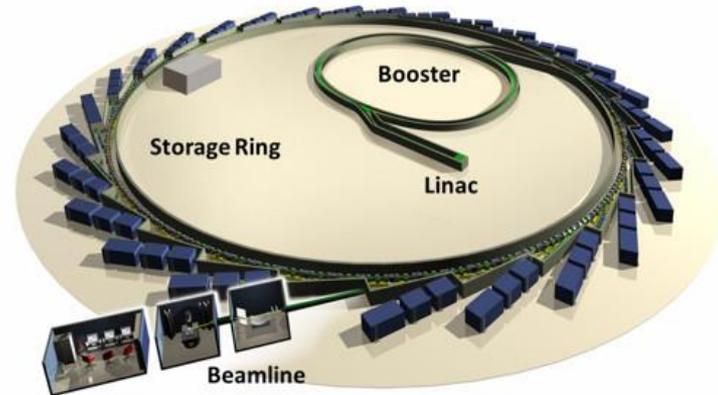


The PandABlocks framework for flexible run-time configuration of Zynq SoCs

Glenn Christian,
Diamond Light Source,
FDF, 20 May 2025

Diamond Light Source

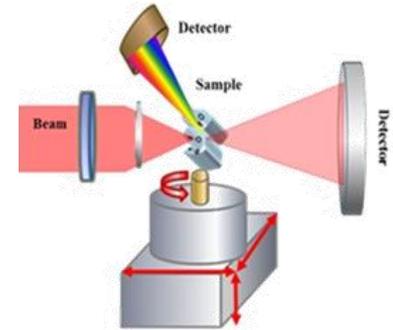
- UK national (third generation) synchrotron light source in Oxfordshire
- Started operation in 2007
- Currently 33 X-ray beamlines serving myriad of science research areas
- 3-GeV 562-m circumference SR with 100 MeV linac and booster synchrotron
- Diamond-II upgrade approved and underway – c. £500M upgrade to 4th gen light source starting end of 2027 (18 months)



PandA Motivation

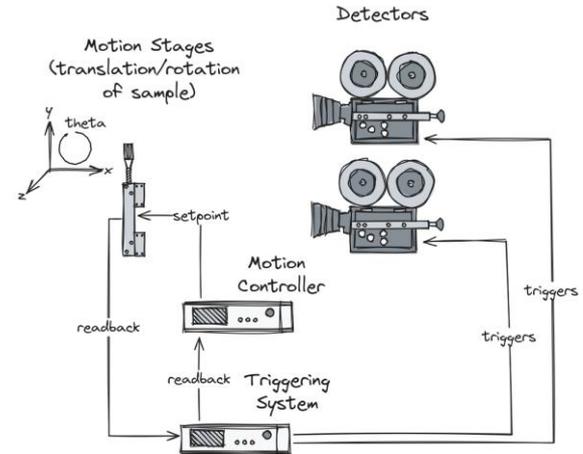
➤ Hardware triggered scanning

- Move samples and trigger detectors continuously during scan
- Decreases detector deadtime, and speeds up scans (fast dynamics)
- Requires precise synchronisation between motion control and detectors



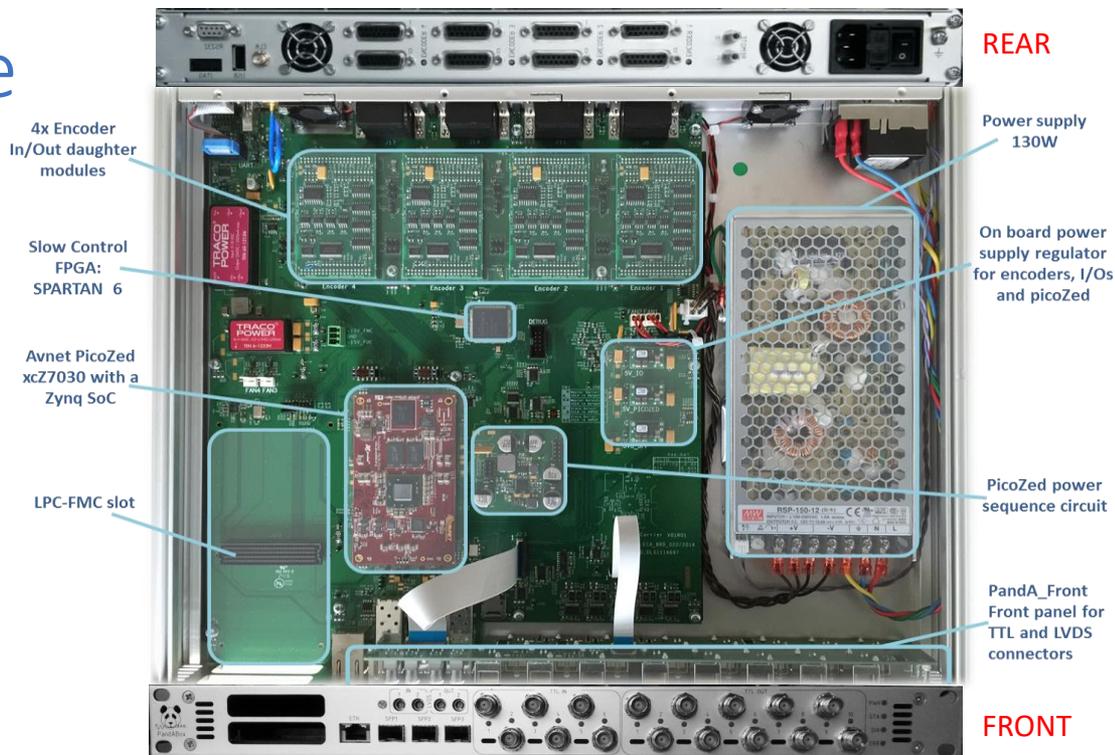
➤ 'PandABox' hardware development

- **PandA** = **P**osition and **A**cquisition
- Flexible FPGA based encoder processing platform
 - Triggers aligned on time or position
- Collaboration between **DLS** and **Soleil** (Started 2015)
- Now includes **MAX-IV**, **ALBA**, and **DESY** – **PandABox-2**



PandABox hardware

- Housed in 1U 19" rack
- Avnet PicoZed SoM – Xilinx Zynq 7030 SoC
 - Kintex-fabric FPGA and ARM Cortex A9 processor
- Four removable encoder modules:
 - Quadrature, SSI, and BiSS protocols
 - RS-485 line driver/receivers
 - Site customisable daughter cards
- Front-panel TTL and LVDS I/O for detector triggering
- Xilinx Spartan-6 (“slow”) FPGA for IO expansion
 - SPI link to Zynq-PL
- 3 front-panel SFPs (fast comms)
- LPC-FMC (ADCs/DACs, additional I/O) etc



Hardware design available on OWHR:

<https://www.ohwr.org/project/pandabox/wikis/home>

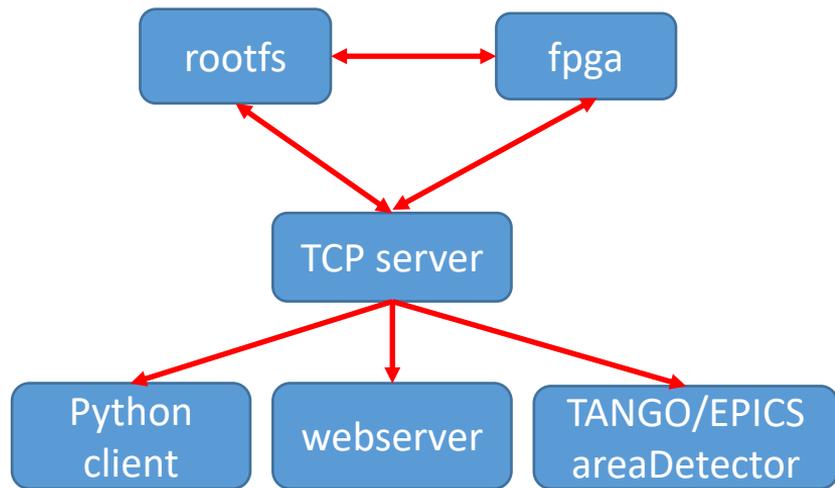


OPEN HARDWARE REPOSITORY

Commercially available from Quantum Detectors:



PandABlocks framework



- FPGA firmware (more in coming slides!)
- Rootfs:
 - FSBL, uboot, linux kernel image, root filesystem for the Zynq PS (Arm core)
- TCP server and kernel driver:
 - interface to Configuration and Status Registers in firmware
 - streaming captured data via DMA

Available from public repo: <https://github.com/PandABlocks>

The screenshot shows the GitHub profile for PandABlocks. The repository name is 'PandABlocks' with a description: 'Zynq based framework with run-time rewireable functional blocks, TCP and webserver'. It has 9 followers and a link to the GitHub repository. The 'Popular repositories' section lists several sub-repositories:

- PandABlocks-FPGA**: VHDL functional blocks with their simulations and test sequences. 20 stars, 18 forks.
- PandABlocks-rootfs**: Root filesystem build with tools for building zyg packages. 4 stars, 5 forks.
- PandABlocks-client**: Python client library for talking to PandABlocks-server. 4 stars, 11 forks.
- PandABlocks-loc**: Create an ICC from a PandA. 3 stars, 6 forks.
- PandABlocks-server**: TCP server exposing an ASCII interface to functional blocks. 2 stars, 3 forks.
- PandABlocks-webcontrol**: Webserver and web GUI based on Malcom. 1 star, 2 forks.

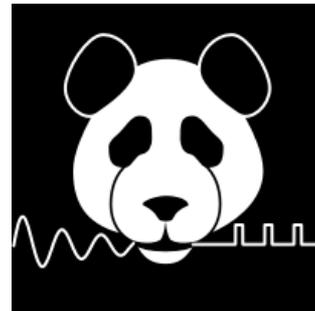
The 'Repositories' section shows a list of repositories with search filters and a 'New' button. The first repository listed is 'PandABlocks-FPGA' with 20 stars and 18 forks, updated 10 hours ago. Other repositories include 'PandABlocks-server', 'PandABlocks-Yocto', and 'PandABlocks-rootfs'.

PandABox



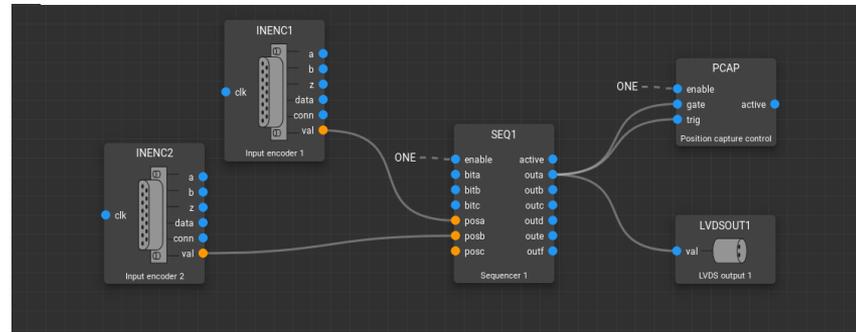
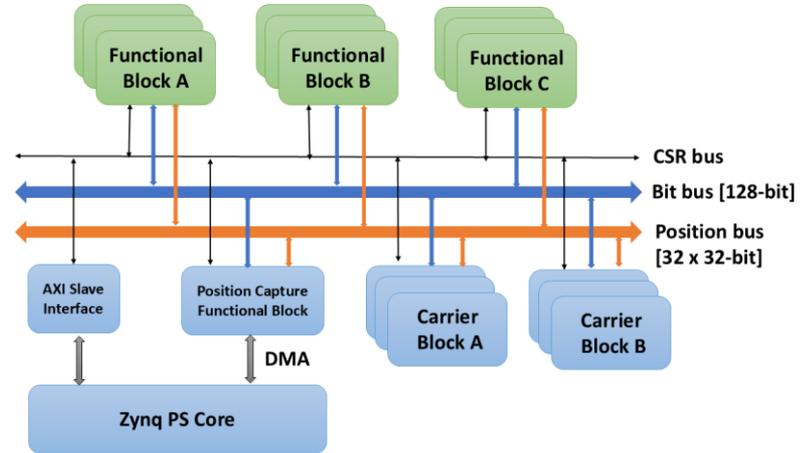
```
88 txcharisk_o <= (others => '0');
89 end if;
90
91 case TX_STATE is
92   when 1 => txdata_o <= BITOUT_1(0) & POSOUT1_1(31 downto 16) & c_k28_5;
93   when 2 => txdata_o <= BITOUT_1(1) & POSOUT1_1(15 downto 0) & POSOUT2_1(31 downto 24);
94   when 3 => txdata_o <= BITOUT_1(2) & POSOUT2_1(23 downto 0);
95   when 4 => txdata_o <= BITOUT_1(3) & POSOUT3_1(31 downto 8);
96   when 5 => txdata_o <= BITOUT_1(4) & POSOUT3_1(7 downto 0) & POSOUT4_1(31 downto 16);
97   when 6 => txdata_o <= BITOUT_1(5) & POSOUT4_1(15 downto 0) & check_byte;
98 end case;
99
100 if TX_STATE = 6 then
101   TX_STATE := 1;
102 else
103   TX_STATE := TX_STATE + 1;
104 end if;
105 end if;
106 end process;
107
108 ksyncce
109 port ma
110   c!l
111   bi!
112   bi!
113 );
114
115 -- Latc
116 latch_j
117 -- this variable will synthesise as a shift register
118 variable ksync_sr : std_logic_vector(5 downto 0);
119 begin
120   if rising_edge(sysclk_i) then
121     -- Detect change of txcharisk and shift into SR
122     ksync_del <= ksync;
123     ksync_sr := ksync_sr(4 downto 0) & (ksync xor ksync_del);
124     if ksync_sr(1) = '1' then
125       POSOUT1_1 <= POSOUT1_i;
126     end if;
127     if ksync_sr(2) = '1' then
128       POSOUT2_1 <= POSOUT2_i;
129     end if;
130     if ksync_sr(4) = '1' then
131       POSOUT3_1 <= POSOUT3_i;
132     end if;
133     if ksync_sr(5) = '1' then
134       POSOUT4_1 <= POSOUT4_i;
135     end if;
136
137     for i in 0 to 5 loop
138       if ksync_sr(i) = '1' then
139         BITOUT_1(i) <= BITOUT_1;
140       end if;
141     end loop;
142   end if;
143 end process;
144
145 end rtl;
```

PandABlocks



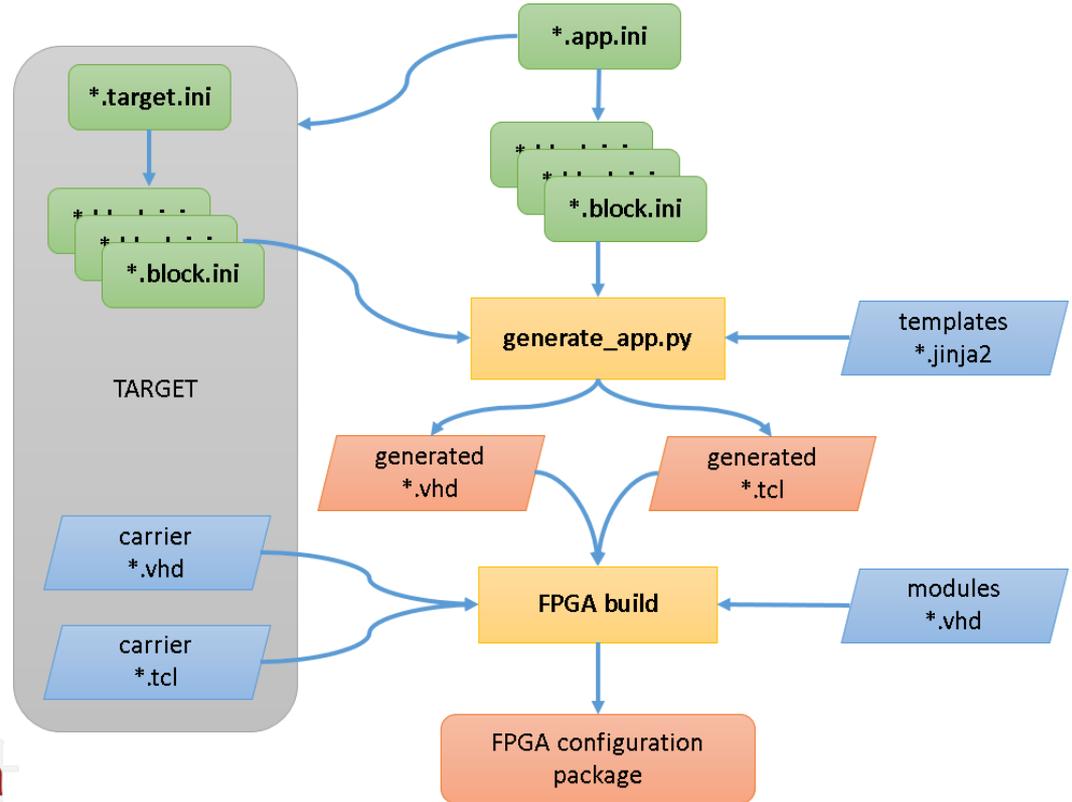
PandABlocks Architecture

- Specific functions and hardware resources represented by functional blocks
 - Hardware specific base blocks, eg TTL, LVDS
 - 'Soft' function blocks, eg LUT, SRgate, pulsegen etc
- Functional blocks **configurable** and **re-wirable** at run-time via multiplexed control and data buses, and Configuration/Status register (CSR) interface
 - 32 x 32-bit data bus (Position bus)
 - 128-bit control bus (Bit Bus)
 - CSR interface uses 128 KB, addressing up to 32 design blocks
- Lots of MUXs! Modest clock speed **125 MHz**



FPGA auto-generation framework

- Goals
 - Simplify inclusion of user-generated functional blocks
 - Implement arbitrary combinations of 'soft' and hardware blocks
 - Simplify targeting other Zynq-based hardware platforms
- Implementation
 - Set of text-based "ini" files
 - "app.ini" – set of 'soft' function blocks in design
 - "target.ini" – set of 'carrier blocks' available on the hardware
 - "block.ini" – defines register interface for each block + other information, eg IP, constraints required
- HDL wrappers and register interfaces for each block generated from templates using Python and jinja2 template engine
- Top-level Makefile drives the build flow



Register Interface

- 128 KB memory-mapped registers:
 - Supports up to 16 instances of 32 design blocks; 64 x 32-bit regs per block
 - AXI-lite interface to PS
 - Translated to simple STRB/ACK interface at the block level

```
-- Memory Bus Interface
-- Read
read_strobe_i   : in  std_logic;
read_address_i  : in  std_logic_vector(PAGE_AW-1 downto 0);
read_data_o     : out std_logic_vector(31 downto 0);
read_ack_o      : out std_logic;
-- Write
write_strobe_i  : in  std_logic;
write_address_i : in  std_logic_vector(PAGE_AW-1 downto 0);
write_data_i    : in  std_logic_vector(31 downto 0);
write_ack_o     : out std_logic;
```

Example: adding new block (new framework)

clock.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clock is
port (
  -- Clock and Reset
  clk_i      : in  std_logic;
  ENABLE_i   : in  std_logic;
  -- Block Input and Outputs
  out_o      : out std_logic;
  -- Block Parameters
  PERIOD     : in  std_logic_vector(31 downto 0);
  PERIOD_wstb : in  std_logic
);
end clock;

architecture rtl of clock is

signal reset      : std_logic;
signal counter32  : unsigned(31 downto 0);
signal DIV        : unsigned(31 downto 0);
signal half_period : unsigned(31 downto 0);

begin

reset <= PERIOD_wstb;
DIV <= unsigned(PERIOD) + 1;
half_period <= unsigned('0' & DIV(31 downto 1));

process(clk_i)
begin
  if rising_edge(clk_i) then
    -- If not enabled, or no period set stop the clocks
    if (ENABLE_i = '0') or (unsigned(PERIOD) = 0) then
      OUT_o <= '0';
      counter32 <= (others => '0');
      -- Reset counter on parameter change.
    elsif (reset = '1') then
      OUT_o <= '1';
      counter32 <= unsigned(PERIOD) - 1;
    else
      -- Reload when reach Zero and assert clock output.
      if (counter32 = 0) then
        OUT_o <= '1';
        -- Half period reached
      elsif (counter32 = half_period) then
        OUT_o <= '0';
      end if;

      -- Free running down counter.
      if (counter32 = 0) then
        counter32 <= unsigned(PERIOD) - 1;
      else
        counter32 <= counter32 - 1;
      end if;
    end if;
  end if;
end process;
end rtl;
```

[.]

description: Configurable clocks

entity: clock

[ENABLE]

type: bit_mux

description: Halt and reset on falling edge, enable on rising

[PERIOD]

type: param time

description: Period of clock output

wstb: True

[OUT]

type: bit_out

description: Clock output

[[[NUM]]]

number:



-- Generate NUM instances of the blocks

GEN : FOR I IN 0 TO (NUM-1) GENERATE



-- Connect to the actual logic entity

clock : entity work.clock

[SEQ]

number:

ENABLE_i => ENABLE(I)(0),

PERIOD => PERIOD(I),

[SRGATE]

number:

PERIOD_wstb => PERIOD_wstb(I),

OUT_o => OUT_o(I),

clk_i => clk_i

);

[SFP3_SY

module:

sfp_site

END GENERATE;

end rtl;

clock_wrapper.vhd

```
logic_vector(NUM-1 downto 0);
_2_array(NUM-1 downto 0);
logic_vector(NUM-1 downto 0);
ral range 0 to (2**read_address_i'length - 1);
ral range 0 to (2**write_address_i'length - 1);
logic_vector(NUM-1 downto 0);
```

if the blocks
) GENERATE

al logic entity
work.{{ entity }}

```
is("bit_mux" %)
+ "i" }} => {{ field.name }}(I)(0),
&lds("pos_mux" %)
+ "i" }} => {{ field.name }}(I),
&lds("time" %)
-> {{ field.name }}(I)(47 downto 0),
&lds(".*out" %)
+ "_o" }} => {{ field.name }}_o(I),
```

ld.numbered_registers(I) %
ame) }} => {{ register.name }}(I),

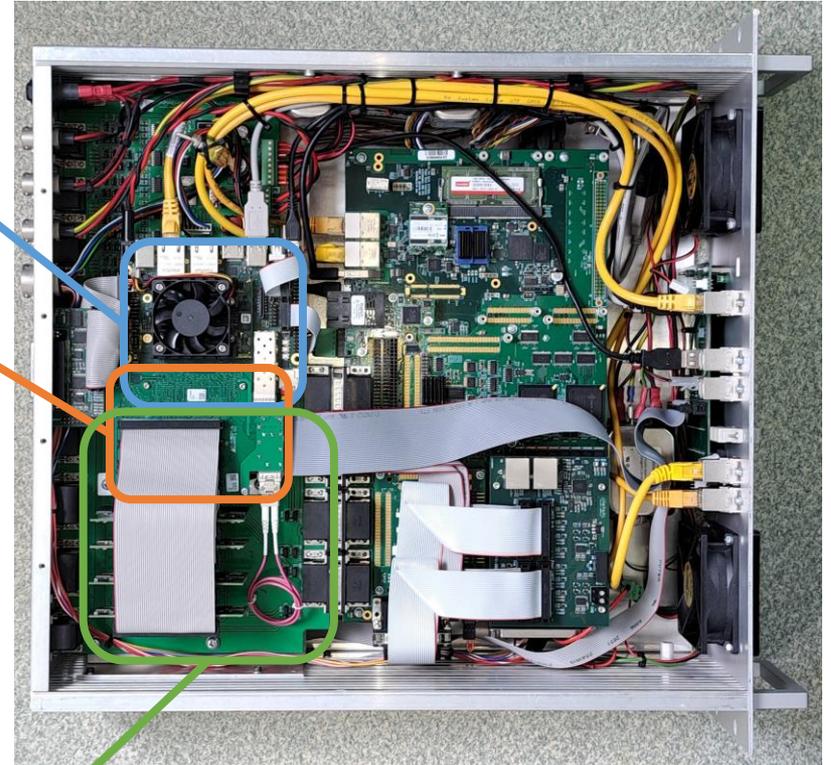
```
ime" %)
+ "_wstb" }} => {{ field.name }}_wstb(I),
```

```
{% for register in field.numbered_registers(I) %}
{{ pad(register.name + "_wstb") }} => {{ register.name }}_wstb(I),
{% endfor %}
{% endif %}
{% endfor %}
{% if type == "dma" %}
  dma_req_o
  => dma_req_o(I),
  dma_ack_i
  => dma_ack_i(I),
  dma_done_i
  => dma_done_i,
  dma_addr_o
  => dma_addr_o(I),
  dma_len_o
  => dma_len_o(I),
  dma_data_i
  => dma_data_i,
  dma_valid_i
  => dma_valid_i(I),
{% endif %}
clk_i
=> clk_i
);
END GENERATE;
```

block_wrapper.vhd.jinja2

FPGA Power Brick

- Enclustra XU6/ST1 integrated into PowerBrick Motion Controller (developed by Faraday Motion Controls and DLS)



ST1

Custom FMC

Line Driver Cards

Targeting new hardware platforms - ZCU102

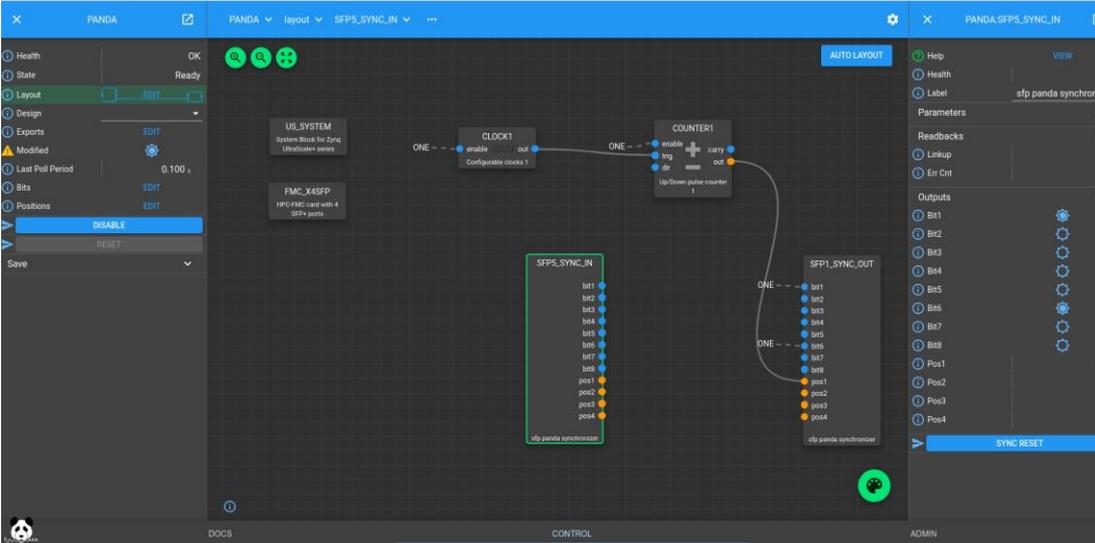
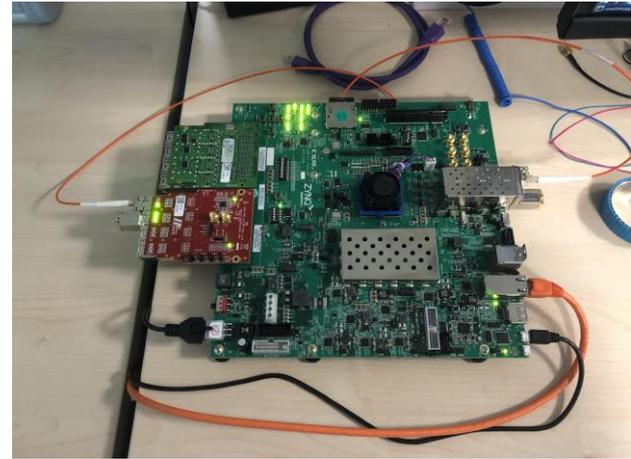
```
[.]
description: HPC-FMC card with 4 SFP+ ports
entity: fmc_x4sfp
type: io
interfaces: FMC
constraints: const/fmc_x4sfp.xdc const/fmc_x4sfp_impl.xdc
extra_interface: fmc_mgt: mgt 4

[.]
description:
  Standard set of PandABox blocks with:
  - no FMC modules
  - Panda synchroniser on all SFPs
target: zcu102
options: !pcap_std_dev
includes: common_soft_blocks.include.ini

[SFP1_SYNC]
module: sfp_panda_sync
ini: sfp_panda_sync_us.block.ini
site: sfp 1

[SFP4_SYNC]
module: sfp_panda_sync
```

target.ini



This replaces the panda_carrier block can be found in

02.target.ini

Synchronisation

- SFP-based blocks for synchronisation and data sharing
- **PandA-EVR** (event receiver): port of MRF timing protocol used at DLS
 - Event based system 2-byte (1 DBUS, 1 event code) @ 125 MHz – 2.5Gbps
 - Can use recovered clock as system clock



Synchronisation

➤ SFP-based blocks for synchronisation and data sharing

➤ **PandA-EVR** (event receiver): port of MRF timing protocol used at DLS

- Event based system 2-byte (1 DBUS, 1 event code) @ 125 MHz – 2.5Gbps
- Can use recovered clock as system clock

➤ **PandASync** – lightweight protocol for data sharing

- Share 8 bitbus and 4 posbus signals between PandAs; 6 cycle repeating pattern
- 4-byte (bitbus + some combination of posbus) @ 125MHz – 5 Gbps
- Requirement for < 1 clock cycles jitter between systems
 - PICXO – Phase Interpolator Controlled Xstal Oscillator (Xilinx IP)
 - Uses TXPI PPM controller
 - ~20 ps timing jitter between TTL outputs

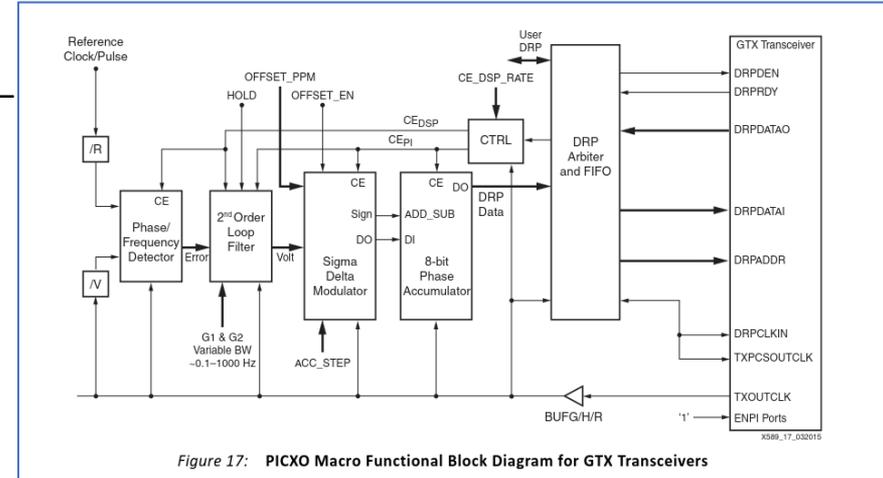


Figure 17: PICXO Macro Functional Block Diagram for GTX Transceivers

Synchronisation

- SFP-based blocks for synchronisation and data sharing
- **PandA-EVR** (event receiver): port of MRF timing protocol used at DLS
 - Event based system 2-byte (1 DBUS, 1 event code) @ 125 MHz – 2.5Gbps
 - Can use recovered clock as system clock
- **PandASync** – lightweight protocol for data sharing
 - Share 8 bitbus and 4 posbus signals between PandAs; 6 cycle repeating pattern
 - 4-byte (bitbus + some combination of posbus) @ 125MHz – 5 Gbps
 - Requirement for < 1 clock cycles jitter between systems
 - PICXO – Phase Interpolator Controlled Xstal Oscillator (Xilinx IP)
 - Uses TXPI PPM controller
 - ~20 ps timing jitter between TTL outputs
- In development – ultra-fine delay (~3 ps) from MGT outputs
 - Plan to use for fast injection scheme on Diamond-II accelerators
 - MGT -> electrical signals, custom FMC design

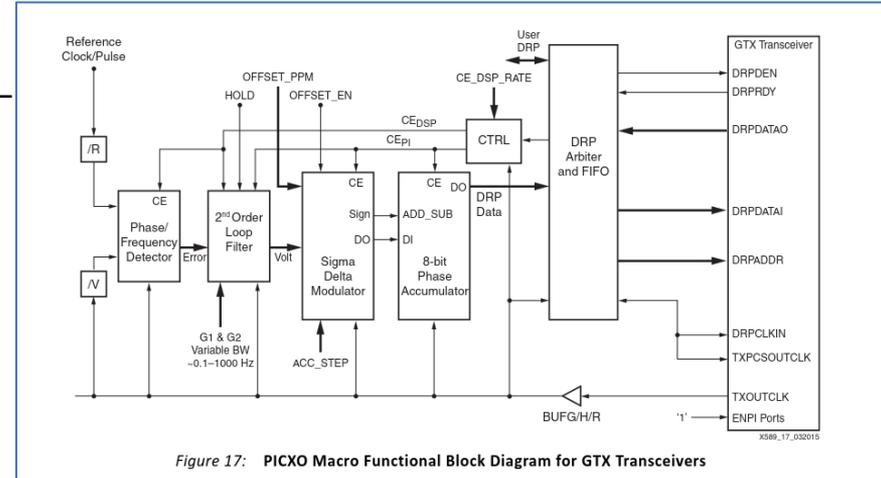
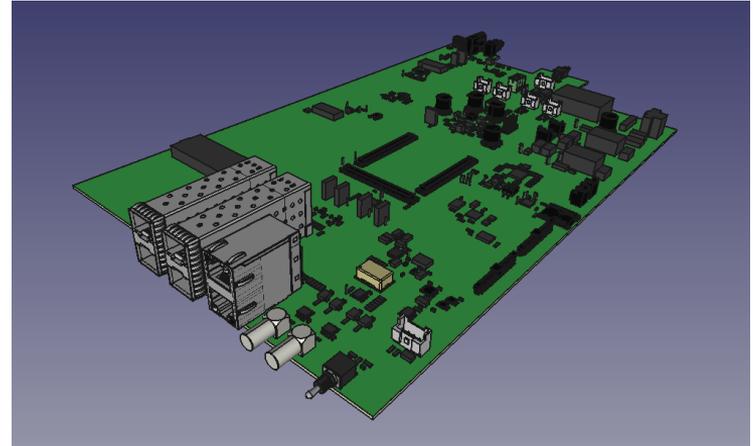
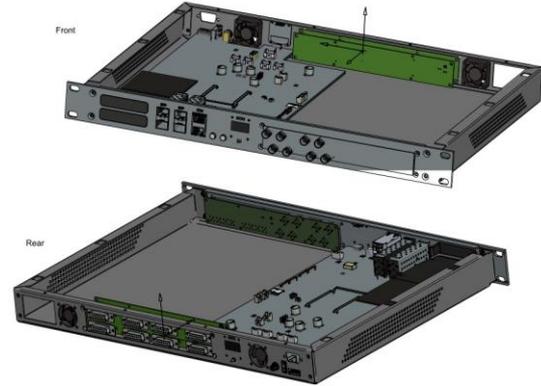


Figure 17: PICXO Macro Functional Block Diagram for GTX Transceivers

PandABox-2 (UltraPandA)

- 1U 19-inch form factor (half width variant)
- Interchangeable front (Digital I/O) and rear (encoder) panels
 - Symmetrical 50-way ribbon cable
 - Opportunity for customisable I/O panels
- LED screens on both front and rear
- Four native SFP+ sockets on front
- LPC-FMC+ socket with 12 MGTs pinned out
- Clocking Support for White Rabbit



Soliel (main board), **ALBA** (encoder + DI/O boards), **DESY** (Mechanics), **DLS** (software/firmware), **MAX-IV**

Development environment

➤ Development container for all PandABox related builds

- Rocky8 image with standard packages (gcc etc)
- ARM cross-compiler; AMD kernel, device-tree, uboot sources
- Source code for all rootfs packages Zynq-PS
- Static python environment!
- NOT AMD tools eg (Vivado, Vitis etc)

➤ Continuous Integration

- Github Actions
- CI builds for FPGA, rootfs and server repos
- Includes python tests (framework) and HDL tests (functional block sims)
- CI builds of the firmware images
 - Github runner uses Vivado on local-hosted cluster
 - De-facto source for releases

The screenshot shows a GitHub Actions workflow for the repository 'PandABlocks / PandABlocks-FPGA'. The workflow is named 'xu5-st1: adding app for panda-sync on ultrascale #1126' and is triggered by a push to the 'fmc-12c' branch. The workflow status is 'Success' with a total duration of 2h 52m 13s and 1 artifact.

The workflow is defined in a 'code.yml' file and consists of several jobs:

- Make: make boot**: Two parallel jobs for 'zynq, PandABox-no-fmc' and 'zynqmp, xu5_st1-no-fmc'.
- Make: make zpkg**: Two parallel jobs for 'PandABox-no-fmc' and 'PandABox-fmc_24vio'.
- Make: make zpkg**: Two parallel jobs for 'PandABox-fmc_acq427' and 'PandABox-fmc_acq430'.
- Make: make zpkg**: Two parallel jobs for 'PandABox-fmc_iback-stp-...' and 'ZedBoard-no-fmc'.
- Start self hosted aws runner**: A job to start the AWS runner.
- test**: A test job.
- setup rclone and mount on runner**: A job to setup rclone and mount on the runner.
- make zpkg on aws (xu5_st1-no-fmc)**: A job to make zpkg on AWS.
- make zpkg on aws (xu5_st1-fmc_acq4...)**: A job to make zpkg on AWS.
- Stop self hosted EC2 runner**: A job to stop the AWS runner.
- release**: A final release job.

The workflow is triggered via push 8 months ago by user 'glennchid' pushed to '2a3ea51 fmc-12c'.

Recent developments

- Yocto

- Current roots : Buildroot inspired system used at DLS for many years
 - Small and maintainable, but not so accessible for external collaborators
- Yocto more industry standard
- Remove burden of maintaining packages
- Hoping to avoid headaches with upgrading kernel versions, Vivado versions, new hardware etc
 - Issues may have addressed already in petalinux based distros
 - We may just be exposing new issues ...

- CocoTB

- Block level HDL tests previously used Vivado simulator (xsim)
 - Original testbenches in Verilog (mixed language support)
- Moved these to CocoTB (open-source python simulation framework)
 - Easier verification for python developers
 - Can now use GHDL or NVC– speed increase!
 - Coverage reports (with NVC)
- Not clear how to deal with simulations of hard blocks (Xilinx IP, primitives etc)



Summary

- Flexible framework to support multitude of beam line applications with a single firmware image (per hardware variant)
- Re-useable software/firmware stack, can be targeted to any Zynq-7000 or ZU+ platform
 - Easy way to bring up new systems, and get access to hardware features such as I2C
- Reusable hardware – new PandA more customisable I/O
 - Finding uses on Diamond-II storage ring for accelerator systems as well as beam lines
- Growing collaboration; welcome new contributors to the framework!!

Acknowledgements

PandA Collaboration

- DLS: Michael Abbott, Famous Alele, Glenn Christian, Tom Cobb, Chris Colborne, Oliver Copping, Andrew Cousins, Emilio Perez Juarez, Adedamola Sode, James Souter, Tom Trafford
- Soliel: Yves-Marie Abiven, Aicha Ammar, Jerome Bisou
- MAX-IV: Valerio Bassetti, Peter Sjöblom
- ALBA: Jose Avila Abellan, Víctor Maín Nadal, Xavier Serra Gallifa
- DESY: Linus Pithan, Iris Schwark, Oliver Seeck, Tobias Spitzbart

Thank you for your attention!

