common Cells

Clk Divider, Rst Synchronizer

CDC Crossings
- Gray FIFO
- 2phase HS
- Edge Detector
- Serial Synchronizers

Counters
- Delta Counter
- LFSRs

Datapath Elements
- Address Decoder (Heavily used in soc_interconnect)
- ECC Decoder/Encoder
- Gray2Binary/Binary2Gray
- Leading Zero Counter
- Stream (ready-valid pipeline) Building Blocks

Data Structures
- Counting Bloom Filter
- FIFO
- SRAM Behavioral
- Pseudo Least Recently Used Tree
AXI IPs

- Width Converters
- Mux/Demux
- Protocol Converters (APB, AXI-Lite, Atomics Filter)
- Fully-Connected XBAR (AXI or AXI-Lite)
- Clock Domain Crossings
- Pipeline Regs
- Burst Splitter
- Slave Isolator
- Address Rewriter
Exercise Time

- Clone/Pull the latest changes of the exercise repo (there are new changes since yesterday): [https://github.com/pulp-training/sw](https://github.com/pulp-training/sw)
- Switch to the `memlayout-exercise` and follow the instructions on: [https://github.com/pulp-training/sw/tree/main/memlayout-exercise](https://github.com/pulp-training/sw/tree/main/memlayout-exercise)
- Join a breakout room of your choice
- In case you need help or have a question, visit: [https://bit.ly/37nnl8P](https://bit.ly/37nnl8P) and enqueue yourself
- If you cannot use Zoom to share your screen or have issues with it: [https://fisch.ddns.net/call/mhq9864w](https://fisch.ddns.net/call/mhq9864w)
- Consult [https://fisch.ddns.net/sites/pulp_training](https://fisch.ddns.net/sites/pulp_training) for SoC schematics and FAQ (hopefully we have time to update it adhoc)

**We will continue at 13:15**
Full-stack AXI IP Integration

1. Write memory map description of IP in HJSON
2. Generate register-file using reggen
3. Develop wrapper that instantiates reg-file, IP, protocol converters and (if at all necessary) additional glue logic
4. Package and register IP using IPApprox
5. Instantiate wrapped IP in pulpissimo, modify soc_interconnect_wrap.sv, soc_mem_map.svh
6. Generate header file and develop driver
7. Test integration in RTL Simulation
Exercise Time

- Clone/Pull the latest changes of the exercise repo (there are new changes since yesterday): https://github.com/pulp-training/sw
- Open the exercise instructions on Github: https://github.com/pulp-training/sw/tree/main/fullstack_ip_integration
- Join a breakout room of your choice
- In case you need help or have a question, visit: https://bit.ly/37nnI8P and enqueue yourself
- If you cannot use Zoom to share your screen or have issues with it: https://fisch.ddns.net/call/mhq9864w
- Consult https://fisch.ddns.net/sites/pulp_training for SoC schematics and FAQ (hopefully we have time to update it adhoc)

- We will wrap things up at 17:37