ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUAPP05

PandABlocks - A FLEXIBLE FRAMEWORK FOR Zynq7000-BASED SoC CONFIGURATION

G. B. Christian*, M. Abbott, T. Cobb, C. Colborne, A. M. Cousins, P. Garrick, T. Trafford,
Diamond Light Source Ltd, Oxfordshire, UK
Y. M. Abiven, J. Bisou, F. Langlois, S. Minolli, G. Renaud, G. Thibaux, S. Zhang,
SOLEIL, Gif-sur-Yvette, France
I. S. Uzun, STFC, Oxfordshire, UK

Abstract

The PandABlocks framework comprises the FPGA logic, TCP server, webserver, boot sources and root filesystem, developed for the PandABox platform by Diamond Light Source and Synchrotron Soleil, for advanced beamline scanning applications. The PandABox platform uses a PicoZed System-on-Module, comprising a Zynq-7030 SoC, coupled to a carrier board containing removable position encoder modules, as well as various input and outputs. An FMC connector provides access to ADC/DACs or additional I/O, and gigabit transceivers on the Zynq allow communication with other systems via SFP modules. Specific functions and hardware resources are represented by functional blocks, which are run-time configurable and re-wireable courtesy of multiplexed data and control buses shared between all blocks. Recent changes to the PandABlocks framework are discussed which allow the auto-generation of the FPGA code and tcl automation scripts, using Python and the jinja2 templating engine, for any combination of functional blocks and SFP/FMC modules. The framework can target hardware platforms other than PandABox and could be deployed for other Zynq-based applications requiring on-the-fly reconfigurable logic.

INTRODUCTION

Many x-ray beamlines conduct experiments which involve moving a motor and synchronously acquiring data from a detector. When detector speeds are slow, it is sufficient to step scan the motor, allowing it to move and settle between each detector frame. For many modern beamlines, with detector frame rates in the hundreds or thousands of hertz range, continuous scanning is required, where the motor constantly moves through a trajectory while the detector is acquiring. This technique reduces detector dead-time, but requires precise synchronisation between motion control systems and detectors. To achieve this, the PandABox (Position and Acquisition Box) platform was developed by Diamond Light Source (DLS) and Synchrotron Soleil as a FPGA-based solution for motion encoder and detector trigger processing [1].

The PandA collaboration began in 2015 to develop a common platform to replace their previous generation of in-house systems for beamline synchronisation (Fig. 1): Zebra at DLS; and SPIETBOX at SOLEIL. Both systems are based around a Xilinx FPGA; a Spartan-6 in the case of Zebra, and Spartan-3 in the case SPIETBOX. The goals of PandABox were to develop a flexible system to address the increasingly demanding requirements for beamline scanning, and to overcome concerns with component obsolescence and technical limitations of Zebra and SPIETBOX. A secondary goal was to share resources between DLS and SOLEIL, with SOLEIL taking primary responsibility for the electronic and mechanical design and DLS developing the FPGA firmware, software and user interface. The hardware design for PandABox is freely available on the Open Hardware Repository (OWHR) [2], and is commercially available from Quantum Detectors [3].

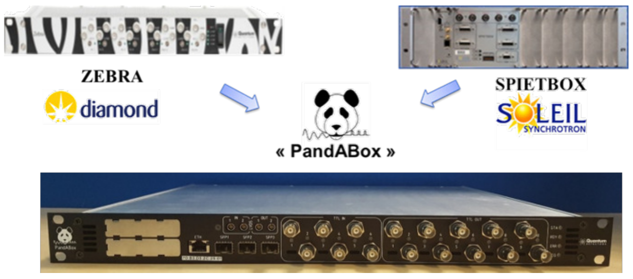

Figure 1: The PandABox collaboration between Diamond Light Source and SOLEIL.

THE PandABox HARDWARE

The PandABox hardware is shown in Fig. 2 [4, 5]. The complete assembly is designed to fit within a 19 inch, 1U rack. The system comprises a custom carrier board and encoder daughter cards, and an off-the-shelf PicoZed Z7030 System-on-module (SoM), produced by Avnet [6]. The PicoZed SoM contains a Xilinx Zynq-7030 System-on-Chip (SoC), as well as various on-board peripherals such as DDR3 and QSPI flash memory, ethernet and USB interface chips. The Zynq-7030 comprises Kintex-equivalent FPGA programmable logic (PL) fabric, alongside an embedded dual-core ARM Cortex A9 processor system (PS). Signals from the PS, PL, and peripherals, are brought out on micro-headers on the underside of the SoM PCB, where they connect to the carrier board. A Xilinx Spartan-6 FPGA is also fitted to the carrier to provide addition I/O and monitoring capabilities, due to a lack of sufficient pins on the Zynq SoC [1]. The Spartan-6, referred to as the 'Slow

* glenn.christian@diamond.ac.uk

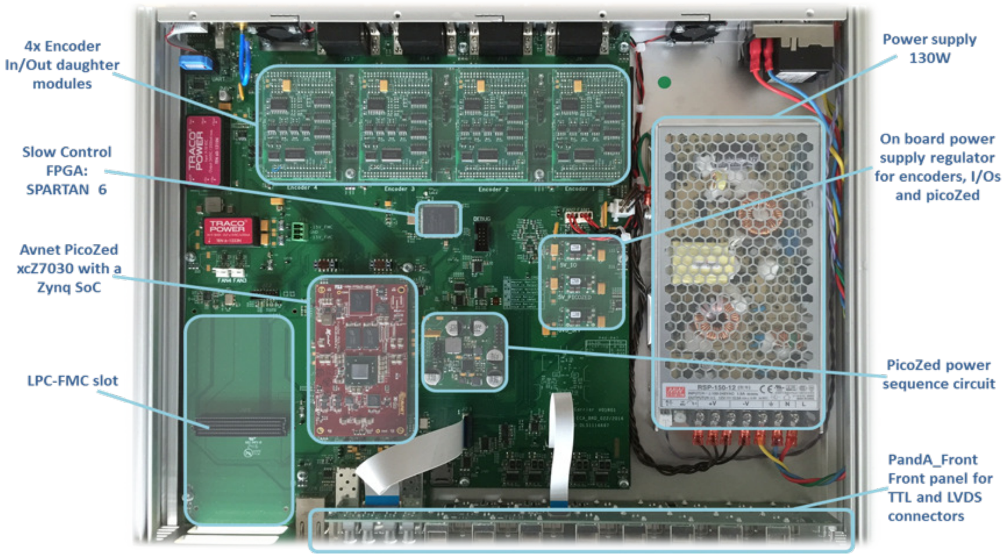
4x Encoder In/Out daughter modules
Slow Control FPGA: SPARTAN 6
Avnet PicoZed xcZ7030 with a Zynq SoC
LPC-FMC slot
Power supply 130W
On board power supply regulator for encoders, I/Os and picoZed
PicoZed power sequence circuit
PandA_Front Front panel for TTL and LVDS connectors

Figure 2: PandABox platform: internal view.

FPGA' due to its use with non-time critical signals, connects via an SPI link to the Zynq PL.

PandABox can interface with up to four motor encoders via D-type connectors on the rear panel. The encoder cards are designed to support a variety of incremental and absolute protocols, such as quadrature, SSI, and BiSS-C. The use of removable daughter cards allows for site specific variants of the cards with different external interfacing, and both SOLEIL and Diamond maintain their own versions of the encoder cards adapted to their individual requirements. The front panel provides six TTL inputs and ten TTL output on BNC connectors, and two LVDS inputs and two outputs on LEMO connectors, used, for example, for detector triggering.

The carrier board contains three SFP sockets and one low-pin count (LPC) FPGA mezzanine card (FMC) connector, to provide extended functionality and additional I/O such as ADCs and DACs. The SFP ports provide the capability for fast serial communication between PandABox units and other devices, such as an MRF event generator [7] for synchronisation with the accelerator timing system, making use of the multi-gigabit transceivers (MGTs) on the Zynq. The MGTs have been tested in loopback mode at up to 6.25 Gbps [4], and are used at DLS in the 'panda_sync' block to share data between PandABoxes at a rate of 5 Gbps.

The FMC socket provides an interface to a range of FMC devices. At DLS this is primarily used with an 8-channel custom 24V I/O card used for receiving GPIO triggers from motor controllers, and FMC analogue-to-digital and digital-to-analogue converters, such as the D-TACQ ACQ430 [8]. Space is reserved on the carrier board behind the FMC to allow for fitting an extended-length (ELF) FMC card, such as the D-TACQ ACQ427, for which a dedicated 15V power supply is provided.

The carrier board also provides a microSD card slot for non-volatile storage, an RS-232 serial port, gigabit ethernet and USB2 port; all connected to the Zynq PS. The FPGAs are connected to a JTAG chain, brought out on the rear panel, that can be used for configuration and debugging, however in routine operation the FPGAs are configured via the linux OS on the Zynq PS.

THE PandABlocks FRAMEWORK

The PandABlocks framework comprises the boot-loaders, Linux kernel image and root filesystem for the PS; FPGA firmware for the PL; and TCP server, client and web-GUI. The hierarchy of the software framework is shown in Fig. 3. The 'rootfs' build system combines the Zynq first-stage bootloader (FSBL) with the built uboot sources to form the second stage bootloader, as well as building the linux kernel from sources and assembling the root filesystem. The configuration of the PS block design is made within the IP Integrator tool of Vivado as part of the FPGA build, and is then exported to the build system for the *rootfs*. As such, there are interdependencies between the rootfs build and the FPGA build.

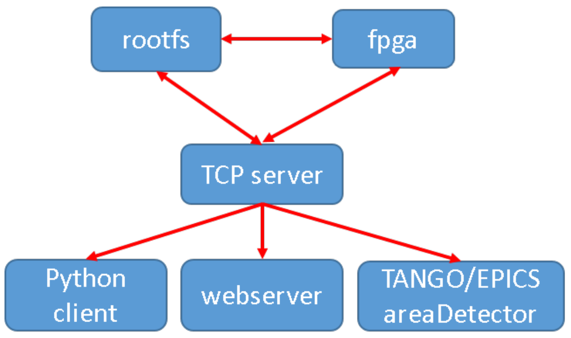

Figure 3: PandABlocks software stack.

The TCP server runs on the Zynq PS and acts as an interface to the FPGA registers and serves as a bridge between the FPGA and remote client systems. The server publishes two sockets, one for configuration control, and the other for streamed data capture. The configuration control socket accepts simple ASCII commands and returns all data in readable ASCII format. The data capture socket supports no commands and simply streams captured data as ASCII text or in a lightly structured binary format. A variety of clients can access the TCP server, including EPICS and TANGO. For graphical configuration and interrogation, a webserver and web-GUI were developed, based on the Malcolm middlelayer framework, developed by DLS for continuous scanning applications [9].

At the core of the PandABlocks framework, providing flexibility and modularity, is the concept of Functional Blocks (FBs) [10]. FBs represent specific functions or hardware resources, and can be both reconfigured and rewired at run-time, i.e. without the need to rebuild the FPGA logic. Hardware FBs usually access external I/O; examples are LVDS and TTL input and output blocks, and position encoder inputs and outputs. Examples of 'soft' FBs include 5-input look-up tables, set/reset gates, clock dividers, pulse generators and sequencer blocks.

Each FB has its own set of configuration and status registers (CSRs) allocated and all FB can access shared control and data buses. By default, 128 KB of memory is allocated for the CSRs. This is divided into 32 pages of 4 KB, with each page addressing one functional block. This is further subdivided to allow for up to 16 instances of each FB, providing up to 64 registers, each 32-bit wide, for each block instance.

Figure 4 shows a diagram representing the blocks and the bus structure. The *bit_bus* can accommodate 128 single control bits, and is used to communicate status signals between FBs. The *pos_bus* can carry up to 32 data words, each of 32 bits, and is used for transferring position type signals between blocks, for example position encoder signals between process steps. Mulitplexers on the position and bit type inputs to blocks allow the arbitrary wiring of signals between blocks. One special block, *Position Capture (PCAP)*, is responsible for the capture of the signals on the pos and bit bus, via DMA transfer to the Zynq PS subsystem.

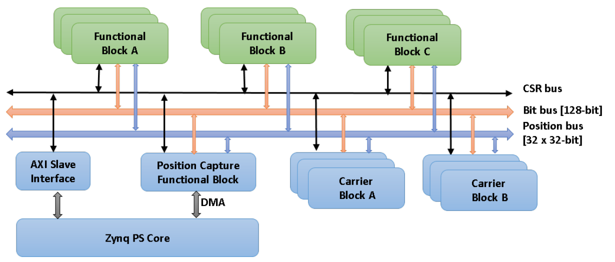

Figure 4: Functional block architecture.

The web-GUI provides a graphical representation of the connections between functional blocks for a given application. Blocks available within the design can be wired together depending on the data type of the block's ports, and parameters can be set and read back for each block. An example is shown in Fig. 5 for a bi-direction raster, or 'snake' scan. In this example, two encoder positions are input to a sequencer block, via the pos bus. The sequencer block is configured to trigger a detector using an LVDS output, based on comparing the position values from the two encoders with a table of compare points. The PCAP block is also triggered at the same time as the output, to provide timestamping and capture of other encoder values.

BUILD AUTOMATION

Whilst PandABlocks provided ease of configuring and wiring functional blocks at run-time, the task of adding new functional blocks to a design, and creating arbitrary combination of SFP and FMC application modules, could be cumbersome and was often prone to error. It was therefore very desirable to automate these processes as much as possible. The generation of the *control module* for each FB, for interfacing the CSR buses, was already automatically generated, as was the allocation of signals to the position and bit buses. This was done using Python and the *jinja2* templating engine, with the VHDL files being produced from templates, and Python variables to keep track of the bus usage. A recent effort was made to extend the autogeneration to include other necessary HDL files, as well as tcl files used in the FPGA build process.

A secondary goal of this work was to facilitate the addition of new hardware targets to the framework. With this in mind, a distinction is made between files which are target-dependant and those that are generic to PandABlocks.

Figure 6 illustrates the new autogeneration framework. The configurations are defined through a set of text-based *ini* files. For each configuration, an *app.ini* file defines the set of soft blocks (as well as SFP and FMC blocks) that are required, as well as specifying the target hardware. For each target, a *target.ini* file defines the hardware, or carrier, blocks available to it, as well any physical constraints specific to the carrier such as the number of SFP and FMC ports, which the Python code will ensure are not exceeded in the *app.ini* file. For the purpose of autogeneration, SFP and FMC blocks are treated as 'soft' blocks, as multiple FMC and SFP based applications are supported, and in the case of PandABox, a different application can be implemented on each of the three separate SFP ports.

The register definitions for each FB are defined in a *block.ini* file. This file specifies the field-type for each register, for example, *param* for constants, *pos_mux* and *bit_mux* for multiplexing signals on the pos and bit bus respectively, and *pos_out* and *bit_out* for appending outputs to the buses. It also specifies additional information that needs to be conveyed to the build system, such as IP that needs to be included in the build, and additional constraints that need to be read. The *block.ini* files from the 'carrier' and 'soft' blocks are input to *'generate_app.py'*, where they are parsed and used alongside the templates to produce the generated HDL and tcl files. The generated files are then combined with the carrier HDL and tcl files, as well

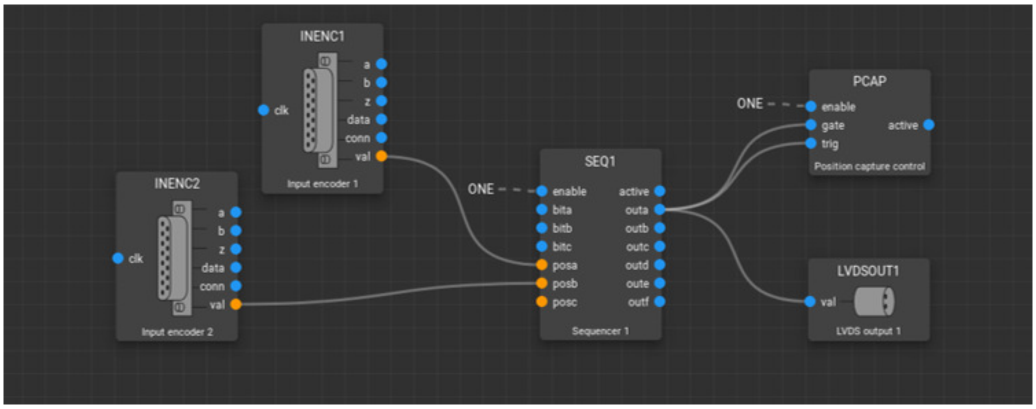

Figure 5: Example of web-GUI for a snake scan.

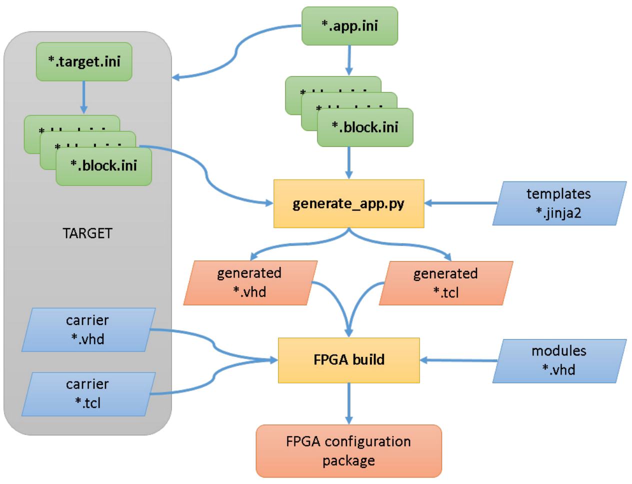

Figure 6: Flow diagram for auto-generated build processes.

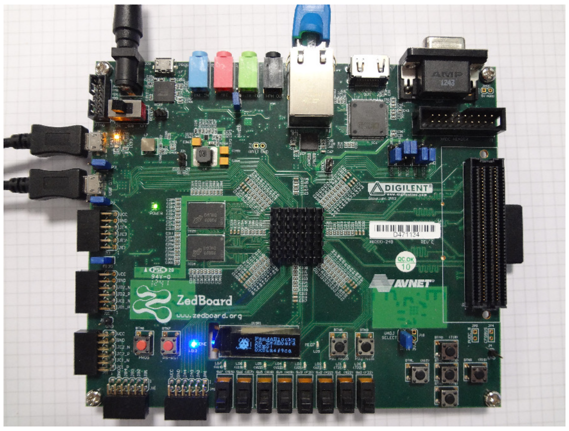

Figure 7: ZedBoard running PandABlock demo app.

the module HDL files, to produce the FPGA configuration package.

Therefore in order to add a new functional block, a user would need to write their functional HDL code, define the registers they require in a *block.ini* file, and add the required number of instances of the block to the *app.ini* file. This is a substantial improvement over the previous method, which required the creation of at least three HDL files for each new block, as well as manual editing of several other files.

ADDITIONAL HARDWARE TARGETS

ZedBoard Demonstration

In order to demonstrate the porting of PandABlocks to a different SoC device, a demonstration was made using the Digilent ZedBoard development platform, Fig. 7 [11]. The ZedBoard featues a Zynq-7020 SoC with Artix-7 equivalent PL fabric, with various peripherals such as an OLED display, user programmable switches and LEDs, an LPC-FMC connector, and several PMOD header for general purpose I/O.

In order to demonstrate the interaction of PandABlocks with the ZedBoard hardware a simple functional block was created which allows the user to read the state of the eight switches, display a value on the eight LEDs, and choose between different text fields to be displayed on the OLED. Being a smaller and slower FPGA than that on the PicoZed, it was not possible to achieve timing closure with the full complement of PandA blocks, so only a small subset of the soft blocks were included in this demo *app*. Other differences between the ZedBoard and PicoZed included details of the memory and USB chips; the former required changes to both the devicetree and to the uboot configuration.

The steps necessary to port PandABlocks from the PicoZed to the ZedBoard were the following: generating a new Zynq PS block design in Vivado IP Integrator for the Zynq-7020; generating the FSBL; editing of the devicetree to account for component differences; and creating a new *target.ini* to define the hardware blocks available to the system. A new *app.ini* was also created to specify the soft blocks to include in the design and to specify the ZedBoard target.

SOLEIL UFX Detector Application

Another example of the PandABlock framework being employed on other target hardware is in the DAQBox for the control and readout from a UFXC32k hybrid pixel detector, developed at SOLEIL [12]· This is also based around a

PicoZed-7030 SoM, coupled to a custom carrier board, Fig. 8. The carrier is essentially a compact version of the PandA carrier, with 3 SFP ports and one FMC connector, but without features such as encoder inputs, LVDS I/O, or Slow FPGA. A custom FMC module was developed to interface to the detector board via high speed cables, and image data is streamed from the three SFP ports to a server via UDP/IP on point-to-point links.

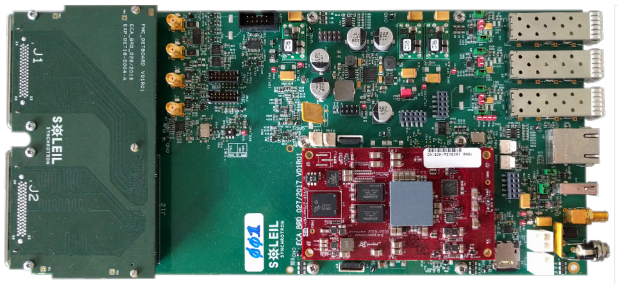

Figure 8: DAQBox carrier board with PicoZed SoM and custom FMC module.

The PandABlocks framework is used with custom SFP and FMC blocks, to provide a register interface to TANGO via TCP. As the carrier board is designed around the PicoZed SoM, no hardware related changes were needed to the Zynq PS configuration. Alongside the addition of the user-defined logic for the SFP and FMB blocks, the necessary changes to the framework for this target consisted of creating a *target.ini* file specifying the hardware blocks available on the carrier, and an associated *app.ini* file. A minor modification was necessary to the TCP server code for programming the FPGAs, to account for the removal of the Slow FPGA on the carrier.

CONCLUSION

PandABlocks is a versatile framework for Zynq-based systems, based on the modular blocks offering on-the-fly configuration and wireability. Developed for the PandABox hardware for advanced beamline scanning applications, it is being used being on several beamlines at DLS and SOLEIL. Recent changes to the framework have been made to facilitate the addition of new functional blocks to a design, as well as to target additional hardware platforms.

REFERENCES

[1] I. S. Uzun *et al.*, "PandA Motion Project - A Collaboration Between SOLEIL and Diamond to Upgrade Their 'Position and Acquisition' Processing Platform", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 302–305. doi:10.18429/JACoW-ICALEPCS2015-MOPGF098

[2] PandABox project on Open Hardware Repository, https://www.ohwr.org/project/pandabox/wikis/home

[3] Quantum Detectors Ltd., https://quantumdetectors.com/

[4] S. Zhang *et al.*, "PandABox: A Multipurpose Platform for Multi-technique Scanning and Feedback Applications", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 143–150. doi:10.18429/JACoW-ICALEPCS2017-TUAPL05

[5] Y.-M. Abiven *et al.*, "PandABox: A Multipurpose Platform Adapted for Multi-technique Scanning and Feedback", in *Proc. 11th Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16)*, Campinas, Brazil, Oct. 2016, pp. 67–71. doi:10.18429/JACoW-PCAPAC2016-THHWPLI001

[6] Avnet PicoZed SoM, http://zedboard.org/product/picozed

[7] Micro-Research Finland Oy, http://www.mrf.fi/

[8] D-TACQ Solutions, http://www.d-tacq.com

[9] T. M. Cobb *et al.*, "Malcolm: A Middlelayer Framework for Generic Continuous Scanning", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 780–784. doi:10.18429/JACoW-ICALEPCS2017-TUPHA159

[10] C. J. Turner *et al.*, "PandABlocks Open FPGA Framework and Web Stack", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1539–1542. doi:10.18429/JACoW-ICALEPCS2017-THPHA068

[11] ZedBoard, http://zedboard.org/product/zedboard

[12] G. Thibaux *et al.*, "Development of a New Data Acquisition System for a Photon Counting Detector Prototype at SOLEIL Synchrotron", presented at the 17th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOMPR005