Conquering complexity with Tcl and Make

Pavel Demin

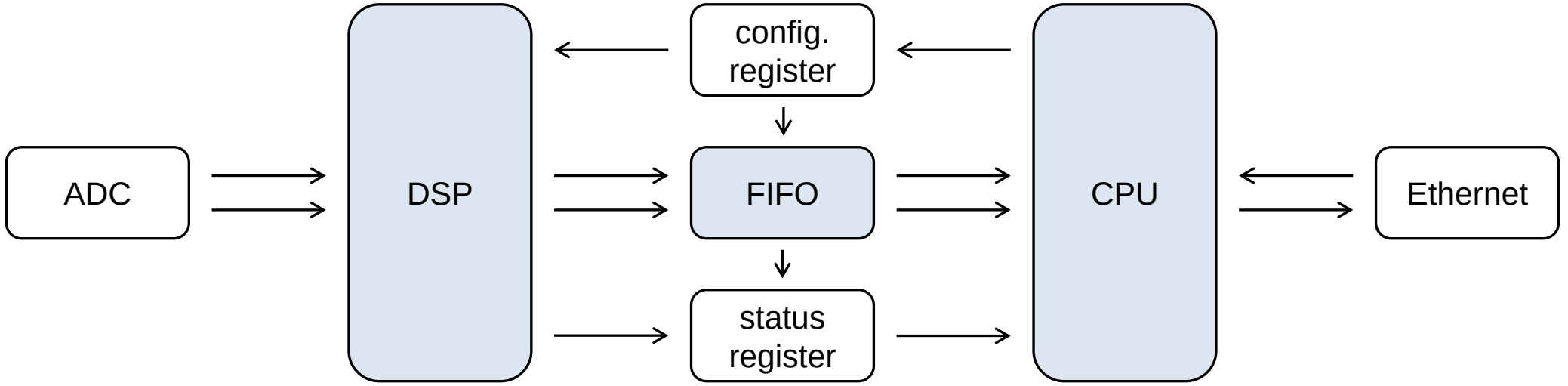
CLUB VIVADO users group

Agenda

- What am I doing with Vivado and AP SoC?
- Sources of complexity
- How am I conquering complexity?
- Examples and summary

What am I doing with Vivado and AP SoC?

Typical test and measurement application

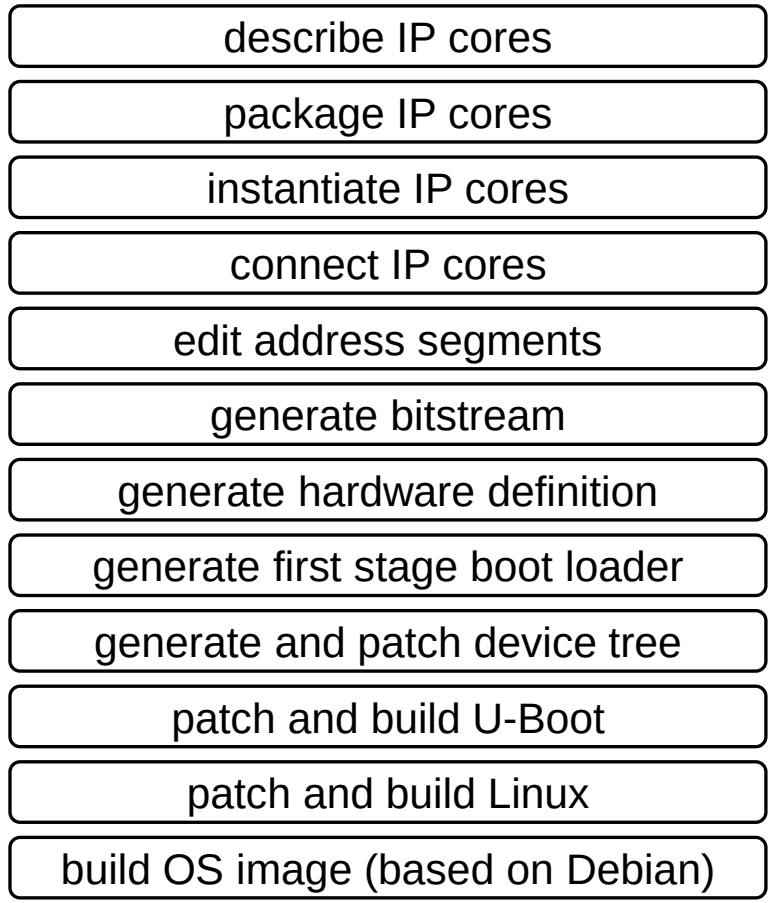


- DSP and math
  - Vivado provides rich library of IP cores
- Interface with custom IP cores
  - Vivado provides AXI-4, AXI4-Lite and AXI4-Stream infrastructure
- Connectivity
  - ARMv7-A + Linux = lots of communication options (Gigabit Ethernet, Wi-Fi, 4G, Bluetooth, …)

Sources of complexity

Sources of complexity (1/3)

- Long development chain
  - like more than 10 steps from IP core to OS image



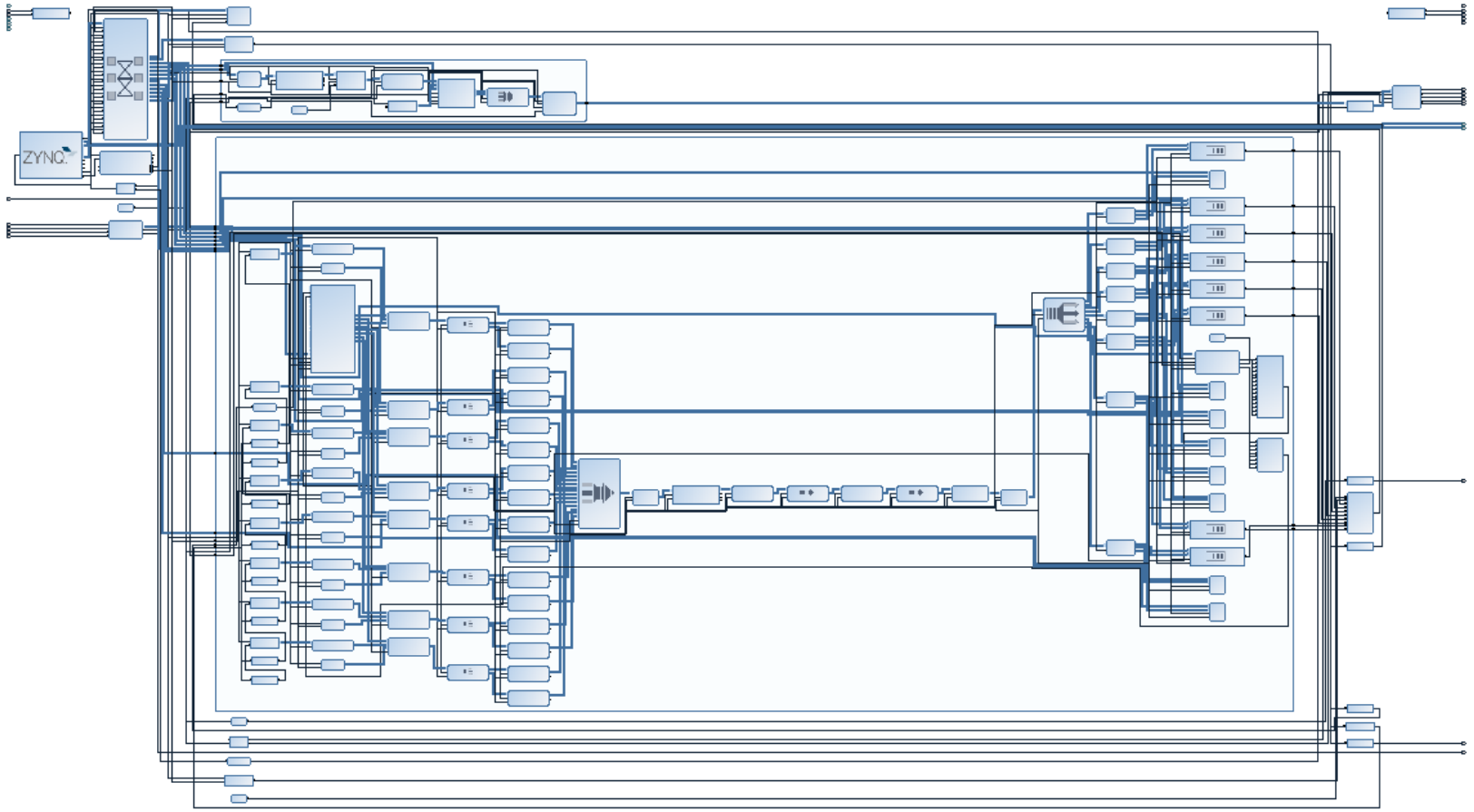
Sources of complexity (2/3)

- Verbose frequently used commands
  - like more than one Tcl command to describe IP core's parameter

```
set core [ipx::current_core]
set name AXI_DATA_WIDTH
set display_name {AXI DATA WIDTH}
set description {Width of the AXI data bus.}
set parameter [ipx::get_user_parameters $name -of_objects $core]
set_property DISPLAY_NAME $display_name $parameter
set_property DESCRIPTION $description $parameter
set parameter [ipgui::get_guiparamspec -name $name -component $core]
set_property DISPLAY_NAME $display_name $parameter
set_property TOOLTIP $description $parameter
```

Sources of complexity (3/3)

- Complex block designs
  - like more than 100 IP cores



How am I conquering complexity?

How am I conquering complexity?

- Long development chain
  - ➔ automate all steps with simple Tcl and Make scripts
- Verbose frequently used commands
  - ➔ define less verbose commands (Tcl procedures)
- Complex block designs
  - ➔ use power of Tcl to create block designs

Automate all steps with simple Tcl and Make scripts

	scripts	arguments		
package IP cores	**core.tcl**	dna_reader_v1_0	xc7z010clg400-1	
instantiate IP cores connect IP cores edit address segments	**project.tcl**	led_blinker	xc7z010clg400-1	
generate bitstream	**bitstream.tcl**	led_blinker		
generate hardware definition	**hwdef.tcl**	led_blinker		
generate first stage boot loader	**fsbl.tcl**	led_blinker	ps7_cortexa9_0	
generate and patch device tree	**devicetree.tcl**	led_blinker	ps7_cortexa9_0	device-tree-xlnx-v2016.2
patch and build U-Boot patch and build Linux	**Makefile**	NAME=led_blinker	all	
build OS image (based on Debian)	**image.sh**	**debian.sh**	led-blinker.img	

- 9 scripts, less than 500 lines of code (Tcl: 150, Make: 116, Shell: 221)

Define less verbose commands (Tcl procedures)

- Just 3 helper commands
  - describe IP core's parameters

```
# core_parameter parameter_name display_name description
core_parameter AXI_DATA_WIDTH {AXI DATA WIDTH} {Width of the AXI data bus.}
```

  - instantiate and configure IP cores

```
# cell core_name cell_name parameters connections
cell xilinx.com:ip:c_counter_binary:12.0 cntr_0 {CE true} {CLK ps_0/FCLK_CLK0}
```

  - create hierarchical modules

```
# module module_name body connections
module rx_0 {source projects/pulsed_nmr/rx.tcl} {fifo_0/S_AXIS adc_0/M_AXIS}
```

Use power of Tcl to describe block designs

- Let's compare some frequently used functionality provided by Vivado

functionality	Verilog	Tcl
describe IP cores	✓	×
package IP cores	×	✓
instantiate IP cores	×	✓
connect IP cores	✓	✓
bus interface support	×/✓(SystemVerilog)	✓
run automation rules	×	✓
edit address segments	×	✓

- This comparison suggests the following approach
  - → use **Verilog** to describe custom IP cores
  - → use **Tcl** to do all other steps

Examples (putting it all together)

Example of directory structure

- Source code is stored in a Git repository

  github.com/pavel-demin/red-pitaya-notes

- Repository contains

  cfg: constraints and board definition files

  cores: IP cores written in Verilog

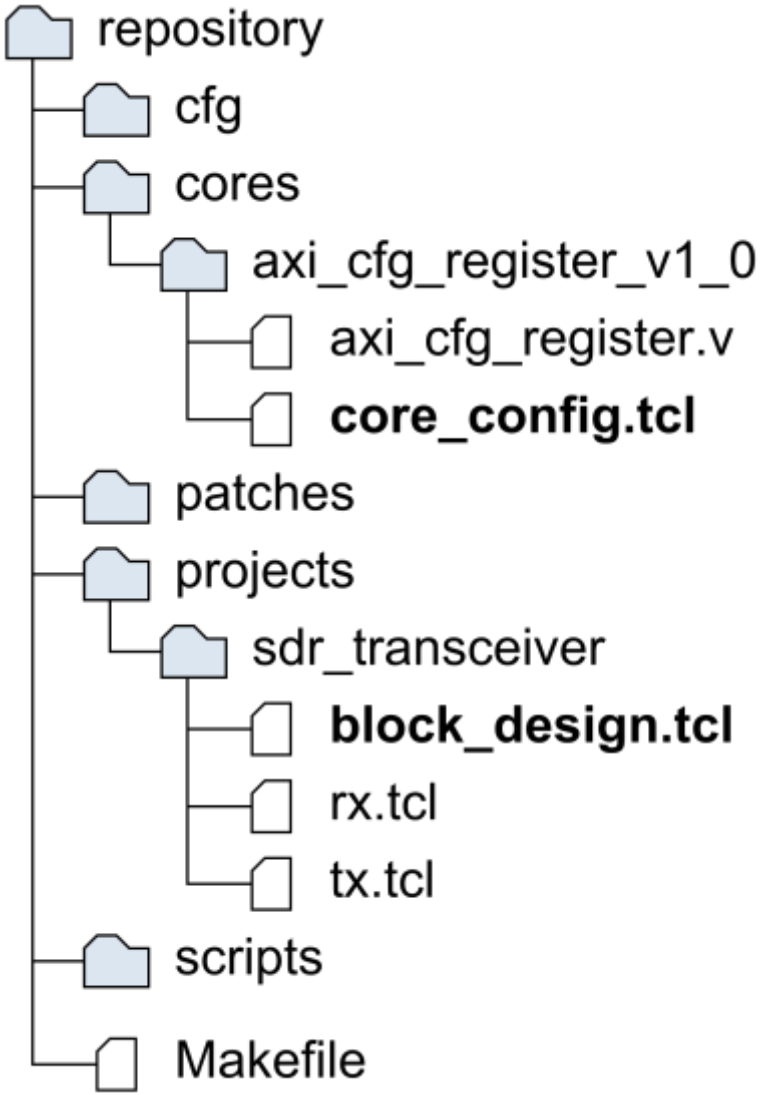
  patches: patches for device tree, U-Boot and Linux

  projects: Vivado projects written in Tcl

  scripts:

  Tcl scripts for Vivado and HSI

  shell scripts that build OS images



Example of core_config.tcl

```
# set display name and description
set display_name {AXI Configuration Register}
set core [ipx::current_core]
set_property DISPLAY_NAME $display_name $core
set_property DESCRIPTION $display_name $core
# set display name and description for parameters
core_parameter AXI_DATA_WIDTH {AXI DATA WIDTH} {Width of the AXI data bus.}
core_parameter AXI_ADDR_WIDTH {AXI ADDR WIDTH} {Width of the AXI address bus.}
core_parameter CFG_DATA_WIDTH {CFG DATA WIDTH} {Width of the configuration data.}
# set S_AXI mode
set bus [ipx::get_bus_interfaces -of_objects $core s_axi]
set_property NAME S_AXI $bus
set_property INTERFACE_MODE slave $bus
# associate S_AXI with aclk
set bus [ipx::get_bus_interfaces aclk]
set parameter [ipx::get_bus_parameters -of_objects $bus ASSOCIATED_BUSIF]
set_property VALUE S_AXI $parameter
```

Example of block_design.tcl

```
# create processing_system7
cell xilinx.com:ip:processing_system7:5.5 ps_0 {
  PCW_IMPORT_BOARD_PRESET cfg/red_pitaya.xml
} {
  M_AXI_GP0_ACLK ps_0/FCLK_CLK0
}
# create all required interconnections
apply_bd_automation -rule xilinx.com:bd_rule:processing_system7 -config {
  make_external {FIXED_IO, DDR}
  Master Disable
  Slave Disable
} [get_bd_cells ps_0]
# create c_counter_binary
cell xilinx.com:ip:c_counter_binary:12.0 cntr_0 {
  OUTPUT_WIDTH 32
} {
  CLK ps_0/FCLK_CLK0
}
```

Example of workflow

- create or update IP core
  - describe IP core using Verilog
  - set IP core's properties via core_config.tcl
- create or update project
  - instantiate and connect IP cores via block_design.tcl
- build bitstream

```
make NAME=project_name clean
make NAME=project_name bit
```

- build OS image

```
make NAME=project_name all
sudo sh scripts/image.sh scripts/debian.sh os.img
```

Example of verification flow

- build Vivado project

```
make NAME=project_name xpr
```

- open project in Vivado

```
vivado tmp/project_name.xpr
```

- interactively analyze project in Vivado (UG900, UG906)

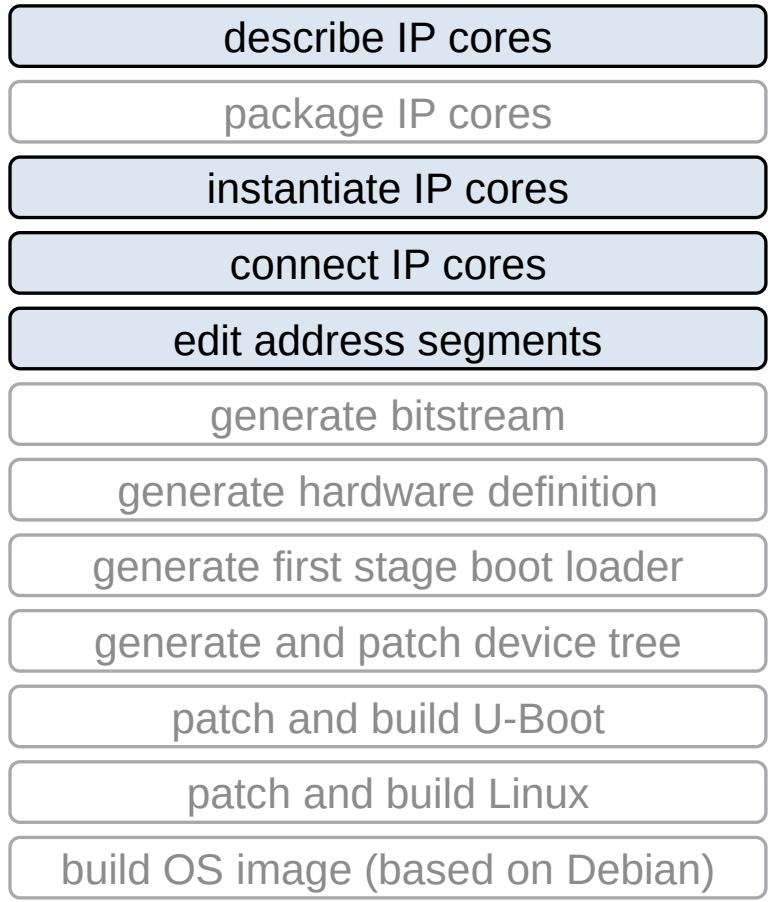
- all Vivado verification flows and methodologies can be used

Summary

Summary

- Thanks to Vivado scriptability it's very easy to define custom toolchains
- My toolchain helps me to
  - focus on IP-centric design flow
  - work with readable source code
  - effectively use revision control systems
  - automate frequent tasks
  - build everything with just a few commands
- My code can be found at

  github.com/pavel-demin/red-pitaya-notes



Interesting links

- Vivado Design Suite Tcl Command Reference Guide (UG835)
- Using Tcl Scripting (UG894)
- Designing with IP (UG896)
- Designing IP Subsystems Using IP Integrator (UG994)
- Creating and Packaging Custom IP (UG1118)
- Generating Basic Software Platforms (UG1138)
- Koheron SDK – FPGA design with Tcl (more helper commands)

XILINX
ALL PROGRAMMABLE™


facebook.com/XilinxInc


twitter.com/#!/XilinxInc

youtube.com/XilinxInc


linkedin.com/company/xilinx


plus.google.com/+Xilinx

xilinx.com/about/app-download.html


XILINX
ALL PROGRAMMABLE™