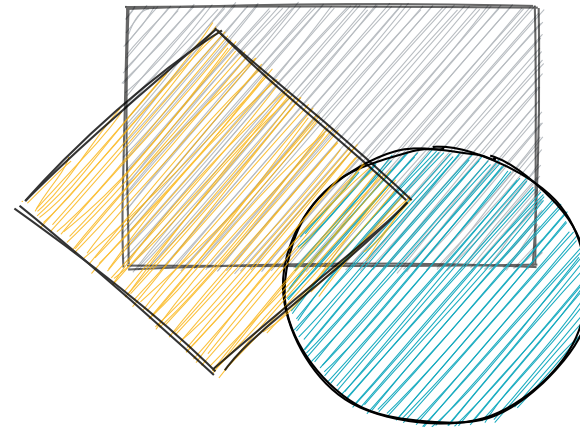


FWK - The DESY FPGA Framework

An open-source FPGA framework by DESY for large scientific projects

Michael Büchler
On behalf of the DESY MSK Firmware Team
Beijing, 2024-09-11

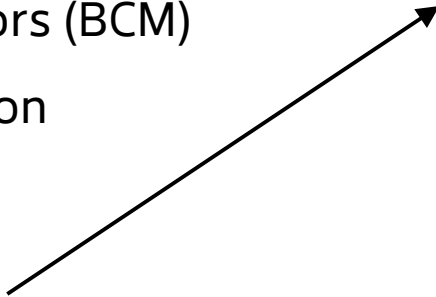


FWK
FPGA firmware framework

Firmware Framework

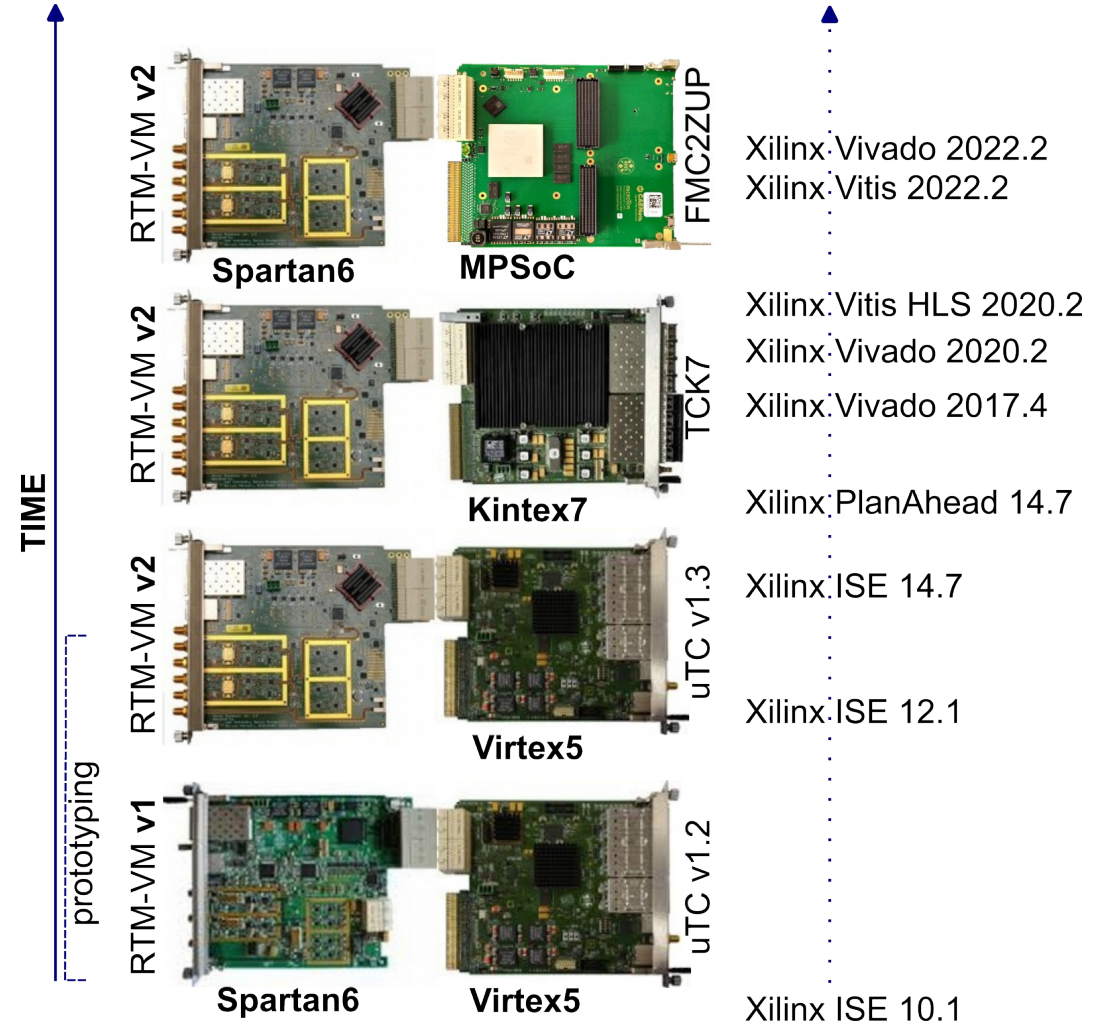
Challenges in FPGA Firmware Development at DESY

- FPGA firmware team at DESY MSK (accelerator beam controls) group
- FLASH and European XFEL facilities, upcoming PETRA IV, and many more
- Various firmware projects:
 - Low-Level RF (LLRF)
 - Beam Arrival Monitor (BAM)
 - Bunch Compression Monitors (BCM)
 - Laser-based Synchronisation
 - ...
- Mostly on μ TCA.4 hardware



EuXFEL LLRF controller example on MTCA.4

same application – different hardware – different tool



Firmware Framework

Challenges in FPGA Firmware Development at DESY

- Long-term support of installations
- Change of hardware and developer tools over the years
- Many projects, small team size

Same FPGA board - same application ***Different use case***

>90% of the code is the same but different FPGA binary file
Different: clock frequency, buffer size, IQ demodulation..



DWC8VM1

SIS8300L2

Single Cavity Controller

→ 10 different configurations

Same FPGA board - different application

- LLRF field detection
- Laser lock controller
- Single Cavity GDR controller
- Single Cavity SEL controller
- Bunch Compression Monitors



DWC10



DWC8VM1



SIS8900



SIS8300L2

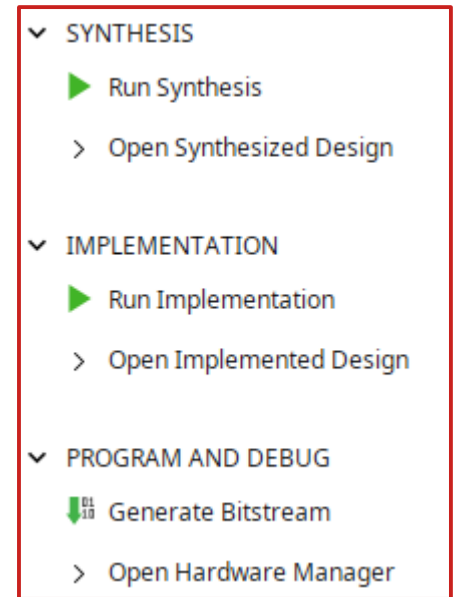
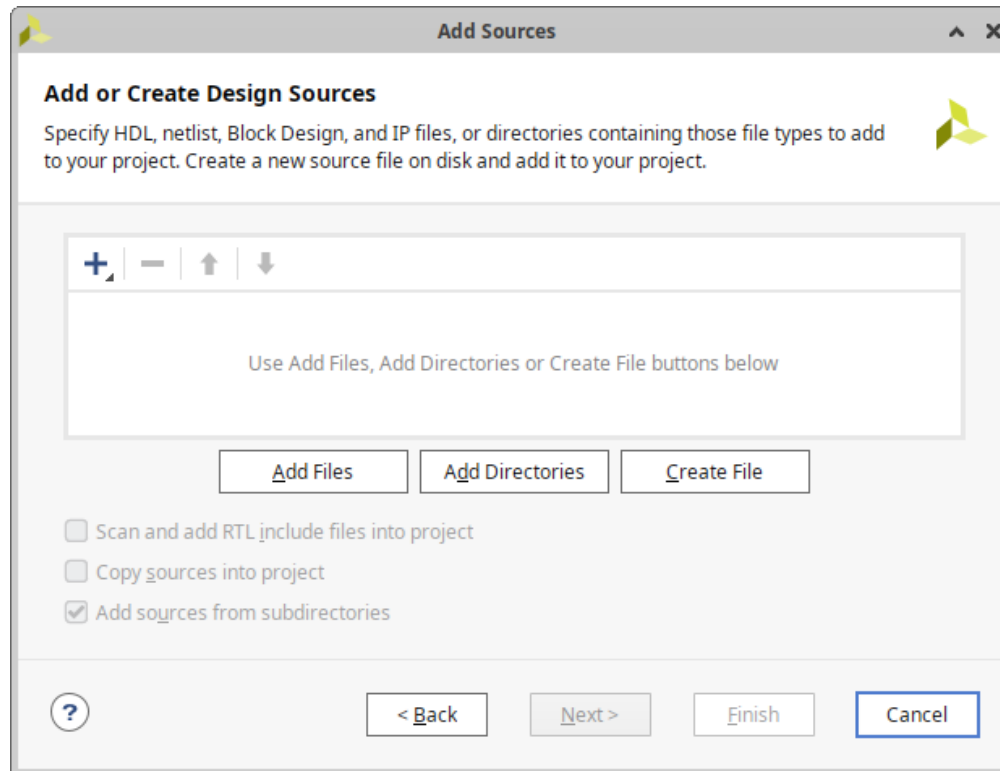
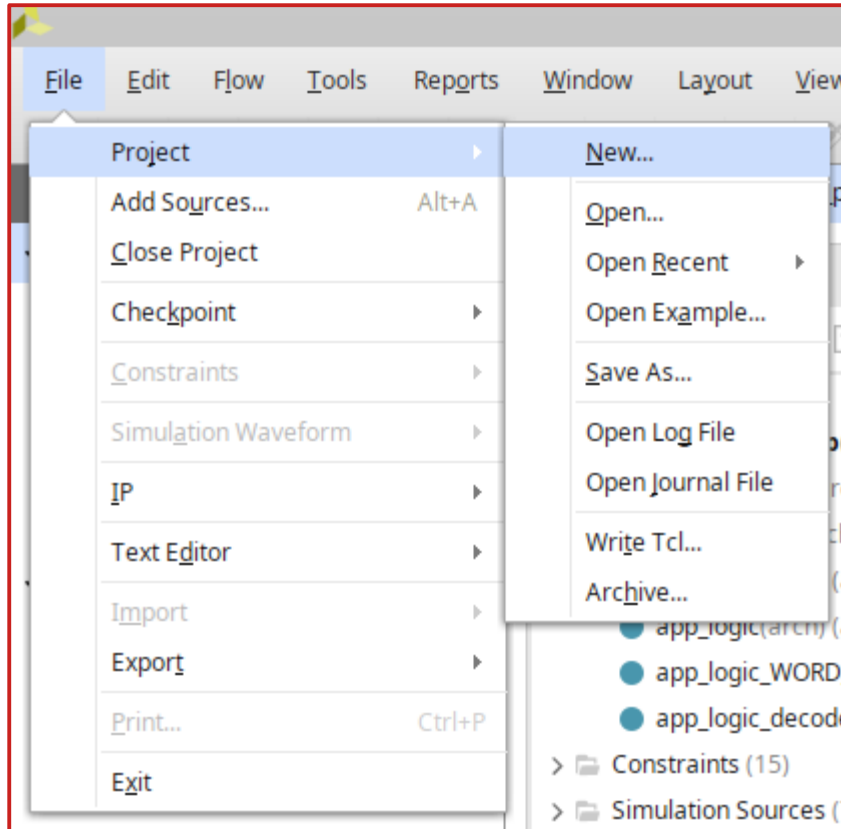
Firmware Framework

Basic firmware development with AMD/Xilinx Vivado



→ Deploy

→ Store project on a network drive





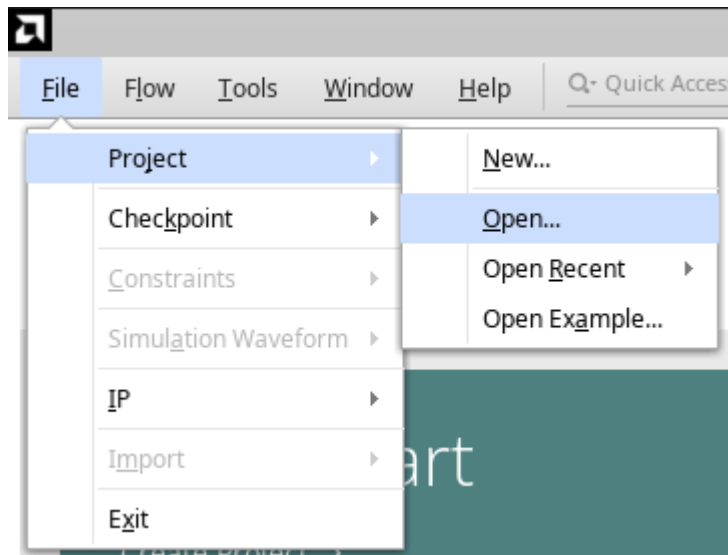
**TWO YEARS
LATER...**

Firmware Framework

New Vivado version, new developer, new requirements

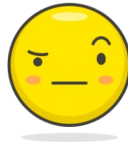


"click-click"



? This project was created using an older version of Vivado. It is strongly recommended that you backup the project prior to migration by opening the project in the older version of Vivado (v2022.2) and running "File->Archive Project". This will ensure all sources, including any IPs used by the design, are preserved properly. Choose to either automatically upgrade to the current version or open the project in read-only mode.


Problems



- + Tool versions
- + Code copy & paste between projects
- + Pre-generated or hand-written control and status registers (CSRs)
- + ...

A Better Solution

Overview of goals

- Allow full automation on a build server → CI 
 - reproducibility
- Abstraction of tool type and version
 - common way to build a project
- Modularity of the code
 - reusability
 - co-development
- Integration of HDL, HLS, Embedded C/C++, and Embedded Linux
- Generation of some source files
 - Address space
 - Version information
- Tcl language to keep flexibility

Benefits

- + Long-term maintainability
- + Improved collaborative development
- + Simplified management of multiple projects
- + Reproducible results

Drawbacks

- Initial learning effort
- Potentially limited flexibility
- Burden of developing & maintaining the framework

History

- Started in 2013 (monolithic code base, still SVN)
- Released under the Apache 2.0 license in 2022

FPGA Firmware Framework (FWK)

Key aspects

Tcl scripts

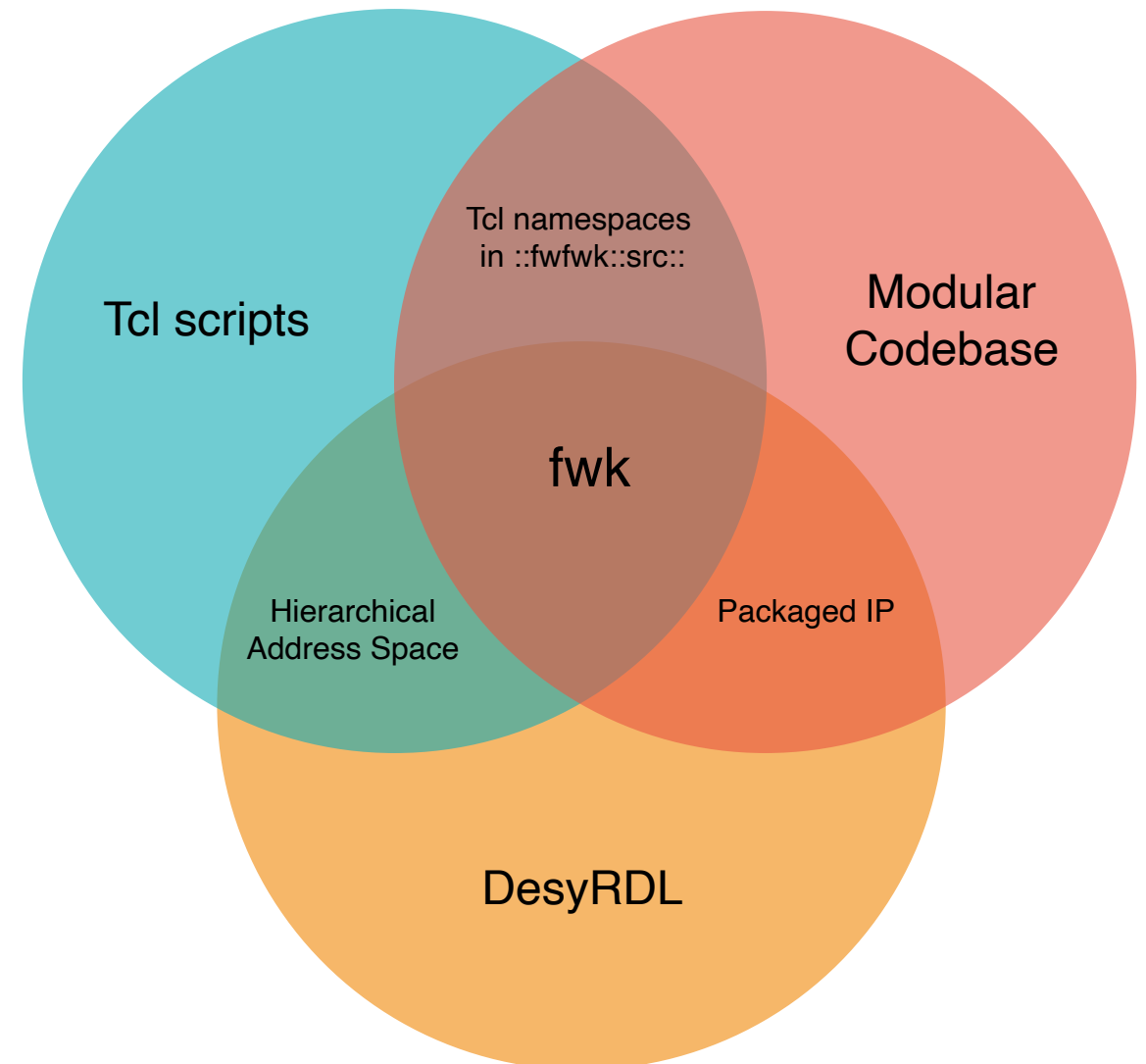
- Interaction with tools abstracted away from the dev
- Automation of tasks

Modular Codebase

- Integration with FWK through a single Tcl file
- May be organized as Git submodules
- Can be packaged as Vivado IP

Address Space

- Defined by the modules using SystemRDL
(or other formats → converted to SystemRDL)
- FWK creates a hierarchy, DesyRDL generates output products (VHDL, C headers, "mapfiles" for ChimeraTK, ..)



FPGA Firmware Framework (FWK)

The Flow, common to all projects

Three simple steps to keep in mind:



Example FWK execution

Unified execution flow

clone repository

```
git clone git@ ... && cd repo
```

clone all submodules repositories

```
git submodules update --init --recursive
```

create virtual environment with tools

```
make env
```

source the Vivado environment

```
source /opt/Xilinx/Vivado/2022.2/settings64.sh
```

create project

```
make cfg=dwc8vm1 project
```

build the project -> generate artifacts

```
make cfg=dwc8vm1 build
```

build documentation

```
make cfg=dwc8vm1 doc
```

open tool gui for design analysis/debug

```
make cfg=dwc8vm1 gui
```

```
src/hw/daq 2.0.0-9-geccdf66 ()
src/hw/desy_vhdl 0.0.0-321-g0f318cc5 ()
src/hw/fpga_config_manager 2.1.0-4-g0b9b99c ()
src/hw/lll 0.0.0-6-g0d9f400b ()
src/hw/rtm_ds8vm1 1.1.1-12-gacf6763 ()
src/hw/rtm_dwc10 2.1.0-1-gffac773 ()
src/hw/rtm_dwc8vm1 2.0.0-9-g3574e72 ()
src/hw/timing 2.0.3-1-g00790c7 ()
-----
== INIT ==
# INFO: Creating namespace ::fwfwk::src
# INFO: Creating namespace ::fwfwk::src::bsp
# INFO: parsing vhdl configuration file /home/mbuechl/src/fpgafw/sis8300ku_example_design/src/
app_example/hdl/pkg_bsp_sis8300ku_config_1.vhd into ::fwfwk::src::bsp::Config
# INFO: Creating namespace ::fwfwk::src::bsp::fpga_config_manager
# INFO: Creating namespace ::fwfwk::src::app
# INFO: parsing vhdl configuration file ../hdl/pkg_app_config_conf_2.vhd into ::fwfwk::src::app
onfig
# INFO: Creating namespace ::fwfwk::src::app::daq
# INFO: Creating namespace ::fwfwk::src::app::timing
# INFO: Creating namespace ::fwfwk::src::app::rtm_dwc8vm1
-----
# INFO: Tree View of the Sources:
+- src
+- app
+- daq
+- rtm_dwc8vm1
+- timing
+- bsp
+- fpga_config_manager
-----
== CREATE ==
# INFO: Creating ( sis8300ku_example_design_dwc8vm1 ) vivado project ...
# INFO: in /home/mbuechl/src/fpgafw/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2022.2/data/ip'.
# INFO: executing ::fwfwk::src::bsp::setPrjProperties
# INFO: Setting FPGA Ordering Code: xcku040-ffva1156-1-c
# INFO: FPGA Family: Kintex Ultrascale
# INFO: Number of logic cells: 40k
# INFO: Package: ffva1156
# INFO: Speed Grade: -1
# INFO: Temperature Grade: -C (Commercial)
-----
== GEN/ADD ADDRESS SPACE ==
# INFO: executing ::fwfwk::src::app::rtm_dwc8vm1::setAddressSpace
# INFO: executing ::fwfwk::src::app::timing::setAddressSpace
# INFO: executing ::fwfwk::src::app::daq::setAddressSpace
# INFO: executing ::fwfwk::src::app::setAddressSpace
# INFO: executing ::fwfwk::src::bsp::fpga_config_manager::setAddressSpace
# INFO: executing ::fwfwk::src::bsp::setAddressSpace
# INFO: executing ::fwfwk::src::setAddressSpace
-----
```

Example FWK execution

Unified execution flow

clone repository

```
git clone git@ ... && cd repo
```

clone all submodules repositories

```
git submodule update --init --recursive
```

create virtual environment with tools

```
make env
```

source the Vivado environment

```
source /opt/Xilinx/Vivado/2022.2/settings64.sh
```

create project

```
make cfg=dwc8vm1 project
```

build the project -> generate artifacts

```
make cfg=dwc8vm1 build
```

build documentation

```
make cfg=dwc8vm1 doc
```

open tool gui for design analysis/debug

```
make cfg=dwc8vm1 gui
```

```
value to ECC_NONE.
Wrote : </home/mbuechl/src/fpga/w/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8vm1/sis8300ku_example_design_dwc8vm1.srcs/sources_1/bd/sis8300ku_bsp_system/sis8300ku_bsp_system.bd>
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /ilmb_cntlr/SLMB/B/Mem'
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /second_ilmb_cntlr/B/Mem'
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /dlmb_cntlr/SLMB/B/Mem'
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /second_dlmb_cntlr/B/Mem'
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /iomodule_0/SLMB/B/Mem'
WARNING: [BD 5-699] No address segments matched 'get_bd_addr_segs -of object /iomodule_0/SLMB/B/Mem'
INFO: [xilinx.com:ip:smartconnect:1.0-1] sis8300ku_bsp_system_axi_interconnect_dma_0: SmartConnect is in High-performance Mode.
INFO: [xilinx.com:ip:smartconnect:1.0-1] sis8300ku_bsp_system_axi_interconnect_ddr_0: SmartConnect is in High-performance Mode.
INFO: [xilinx.com:ip:smartconnect:1.0-1] sis8300ku_bsp_system_axi_interconnect_reg_0: SmartConnect is in Low-Area Mode.
WARNING: [xilinx.com:ip:smartconnect:1.0-1] sis8300ku_bsp_system_axi_interconnect_reg_0: IP sis8300ku_bsp_system_axi_interconnect_reg_0 is configured in Low-area mode as all propagated traffic w-bandwidth (AXI4LITE). SI S01_AXI has property HAS_BURST == 1. WRAP bursts are not supported w-area mode and will result in DECERR if received.
WARNING: [xilinx.com:ip:smartconnect:1.0-1] sis8300ku_bsp_system_axi_interconnect_reg_0: If WRANSATIONS are required then turn off Low-area mode using ADVANCED_PROPERTIES. Execute following property CONFIG.ADVANCED_PROPERTIES {__experimental_features__ {disable_low_area_mode 1}} [d_cells /sis8300ku_bsp_system_axi_interconnect_reg_0]
validate_bd_design: Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 2452.5 gain = 126.031 ; free physical = 38099 ; free virtual = 55187
INFO: [BD 41-1662] The design 'sis8300ku_bsp_system.bd' is already validated. Therefore parameter propagation will not be re-run.
WARNING: [BD 41-2671] The dangling interface net <m00_aw_node_M_AXIS_ARB> will not be written to the BD file.
WARNING: [BD 41-2671] The dangling interface net <m01_aw_node_M_AXIS_ARB> will not be written to the BD file.
WARNING: [filemgmt 56-443] The ECC Algorithm string is empty. Setting the Memory Map to default value to ECC_NONE.
WARNING: [filemgmt 56-443] The ECC Algorithm string is empty. Setting the Memory Map to default value to ECC_NONE.
Wrote : </home/mbuechl/src/fpga/w/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8vm1/sis8300ku_example_design_dwc8vm1.srcs/sources_1/bd/sis8300ku_bsp_system/sis8300ku_bsp_system.bd>
VHDL Output written to : /home/mbuechl/src/fpga/w/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8vm1/sis8300ku_example_design_dwc8vm1.gen/sources_1/bd/sis8300ku_bsp_system/synth/sis8300ku_bsp_system.vhd
VHDL Output written to : /home/mbuechl/src/fpga/w/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8vm1/sis8300ku_example_design_dwc8vm1.gen/sources_1/bd/sis8300ku_bsp_system/sim/sis8300ku_bsp_system.vhd
VHDL Output written to : /home/mbuechl/src/fpga/w/sis8300ku_example_design/prj/sis8300ku_example_design_dwc8vm1/sis8300ku_example_design_dwc8vm1.gen/sources_1/bd/sis8300ku_bsp_system/hdl/sis8300ku_bsp_system_wrapper.vhd
# INFO: executing ::fwfwk::src::doOnCreate
-----
# INFO: Project creation completed successfully

Exiting tool...
INFO: [Common 17-206] Exiting Vivado at Wed Sep 4 17:10:11 2024...
~/src/fpga/w/sis8300ku_example_design
```

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

```
# create project
```

```
make cfg=fractional_divider_signed project
```

```
# run the simulation
```

```
make cfg=fractional_divider_signed sim
```

```
# open resulting waveforms in GTKWave
```

```
make cfg=fractional_divider_signed gui
```

```
~/src/fpgafw/verif_desy_vhdl$ ls -l cfg
total 60
-rw-rw-r-- 1 mbuechl mbuechl 134 Sep 22 2022 ad56xx.cfg
-rw-rw-r-- 1 mbuechl mbuechl 134 Sep 22 2022 ad79xx.cfg
-rw-rw-r-- 1 mbuechl mbuechl 128 Sep 22 2022 cic.cfg
-rw-rw-r-- 1 mbuechl mbuechl 156 Sep 22 2022 common_logic_utils.cfg
-rw-rw-r-- 1 mbuechl mbuechl 127 Sep 22 2022 default.cfg
-rw-rw-r-- 1 mbuechl mbuechl 156 Jan 18 2023 divider_restoring.cfg
-rw-rw-r-- 1 mbuechl mbuechl 172 Sep 4 13:16 fractional_divider_signed.cfg
-rw-rw-r-- 1 mbuechl mbuechl 176 Sep 22 2022 fractional_divider_unsigned.cfg
-rw-rw-r-- 1 mbuechl mbuechl 130 Okt 18 2023 lfsr.cfg
-rw-rw-r-- 1 mbuechl mbuechl 134 Sep 22 2022 math_iq.cfg
-rw-rw-r-- 1 mbuechl mbuechl 142 Sep 22 2022 math_signed.cfg
-rw-rw-r-- 1 mbuechl mbuechl 146 Sep 22 2022 math_unsigned.cfg
-rw-rw-r-- 1 mbuechl mbuechl 140 Sep 22 2022 math_utils.cfg
-rw-rw-r-- 1 mbuechl mbuechl 207 Sep 2 21:55 trigger_delay.cfg
-rw-rw-r-- 1 mbuechl mbuechl 154 Sep 20 2023 xdma_irq_handler.cfg
```

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

```
make cfg=fractional_divider_signed gui
```

```
fractional_divider_signed.cfg
1 # project default configuration
2 projectName=verif_desy_vhdl
1 ProjectConf=fractional_divider_signed
2 ProjectTcl=tcl/fractional_divider_signed.tcl
3 AddrType=map
4 ToolType=cocotb

main  cfg  2:1 33%
```

The terminal rendering is missing some spaces, sorry!!

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

```
make cfg=fractional_divider_signed gui
```

```
fractional_divider_signed.cfg fractional_divider_signed.tcl
15 #####
14 # Main tcl for restoring divider verification project
13 #####
12
11 # =====
10 procinit {} {
9   }
8
7 # =====
6 procsetSources {} {
5
4   variableSources
3
2     lappendSources { "../src/desy_vhdl/hdl/common/pkg_common_logic_utils.vhd" "VHDL" "desy"}
1     lappendSources { "../src/desy_vhdl/hdl/math/pkg_math_utils.vhd" "VHDL" "desy"}
16 | lappendSources { "../src/desy_vhdl/hdl/math/pkg_math_unsigned.vhd" "VHDL" "desy"}
1     lappendSources { "../src/desy_vhdl/hdl/math/comp/fractional_divider_unsigned.vhd" "VHDL" "desy"}
2     lappendSources { "../src/desy_vhdl/hdl/math/comp/fractional_divider_signed.vhd" "VHDL" "desy"}
3   }
4
5 # =====
6 procsetAddressSpace {} {
7   }
8
9 # =====
10 procdoOnCreate {} {
11
12   variableSources
| main | tcl | TS 16:3 40%
```

The terminal rendering is missing some spaces, sorry!!

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

```
make cfg=fractional_divider_signed gui
```

```
fractional_divider_signed.cfg fractional_divider_signed.tcl
18 | lappendSources { "../src/desy_vhdl/hdl/common/pkg_common_logic_utils.vhd" "VHDL" "desy"}
17 | lappendSources { "../src/desy_vhdl/hdl/math/pkg_math_utils.vhd" "VHDL" "desy"}
16 | lappendSources { "../src/desy_vhdl/hdl/math/pkg_math_unsigned.vhd" "VHDL" "desy"}
15 | lappendSources { "../src/desy_vhdl/hdl/math/comp/fractional_divider_unsigned.vhd" "VHDL" "desy"}
14 | lappendSources { "../src/desy_vhdl/hdl/math/comp/fractional_divider_signed.vhd" "VHDL" "desy"}
13 | }
12 |
11 | # =====
10 | procsetAddressSpace {} {
9 | }
8 |
7 | # =====
6 | procdoOnCreate {} {
5 |
4 |     variableSources
3 |
2 |     addSourcesSources
1 |
32 | set::fwfwk::src::Top desy.fractional_divider_signed
1 | settb_path [filenormalize ../src/desy_vhdl/sim/math/fractional_divider/
   | tb_fractional_divider_signed.py]
2 | settb_path_d2 [filenormalize ../src/desy_vhdl/sim/math/fractional_divider/
   | tb_fractional_divider_signed_d2.py]
3 | lappend::fwfwk::src::SimTop "$tb_pathG_INPUT_LENGTH 18 G_OUTPUT_LENGTH 18"
4 | lappend::fwfwk::src::SimTop "$tb_path_d2G_INPUT_LENGTH 18 G_OUTPUT_LENGTH 18 G_SEPARATE_COMPARE
   | true"
6 |
|
| main tcl TS 32:54 80%
```

The terminal rendering is missing some spaces, sorry!!

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

```
make cfg=fractional_divider_signed gui
```

```
== SIMULATE ==  
  
> Running Makefile_tb_fractional_divider_signed  
  
# INFO: FILE: /home/mbuechl/src/fpgafw/verif_desy_vhdl/src/desy_vhdl/sim/math/fractional_divider/tb_fr  
onal_divider_signed.py  
make[1]: Entering directory '/home/mbuechl/src/fpgafw/verif_desy_vhdl/prj/verif_desy_vhdl_fractional_d  
r_signed'  
rm -f results.xml  
"make" -f Makefile_tb_fractional_divider_signed results.xml  
make[2]: Entering directory '/home/mbuechl/src/fpgafw/verif_desy_vhdl/prj/verif_desy_vhdl_fractional_d  
r_signed'  
mkdir -p sim_build  
/usr/local/bin/ghdl -i --std=93 --workdir=sim_build --work=desy /home/mbuechl/src/fpgafw/verif_desy_vh  
rc/desy_vhdl/hdl/common/pkg_common_logic_utils.vhd /home/mbuechl/src/fpgafw/verif_desy_vhdl/src/desy_v  
hdl/math/pkg_math_utils.vhd /home/mbuechl/src/fpgafw/verif_desy_vhdl/src/desy_vhdl/hdl/math/pkg_math_ur  
d.vhd /home/mbuechl/src/fpgafw/verif_desy_vhdl/src/desy_vhdl/hdl/math/comp/fractional_divider_unsigned  
/home/mbuechl/src/fpgafw/verif_desy_vhdl/src/desy_vhdl/hdl/math/comp/fractional_divider_signed.vhd &&  
/usr/local/bin/ghdl -i --std=93 --workdir=sim_build --work=work && \  
/usr/local/bin/ghdl -m --std=93 --workdir=sim_build -Psim_build --work=work desy.fractional_divider_si  
rm -f results.xml  
MODULE=tb_fractional_divider_signed TESTCASE= TOPLEVEL=desy.fractional_divider_signed TOPLEVEL_LANG=vh  
/usr/local/bin/ghdl -r --std=93 --time-resolution=ps --workdir=sim_build -Psim_build --work=work desy.  
tional_divider_signed --vpi=/home/mbuechl/src/fpgafw/verif_desy_vhdl/.venv/lib/python3.12/site-package  
cocotb/libs/libcocotbvpi_ghdl.so -gG_INPUT_LENGTH=18 -gG_OUTPUT_LENGTH=18 --wave=dump.ghw --vcd=dump.vcd  
loading VPI module '/home/mbuechl/src/fpgafw/verif_desy_vhdl/.venv/lib/python3.12/site-packages/cocotb  
/libcocotbvpi_ghdl.so'  
-.-ns INFO gpi .mbed/gpi_embed.cpp:108 in set_program_name_in_ve  
nv Using Python virtual environment interpreter at /home/mbuechl/src/fpgafw/verif_desy_vhdl/.venv/bi  
n/python
```

The terminal rendering is missing some spaces, sorry!!

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

```
make cfg=fractional_divider_signed gui
```

```
VPI module loaded!
0.00ns INFO cocotb Running on GHDL version 4.1.0 (4.1.0.r0.g7188e92cf)
[Dunoon edition]
0.00ns INFO cocotb Running tests with cocotb v1.9.1 from /home/mbuechl
/src/fpgafw/verif_desy_vhdl/.venv/lib/python3.12/site-packages/cocotb
0.00ns INFO cocotb Seeding Python random module with 1725459228
0.00ns INFO cocotb.regression pytest not found, install it to enable better Asser
tionError messages
0.00ns INFO cocotb.regression Found test tb_fractional_divider_signed.test_value_
over_value
0.00ns INFO cocotb.regression          running test_value_over_value (1/1)
Test val1 / val2
230120.00ns INFO cocotb.regression test_value_over_value          passed
230120.00ns INFO cocotb.regression *****
*****
** TEST
STATUS SIM TIME (ns) REAL TIME (s) RATIO (ns/s) **
*****
** tb_fractional_divider_signed.test_value_over_val
ue PASS 230120.00 2.64 87074.19 **
*****
** TESTS=1 PASS=1 FAIL=0 SKIP=0
230120.00 2.69 85473.80 **
*****
*****
make[2]: Leaving directory '/home/mbuechl/src/fpgafw/verif_desy_vhdl/prj/verif_desy_vhdl_fractional_d
signed'
```

The terminal rendering is missing some spaces, sorry!!

[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Example FWK execution

Verification example ¹

- GHDL as the simulator
- cocotb for writing the testbenches

create project

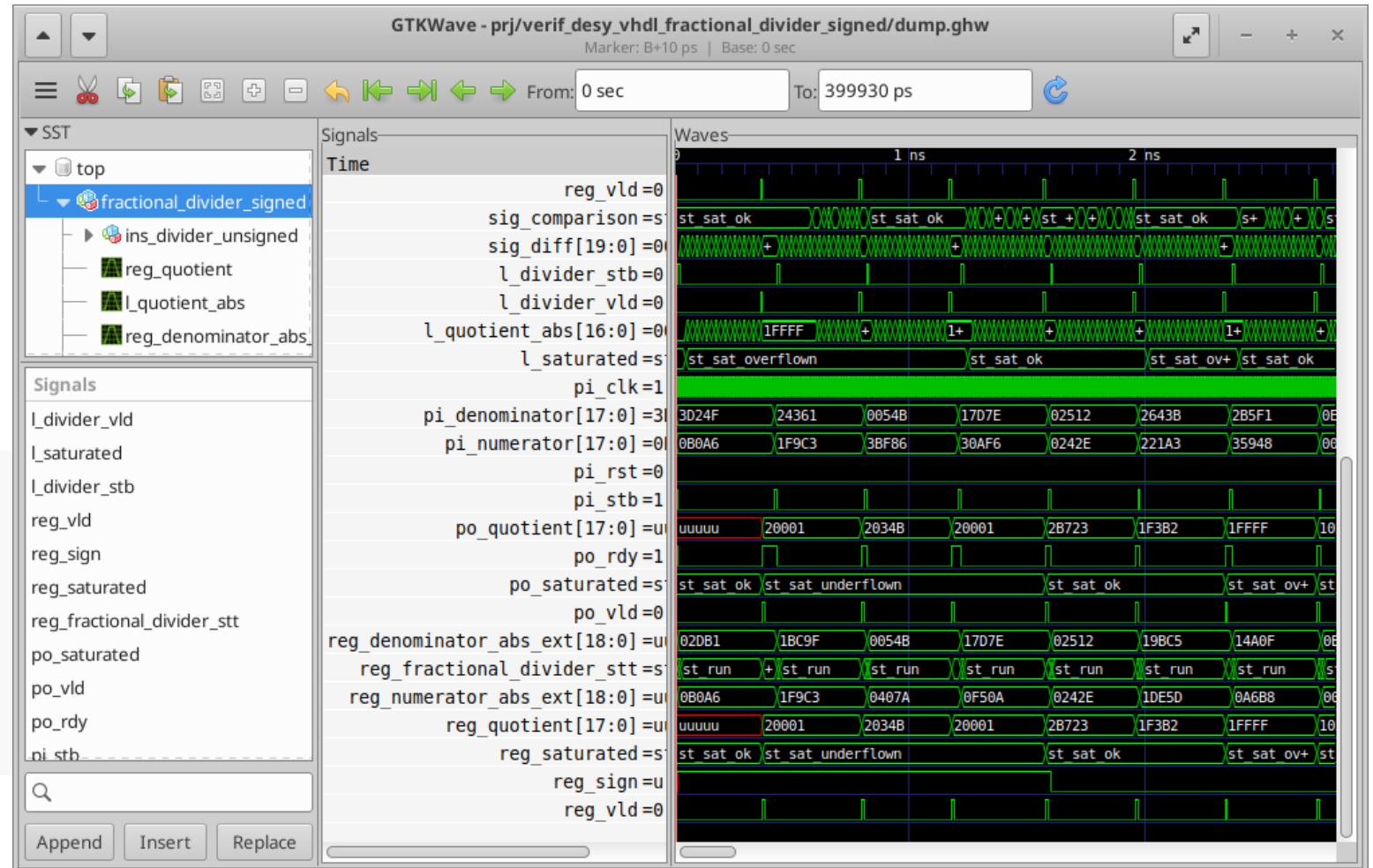
```
make cfg=fractional_divider_signed project
```

run the simulation

```
make cfg=fractional_divider_signed sim
```

open resulting waveforms in GTKWave

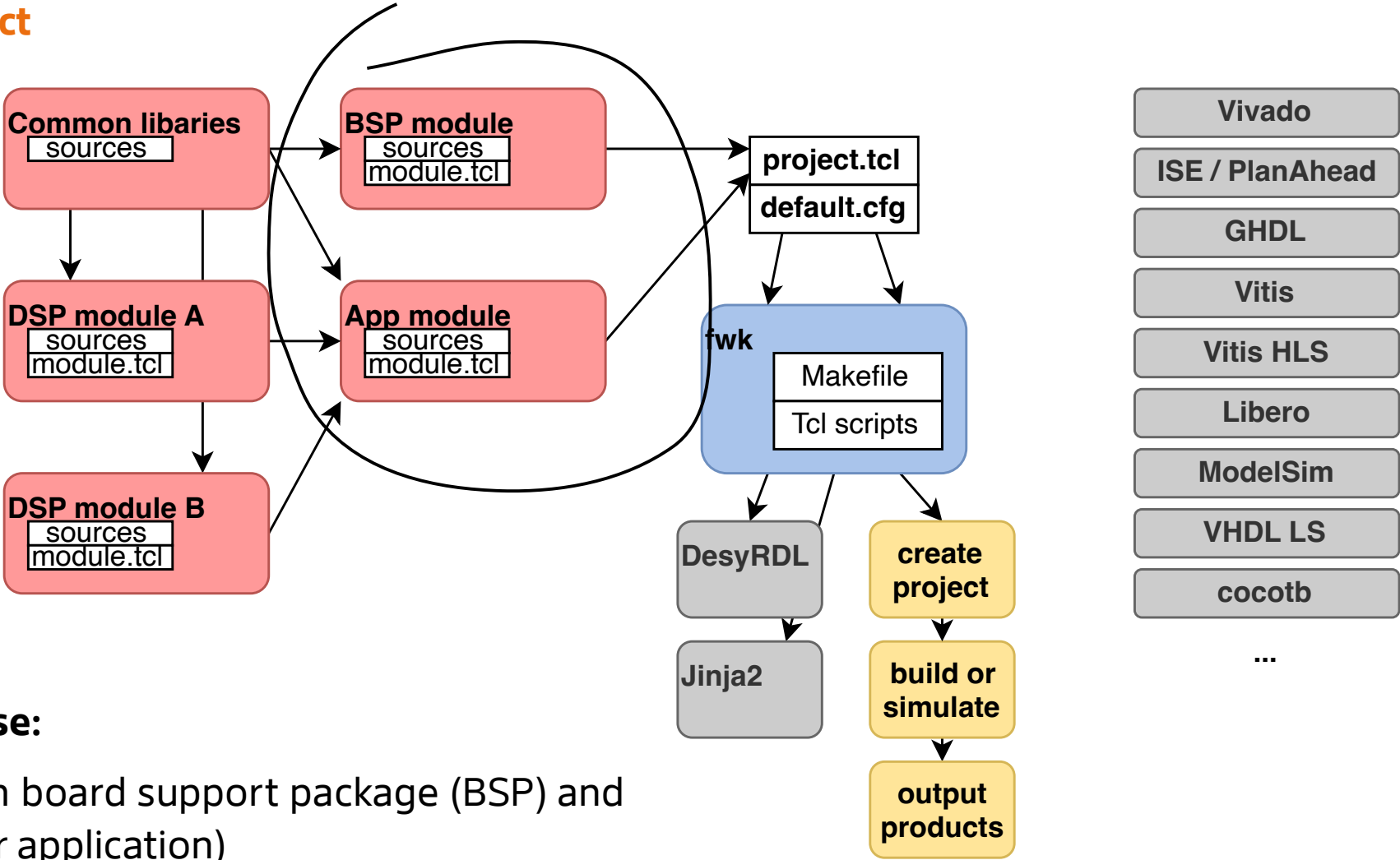
```
make cfg=fractional_divider_signed gui
```



[1] https://gitlab.desy.de/fpgafw/projects/verif/verif_desy_vhdl

Firmware Framework Structure

Overview of a project

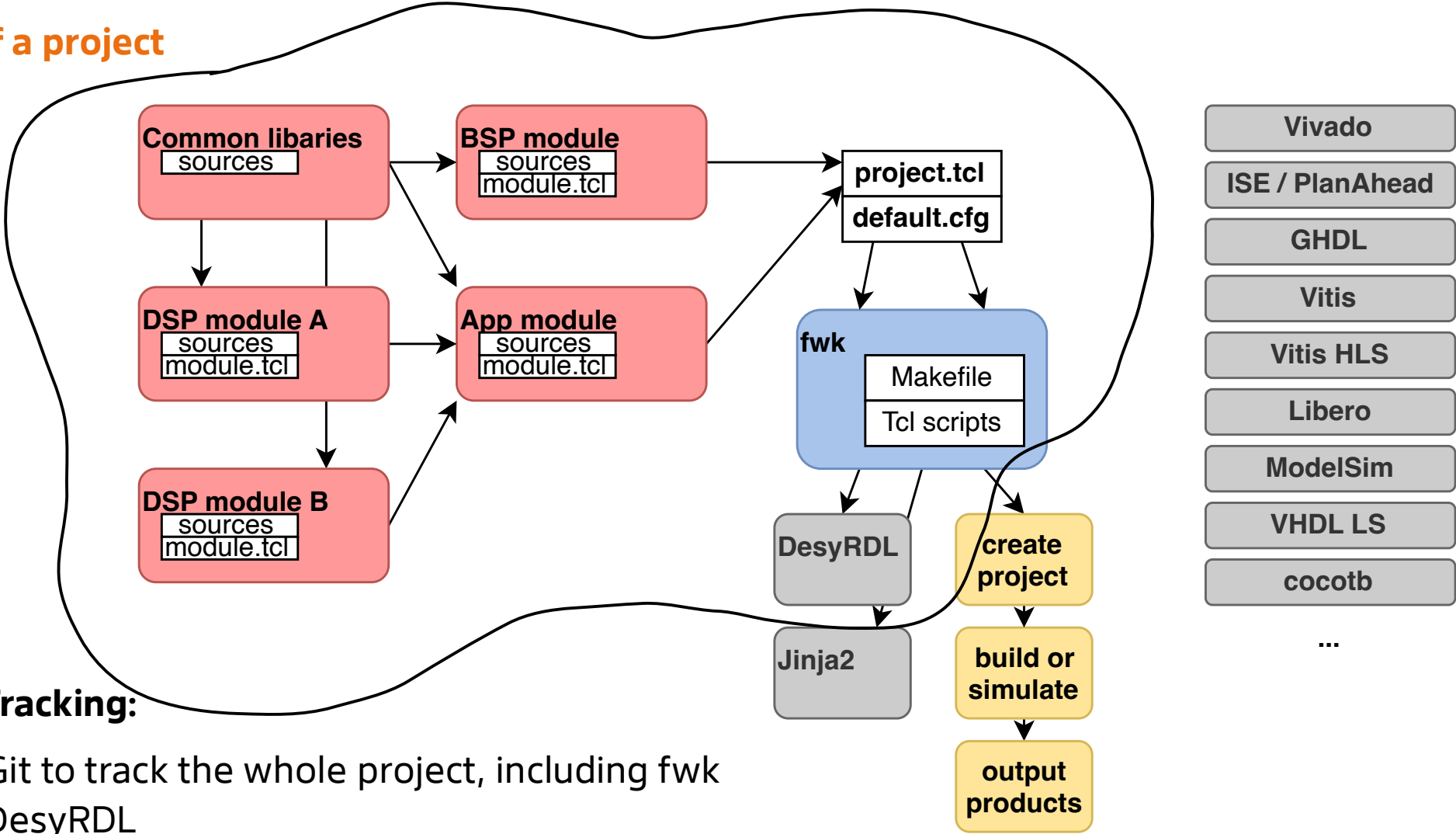


Modular code base:

- Split between board support package (BSP) and payload (your application)
- Can be managed as Git submodules

Firmware Framework Structure

Overview of a project



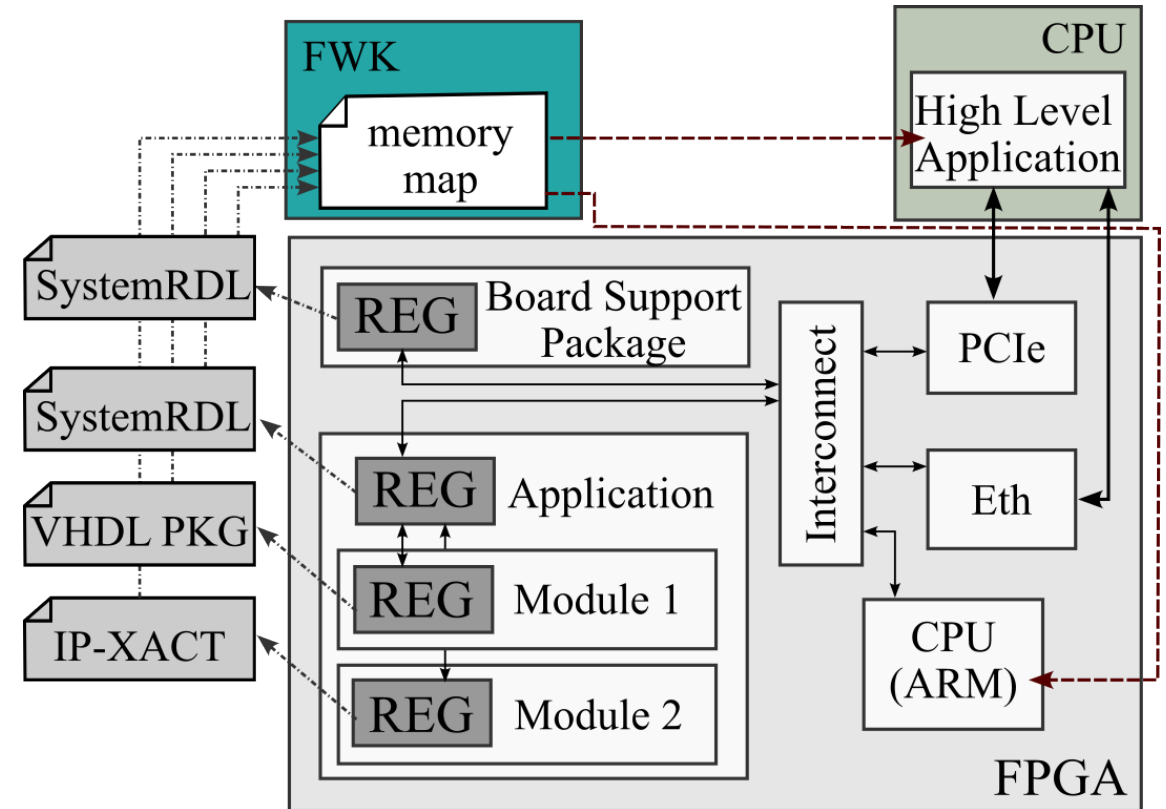
Version Tracking:

- Use Git to track the whole project, including fwk and DesyRDL
- The Git status can be baked into the firmware

Address Space Construction

A common problem in FPGA world

- FWK collects all the address space components
- Constructs an address space tree within Tcl
- Uses DesyRDL to generate address space artifacts: HDL implementation, documentation, software files
- Various formats supported: SystemRDL (preferred), IP-XACT, "IBUS" (proprietary)
- Generates address maps for each *access channel* (separate trees or multi master mode), e.g.:
 - PCIe
 - Embedded Linux (e.g. on an ARM CPU)
 - Embedded bare metal application
 - Ethernet



Address Map Construction

SystemRDL

Define register mapping in a standardized way

SystemRDL 2.0 is a register description language that can fully describe an address map (registers, memory regions)

Why SystemRDL?

- Open standard, released by accellera
- Open source compiler in Python by @amykyta3^{1,2}
 - Written in Python
 - Perfect fit for a custom generator
 - Well documented

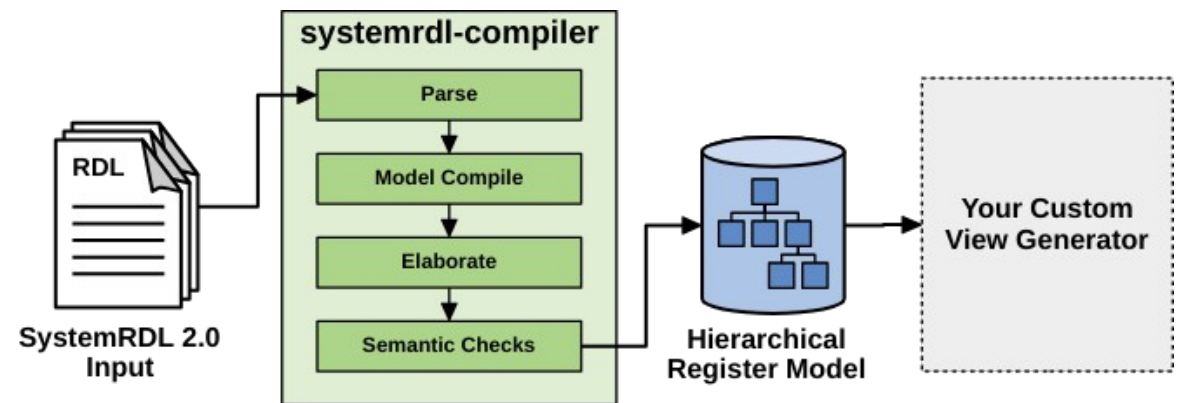


Example SystemRDL registers

```
20 reg {
21   desc = "Bytes to write from buffer.";
22   default sw = rw;
23   default hw = r;
24   field {} data[15:0] ;
25 } BYTES_TO_WRITE @0x200c ;
26
27 reg {
28   desc = "Bytes to read from buffer.";
29   default sw = rw;
30   default hw = r;
31   field {} data[15:0];
32 } BYTES_TO_READ @0x2010;
33
34 reg {
35   desc = "Control register for handshaking.";
36   default sw = rw;
37   default hw = r;
38   default swmod;
39   field {} data[7:0];
40 } CONTROL[4];
```

[1] <https://github.com/SystemRDL/systemrdl-compiler>
[2] <https://systemrdl-compiler.readthedocs.io/en/stable/>

SystemRDL and Accellera are trademarks of Accellera Systems Initiative Inc. <https://accelera.org/trademarks>



SystemRDL compiler flow
(image by @amykyta3)

DesyRDL

Tool for address space and register generation

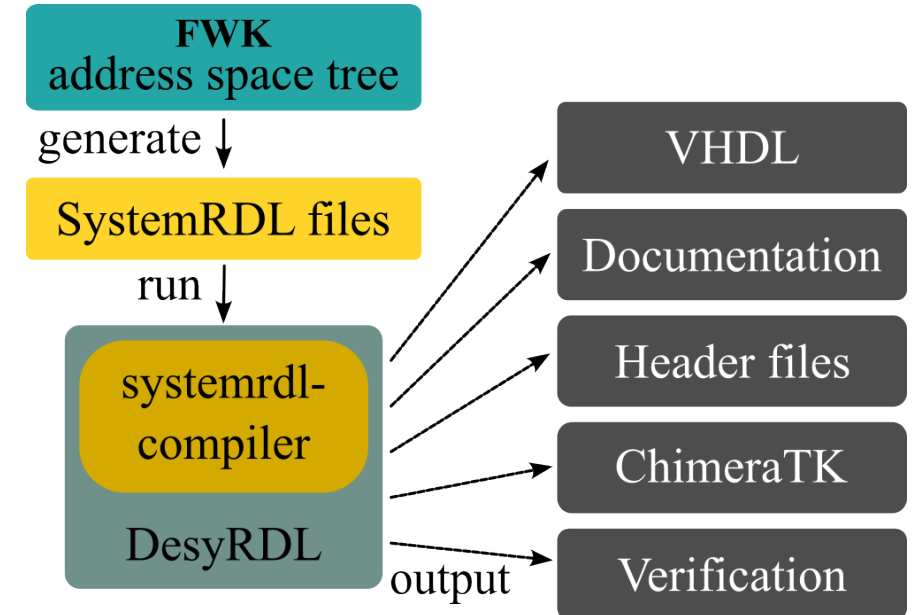
An address space generation backend, and a key components of the whole firmware framework.

- Written in Python
- Uses systemrdl-compiler
- Open Source

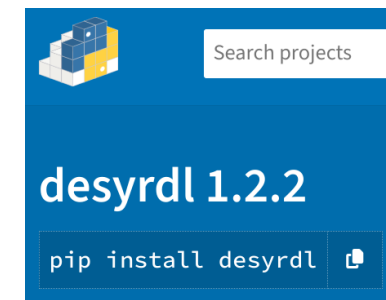
FWK collects and/or generates SystemRDL files.
DesyRDL generates address space artifacts:

- **VHDL** for each module/IP with configurable interfaces: AXI4 Lite, IBUS(proprietary), Wishbone (planned)
- **Documentation** in a form of markup files (AsciiDoc) with list of registers
- ChimeraTK “**mapfiles**”
- **Verification register models** (currently for cocotb only)

Generation of artifacts by DesyRDL tool.



Available on pypi.org



Integration example

One application's main.tcl file

Steps involved:

- Add further modules (dependencies)
- Set configuration constants
- Collect source files (VHDL, XDC)

```
proc init {} {
    variable Config
    variable ConfigFile

    addSrcModule example_desyrdl ${::fwfwk::SrcPath}/example_desyrdl/tcl/main.tcl
    addSrcModule timing          ${::fwfwk::SrcPath}/timing/tcl/main.tcl
    addSrcModule daq             ${::fwfwk::SrcPath}/daq/tcl/main.tcl

    set ConfigFile ../hdl/pkg_app_config.vhd
    parseVhdlConfigFile Config $ConfigFile

    # set modules Config
    set timing::Config(C_OUT_TRG)      2
    set timing::Config(C_EXT_TRG)     8
}

proc setSources {} {
    variable Sources
    variable Config
    variable ConfigFile

    lappend Sources [list $ConfigFile "VHDL" ""]
    lappend Sources {"../hdl/app_example_top.vhd" "VHDL"}
    lappend Sources {"../hdl/fmc1z7io_payload.vhd" "VHDL"}

    if { "xc7z030" == $Config(DEVICE) } {
        lappend Sources {"../cstr/app_fmc1z7io_30.xdc" "XDC" "" "" "LATE"}
    } else {
        lappend Sources {"../cstr/app_fmc1z7io_x5.xdc" "XDC" "" "" "LATE"}
    }
}
```

1,1

Top

This is a simplified version of what the DAMC-FMC1Z7IO example design provides.

See:

https://gitlab.desy.de/fpgafw/projects/example/damc_fmc1z7io_example_design/

Integration example

One application's main.tcl file

Steps involved:

- Add further modules (dependencies)
- Set configuration constants
- Collect source files (VHDL, XDC)

- Declare address space components
- Add source files to the Project

```
proc setAddressSpace {} {  
    variable AddressSpace  
    variable AddressSpaceDAQ  
  
    addAddressSpace AddressSpace "app_example" RDL {} ../rdl/app_example.rdl  
    addAddressSpace AddressSpace "EXMPLRDL" INST {} example_desyrdl::AddressSpace  
    addAddressSpace AddressSpace "TIMING" INST {} timing::AddressSpace  
    addAddressSpace AddressSpace "DAQ" INST {} daq::AddressSpace  
  
    addAddressSpace AddressSpaceDAQ "app_daq" RDL {} ../rdl/app_daq.rdl  
  
    addAddressSpace ::fwfwk::AddressSpace "APP" INST {C0 0x00800000 8M} AddressSpace  
    addAddressSpace ::fwfwk::AddressSpace "APP" INST {C13 0x40800000 8M} AddressSpace  
    addAddressSpace ::fwfwk::AddressSpace "DAQBUF" INST {C13 0x00000000 64M} AddressSpaceDAQ  
  
    addAddressSpace ::fwfwk::AddressSpace "APP" INST {C8 0x40800000 8M} AddressSpace  
}  
  
proc doOnCreate {} {  
    addSources "Sources"  
}  
  
proc doOnBuild {} {  
}  
  
proc setSim {} {  
}  
~  
33,1 Bot
```

This is a simplified version of what the DAMC-FMC1Z7I0 example design provides.

See:

https://gitlab.desy.de/fpgafw/projects/example/damc_fmc1z7io_example_design/

Documentation

Docs-as-code

Documentation is kept together with the source code in one repository

- Use markup language and the same development flow
- Update source code → update documentation

FWK collects all documentation components.

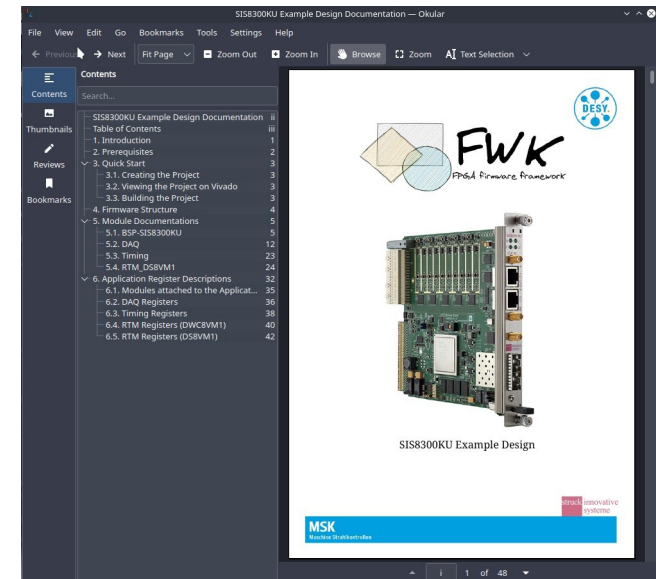
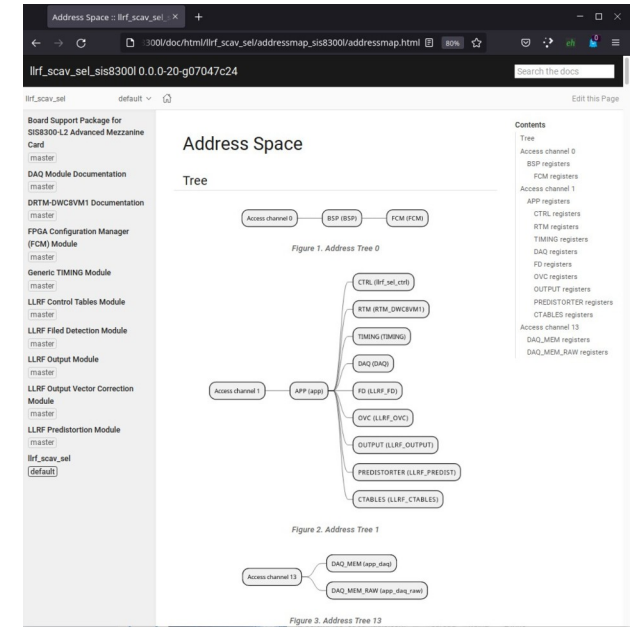
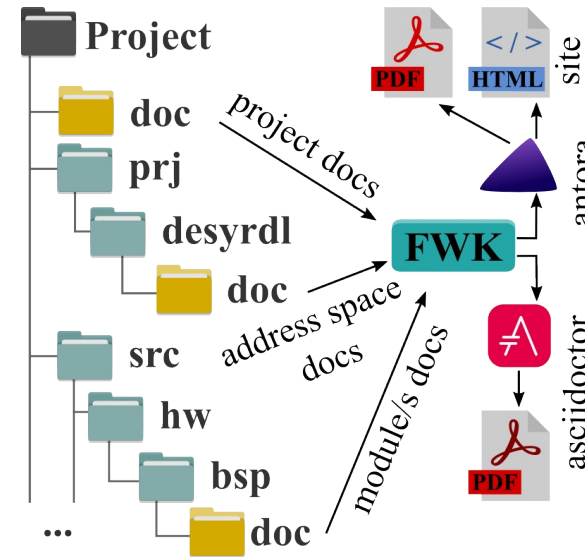
Automated export from repository to HTML, PDF, ...

AsciiDoc: main documentation language

A powerful text markup language,
more advanced than markdown
and less complex than LaTeX.

Tools:

- **Antora** as a framework for static websites (also PDF)
- **Asciidoctor** tools for HTML, DocBook, manual pages, PDF, EPUB



Towards Open Source

Initial version developed in 2013 for EuXFEL and MTCA.4 – presented on ICALEPCS 2015

Why open source:

- Increase and simplify collaboration with different facilities/universities/companies
- Get constructive feedback from the community and improve our solutions.

Challenges:

- Where and how to publish? -> **gitlab.desy.de**
- Which license to pick? -> FWK License: **Apache 2.0**
- How to release HDL code? -> code cleanup and restructure, license: **CERN-OHL-W-2.0**

Published on DESY GitLab in 2022, presented on ICALEPCS 2023

Repository: <https://gitlab.desy.de/fpgafw/fwk>

Documentation: <https://fpgafw.pages.desy.de/docs-pub/fwk/index.html>

(still as a partial mirror of internal repository and documentation site – plans to migrate all to gitlab.desy.de)

WEPGF074

Proceedings of ICALEPCS2015, Melbourne, Australia - Pre-Press Release 23-Oct-2015 11:00WEPGF074

FPGA FRAMEWORK FRAMEWORK FOR MTCA.4 AMC MODULES*
Lukasz Hutkowiak, Tomasz Kozak, Bin Yang, DESY, Hamburg, Germany
Pawel Pepek, DMCS, Lodz University of Technology, Lodz, Poland
Radoslaw Rybaniec, ISE, Warsaw University of Technology, Warsaw, Poland

Abstract
Many of the modules in specific hardware architectures use the same or similar communication interfaces and IO connections. MTCA.4 (MTCA) is one example of such a case. All basic communication with the control processing unit (CPU) over PCI Express (PCIe), and data to each other using Multi-Gigabit Transceivers (MGT), use the same hardware resources and have the same Zone 3 IO or FPGA resource and (FMC) connectors. All these interfaces are connected and implemented as Field Programmable Gate Array (FPGA) chips. It makes possible to separate the interface logic from the application logic. This structure allows to reuse already done firmware for one application and to create new application on the same module. Also, already developed code can be reused in new boards as a library. Proper structure allows the code to be reused and makes it easy to create new firmware.

This paper will present structure of firmware framework and activities taken to speed up firmware development for MTCA.4 architecture. European XTEL control system, firmware, which uses the described framework, will be presented as example.

INTRODUCTION
The MTCA.4 standard is derived from the Advanced Telecommunications Computing Architecture. It is enhanced for Real IO and Precision Timing and offers a compact environment for transmission and parallel processing of large amount of data. The modularity and connectivity is defined by the standard PRAGMATIC MTCA.4 specification [1,2]. Many modules are called Advanced Module Cards (AMC). Each of them can be paired with the Real Transition Module (RTM), handling communication and signal transfer is handled over the so-called Zone 3 region. The basic architecture follows the idea of a centralized powerful processing unit that is connected to various AMC IO boards over several PCIe lanes, dedicated trigger lines, clock lines and platform related management lines. The AMC hardware offers also a ports for low-latency links connection (optical differential pairs, GbE-backbones) that can reach up to 10 Gbit/s allowing the boards to communicate using serial transmission (e.g. utilizing the MGT).

The main function of the AMC modules is to provide communication interfaces and perform digital signal processing. All IO lines are connected to the FPGA chip on AMC board, shown in Figure 1. Depending on the

application and the used AMC, different RTM, FMC, or front IO modules can be connected. Thanks to flexibility of the FPGA, many different applications can be run on one AMC. All using the same hardware resources if a report of hardware is already provided in the code, it can be reused in many applications. The MTCA.4 firmware framework introduces an additional abstraction layer that separates the hardware dependent logic from user application logic. The framework specifies universal interfaces on this layer. This allows the same firmware and software components to be reused, irrespective of the type of the used hardware. It means that one application logic can be run on different AMC modules.

*Work supported by DESY XMA group.
lukasz.hutkowiak@desy.de

Hardware Technology ISBN 978-3-95446-146-9

MO4A003

978-3-95446-146-9 ISBN 978-3-95446-146-9
978-3-95446-216-7 ISBN 978-3-95446-216-7
0611-19-18429-0 ICALEPCS2023-04A003

THE DESY OPEN SOURCE FPGA FRAMEWORK

M. Büchler, N. Omsdagehl, Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract
Modern FPGA Firmware development involves integrating various intellectual properties (IP), modules written in hardware description language (HDL), high-level synthesis (HLS), and software/hardware (SW/HW) co-design. Lines or hardware applications. This process may involve multiple tools from the same or different vendors, making it complex and challenging. Additionally, scientific institutions such as DESY require long-term maintenance and reproducibility for designs that may involve multiple developers, further complicating the process. To address these challenges, we have developed an open source FPGA framework (FWK) at DESY that streamlines development, facilitates collaboration, and reduces complexity. The FWK achieves this by providing an abstraction layer, a defined structure, and guidelines to create high-quality designs with ease. FWK also generates documentation and address design necessary for high-level synthesis frameworks like Chisel/TK. This paper presents an overview and the idea of the FWK.

INTRODUCTION
At the scientific institute DESY [1], Field Programmable Gate Array (FPGA) have become indispensable components within a wide array of control and diagnostic systems. The adoption of FPGA in these applications is primarily driven by their exceptional multichannel computation power and unparalleled flexibility. However, the FPGA framework development process at DESY presents a set of significant challenges. The hardware's flexibility, including XAXL and FLASH, demands long-term support and maintenance, spanning often 20 years [2]. Over such extended periods, numerous features evolve or undergo modifications, sometimes are updated or replaced, and hardware requirements change or disappear. What further complicates the process is the involvement of multiple developers and collaborations, with projects spanning between responsible persons.

These hardware development challenges coincide with the management of multiple projects, often handled by small teams, and run in parallel with continuous stream of new developments and rapid prototyping efforts. To further complicate matters, the most recent developments encompass a convergence of multiple technologies, entailing the integration of code written in Hardware Description Language (HDL), High-Level Synthesis (HLS), Embedded C/C++ and Embedded Linux into a unified and coherent design. To address these challenges, the development of FPGA framework development based on, a dedicated firmware framework has been

developed. This framework is an abstraction layer that captures a set of rules, scripts, functions, and procedures that are universal and reusable. The initial framework idea is presented in Figure 1. This framework has been designed to tackle a wide range of issues:

- Universal support for multiple vendor tools through an abstraction layer.
- Consistent framework construction for a unified approach across projects.
- Responsible builds for enhanced stability and reliable code.
- Code modularity for accelerated development and reduced redundancy.
- Collaboration support for multiple developers working on the same project.
- Faster creation of new hardware developments.
- Streamlined application deployment on ready hardware platforms.
- Efficient configuration management for easy customization and flexibility.
- Streamlined address space management and integration with Chisel/TK framework [3].
- Automatically generating the technical and user documentation.
- Integrated design verification capabilities.
- Quality assurance enhancement through automation in reviewing and builds, ready for continuous integration.
- Ability to combine multiple technologies such as HDL, HLS, Embedded C/C++ and Linux into one design.

The initial iteration of the firmware framework was created in 2013 for MTCA.4 system at DESY [4]. Initially, the framework was closely integrated with the code in a monorepo repository, leading to challenges related to sharing and scalability as project numbers grew. To address these issues, the framework was restructured to accept the framework from the code, restructure it, and release it as an open source. This iteration was motivated by the aim of supporting other institutions and facilitating collaboration through open sourcing [5].

Hardware FPGA & DAI Hardware

Pre-Press Release 23-Oct-2015 11:00

Results and Conclusions

Successfully used in MSK group at DESY over 10 years:
(also in various other groups and institutes)

- Team of 3 to 7 members
- ~40 projects
- 15+ distinct hardware boards
- Contributions of ~50 developers to the code base

FWK showed significant enhancements in:

- Collaboration
- Code quality
- Reproducibility
- Change traceability

FWK significantly reduces development time and improves maintainability, but it requires an initial learning curve and ongoing maintenance commitment, which should be carefully considered when implementing the FWK.

Thank you .

Contact

Deutsches Elektronen-
Synchrotron DESY

www.desy.de

Michael Büchler
MSK group
michael.buechler@desy.de