Using FPGA-Based AMC Carrier Boards for FMC to Implement Intelligent Data Acquisition Applications in MTCA Systems Using OpenCL

Mariano Ruiz, *Senior Member, IEEE*, Antonio Carpeño, Daniel Rivilla, Miguel Astrain, *Graduate Student Member, IEEE*, A. Piñas, and Victor Costa

***Abstract*— The micro telecommunications computing architecture (MTCA) standard is widely used in developing advanced data acquisition and processing solutions in the big physics community. The number of applications implemented using commercial advanced mezzanine cards (AMC) using AMD-Xilinx and Intel field-programmable gate array (FPGA) systems on chip is growing due to the flexibility and scalability of these reconfigurable hardware devices and their suitability to implement intelligent applications using artificial intelligence and machine learning techniques. This article presents the specific design methodologies for hardware acceleration proposed by both FPGA manufacturers. Comparative results are obtained from two different software/hardware setups using two different AMCs, one based on Intel FPGA Arria 10 and another based on Xilinx ZynqMP. This article illustrates the process of how to modify the board support package, required by the hardware acceleration methodology, to implement the JESD204B and low-voltage differential signaling (LVDS) interfaces with the FPGA mezzanine card (FMC) modules containing the ADCs, to prepare the AMC cards to implement such kind of applications. The data acquisition and processing implementation inside these reference designs, with both languages OpenCL and high-level synthesis (HLS), is described. An important feature, needed for many applications in the big physics field, is the interface with the Experimental Physics and Industrial Control System (EPICS) software framework using the ITER Nominal Device Support (NDS) framework, which is briefly described.**

***Index Terms*— Field-programmable gate array (FPGA), hardware acceleration techniques, high-level synthesis (HLS), micro telecommunications computing architecture (MTCA), OpenCL, system on chip (SoC).**

I. Introduction

TWO emergent technological factors favor the development of advanced data acquisition and processing solutions on micro telecommunications computing architecture (MTCA) platforms [1], [2]. On the one hand, integrated devices are based on field-programmable gate arrays (FPGAs) [3], [4], which may or may not include an embedded processing system (typically an ARM-Cortex-based hard-core). On the other hand, the growing potential of development tools and languages must be considered for implementing digital signal processing and machine learning applications in FPGA-based systems [5], [6]. These factors are of maximum utility when designing advanced data acquisition and processing systems in the MTCA standard [7], [8], [9], [10] and are commonly implemented in advanced mezzanine card (AMC) carrier boards [11].

The core of these boards is a medium-sized FPGA family device connected to an FPGA mezzanine card (FMC) module VITA 57.1 compliant [12] and to one or more Peripheral Component Interface Express (PCIe) lanes available in the backplane connector of the chassis. The manufacturers provide two kinds of solutions. First, some systems use a conventional FPGA, which receives the data from one or more FMC modules and send the processed samples to the main computer or host through PCIe or Ethernet interfaces. Second, systems also feature a processing element of one or several ARM cores inside the device next to the programming logic, the so-called system-on-chip (SoC) devices. It is in the programmable logic fabric of these devices where the logic in charge of performing more complex operations with the data received from the FMC module is implemented, while the use of an embedded processor relieves the system of the need for an external processor. Both architectures have advantages and drawbacks and are intended for specific fields of application. Moreover, several commercial AMC cards are now available that mount both Intel[1] FPGA and AMD-Xilinx devices. This article aims to demonstrate the development of advanced data acquisition systems in MTCA using the hardware acceleration methodologies proposed by both manufacturers. The suggested design cycle is based on the following:

1) the use of FPGAs or SoCs available in the AMC boards;
2) the integration of high-speed analog-to-digital converter (ADC) modules in FMC format connected to the FPGA by low-voltage differential signaling (LVDS) [13] and JESD204B standard interface defined by JEDEC organization [14];

Manuscript received 13 February 2023; accepted 7 March 2023. Date of publication 10 March 2023; date of current version 16 June 2023. This work was supported by the Comunidad de Madrid under Grant PID2019-108377RB-C33, Grant MCIN/AEI/ 10.13039/501100011033, and Grant PEJ-2019-AI/TIC-14507.

The authors are with the Instrumentation and Applied Acoustic Research Group, Universidad Politécnica de Madrid, 28031 Madrid, Spain (e-mail: mariano.ruiz@upm.es).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2023.3255554.

Digital Object Identifier 10.1109/TNS.2023.3255554

[1]Registered trademark.

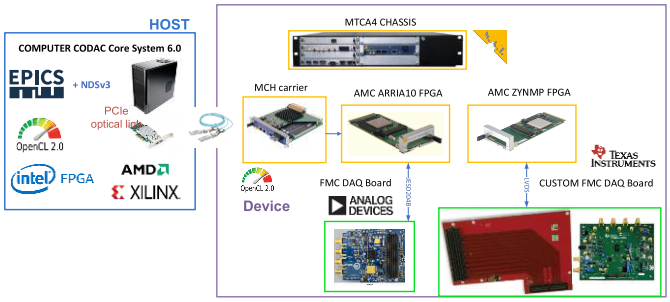

Fig. 1. MTCA hardware platform and software tools used for the application development.

3) the development of data acquisition control and processing tasks developed using OpenCL [15] and/or high-level synthesis (HLS) [16];
4) the implementation of software layers interfacing with the Experimental Physics and Industrial Control System (EPICS) framework [17].

II. Hardware and Software Setup and General View

Fig. 1 shows the MTCA platform implemented using commercial-off-the-shelf elements (COTS). The Linux host is connected to the MTCA Carrier Hub (MCH) using a PCIe $\times 16$ card. The chassis includes the power supplies and two AMC carrier cards with an FMC board installed in each one. One of them mounts a Xilinx SoC device, and the other mounts an Intel FPGA device. The software environment needed for the development is based on Vitis, Vivado, and Petalinux 2021.1 for the Xilinx design cycle; and Quartus Pro 2019.4 and OpenCL SDK for the Intel design cycle. The Linux platform used for the implementation of the system is the ITER Control and Data Acquisition (CODAC) Core System and the ITER Nominal Device Support (NDS) framework [16].

The MTCA chassis contains the following.

1) The MCH carrier board (NAT-MCH-PHYS80).
2) AMC carrier card for FMC mounting an Arria10-SoC device (NAMC-ARRIA10-FMC), with an AD-FMCDAQ2-EBZ from Analog Devices (High-speed ADC 1.0 GSPSs. JESD204B interface) installed.
3) AMC carrier card for FMC mounting a ZynqMP Ultra-scale device (NAMC-ZYNQMP-FMC), with a custom FMC mounting an Analog Devices AFE5808 (ADC eight channels, 65MSPS LVDS interface) installed.
4) The MTCA chassis (PowerBridge RackPak/M4-2) and the host PC are connected by an optical fiber (NPCIEx8-Opt-QSFP-uplink).

Each AMC carrier contains FPGA-based hardware that allows two possible approaches for the hardware acceleration methodology implementation depending on the use or not of the embedded ARM processor. Fig. 2 shows the simplified solution diagram that does not use the internal processor, while Fig. 3 shows the opposite case. In both configurations, the programmable logic fabric contains the board support package (BSP, using Intel terminology) or the platform (using Xilinx terminology), which is the hardware that implements the

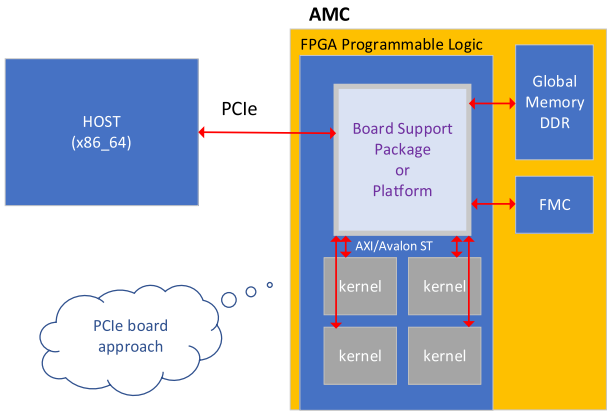

Fig. 2. Block diagram of the hardware implemented in the AMC using an FPGA device. BSP and kernels.

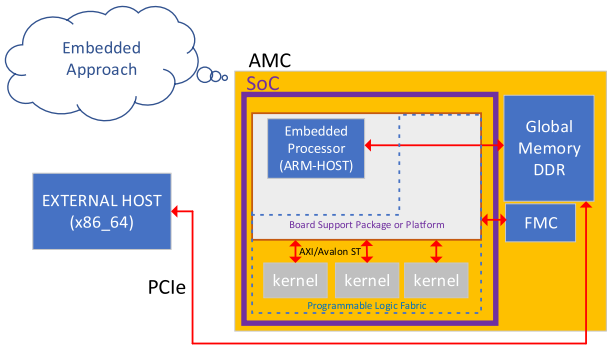

Fig. 3. Block diagram of the hardware implemented in the AMC using an SoC device. BSP and kernels.

necessary logic to support the kernels execution and the data movement and implement the physical interface with the FMC module using hard-core elements such as GigaBit Transceivers and PCIe. The blocks responsible for the double data rate (DDR) memory management are also implemented here, with Hardware Description Languages (HDLs), such as VHDL or Verilog, or by intellectual property (IP) integration. The fabric also implements the hardware to acquire and process the data, which, in our use cases, is implemented using pieces of code written in c-like languages, such as HLS or OpenCL, which are known by the name of kernels. The design of the high-speed serial interfaces available in the FMC, or the memory management blocks, is usually a complex operation. This fact could hamper the intended goal of speeding up the design cycle. However, this does not represent a major problem because this task is carried out only once when the platform is created and then used by the final user as a black box.

The differences between both approaches come when coping with the design of the software part of the application. The host implements the OpenCL Runtime to support the configuration process and the data retrieving, together with the user code, which is application dependent. This runs in the ARM processor connected to the architecture by the Advanced eXtensible Interfaces (AXI) for SoC devices or in an external computer connected to the architecture by PCIe interface for the FPGA-only devices.

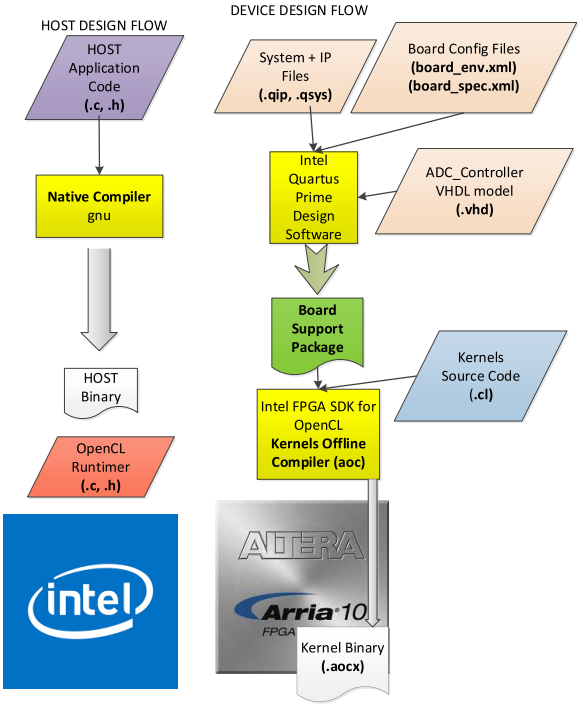

Fig. 4. Intel FPGA hardware acceleration design cycle using OpenCL SDK for FPGA devices interfacing with the host using the PCIe interface.

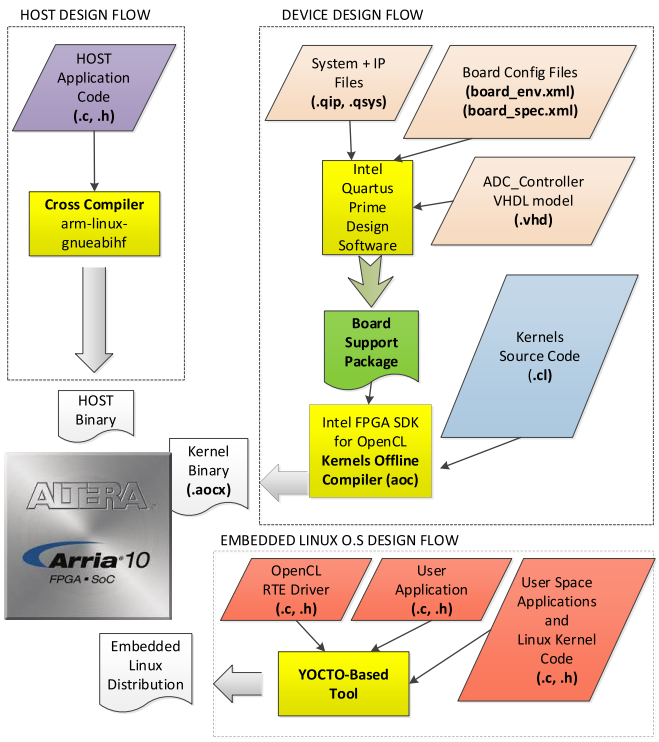

Fig. 5. Intel FPGA hardware acceleration design cycle using OpenCL SDK for SoC devices.

It is worth highlighting the use of the hardware acceleration model proposed by Intel FPGA and Xilinx to develop the kernel in charge of data acquisition and processing. This part of the process mainly improves the development time because it is designed using c-like code instead of HDL. Furthermore, the communication with the host is also resolved on the BSP or the platform. Therefore, the final user is not aware of the complexities of this part of the application and can focus on the processing algorithm design.

III. Design Cycles Using the Hardware Acceleration Approach

Intel FPGA uses different design cycles, shown in Figs. 4 and 5, to create the BSP depending on whether the solution is for an Arria SoC device (SX subfamily) or an FPGA-only device (GX subfamily). In both cases, a BSP is provided as a reference design. These basic BSPs contain all the elements needed to manage the kernels execution and data movement between the host and the device, although they can be extended or adapted to any hardware platform after a porting process [18], [19], and this is a common need for advanced acquisition designs. The goal is to create the minimum hardware, with the support infrastructure to add hardware acceleration elements (kernels) and specific user logic. This logic must correspondingly include PCIe or Avalon interfaces with the external or embedded processor, and other elements such as clock generators, reset controllers, clock domain crossing bridges, standard programmable interface (SPI) the preferred standard for ADC configuration, and the digital lines for FMC circuits control and status overseeing. To carry out the transportation of samples from the ADC to the FPGA, high-speed serial interfaces, such as JESD204B or LVDS, are used and placed inside the BSP. Finally, the hardware to access the global DDR memory and other memory management elements, such as the snoops and coherency control logic, must be placed in the BSP. All these elements are created and compiled with Quartus Prime Pro and are placed in a fixed partition of the FPGA.

Then, the rest of the application-dependent acceleration logic, comprising one or several kernels, is added to the fixed partition using incremental compilation and partial reconfiguration strategy. Kernels are connected to the static part of the programmable logic through Avalon-Streaming interfaces and receive the acquired samples through these high-speed interfaces and perform the data processing algorithm. All these operations are transparent to the final user and performed by the OpenCL compiler, whose result is a bitstream with extension ".aocx."

Regarding the software part of the cycle, shown to the left of the figures, Intel FPGA adds a set of software layers that allow the management of this hardware, called run-time environment (RTE) and the API for $\times 86$ and ARM platforms. Both solutions share the process to create the user application responsible for the application-specific system control and management, though with the use of an ARM cross-compiler in the case of SoC targets. When designing an SoC solution, however, it is also necessary to create the embedded Linux distribution to be executed by the embedded processor. As shown in the bottom part of Fig. 6, Intel provides a Yocto-based tool [20] to help in the development of these software layers as well.

Regarding Xilinx FPGA families, the manufacturer provides two design cycles: one for SoC-based solutions and another for

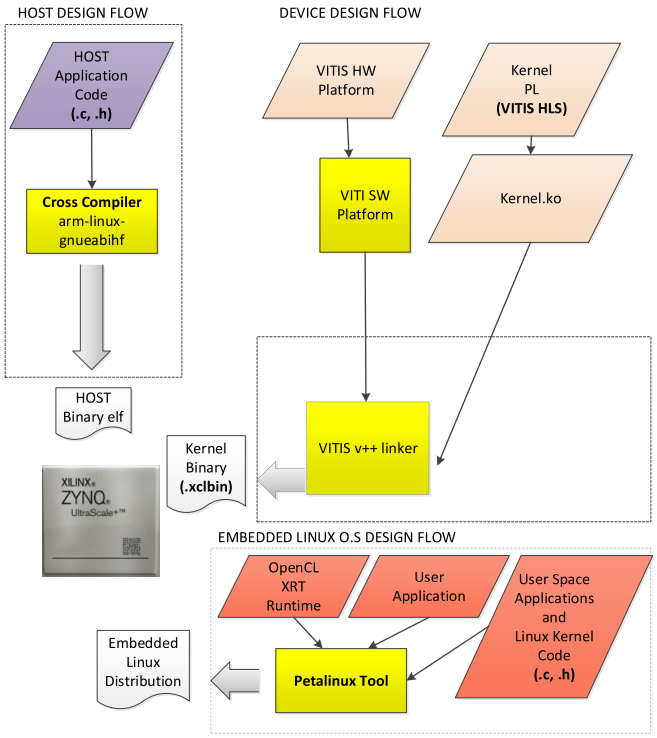

Fig. 6. AMD-Xilinx design cycle for hardware acceleration using the VITIS environment for embedded systems.

Xilinx's PCIe-based platform. However, only the former can be adapted to custom applications because PCIe architectures are not open and only work with Alveo boards, which do not allow the platform customization to include hardware to address application-specific needs. Thus, only the SoC-based design cycle is shown in Fig. 6. The reference platforms for SOC-based solutions are provided for some commercial hardware [21], although they can be modified to be used with user designs. There are two variations, one that uses the conventional configuration and the other that allows partial configuration through the Dynamic Function eXchange feature.

The device contains ZynqMP UltraScale$^+$, which embeds an ARM-based processor system that runs the user software application installed in an embedded Linux distribution. Within the programmable logic fabric of the FPGA there is an architecture with components similar to those available for the Arria10 BSP. All these elements are created and compiled with Vivado and are placed in an FPGA fixed partition. The kernels implement the specific configuration of the system and the data acquisition and processing and are connected to the static part of the programmable logic via AXI-streaming buses. As shown in the right of Fig. 6, the kernel code is designed, written, and debugged by using the Vitis tool. The whole system is compiled with Vivado, guided by the Vitis linker, and a bitstream file is generated with an extension ".xclbin," in a transparent manner for the user.

The Xilinx Dynamic Function eXchange (DFX) feature allows changing some blocks of logic, while other areas of the system are still working. This allows configuring the FPGA with new kernels on the fly without requiring reprogramming.

The software cycle, shown in the left of Fig. 6, is similar to Intel with only two differences; first, the API interacts with the Xilinx Run Time (XRT); second, to create the embedded Linux distribution, Xilinx provides the Petalinux tool [22].

To manage and communicate with the kernels implemented in the FPGA, the user application running in the host must use de functions contained in the OpenCL standard API, which is part of the specification. In this regard, Intel implements the 1.2 version of the standard but not completely. However, the API implemented by Xilinx is a complete 1.2 version. The most important feature affected by this fact is that Xilinx OpenCL application can make use of "out of order" commands, allowing the design of easier parallelism of kernel execution, thus increasing the throughput and reducing the latency of operations.

Another important feature inherent to OpenCL or HLS applications is using "pragmas" or "attributes" to lead the compilation toward faster kernels. These elements are mixed with the c-like code and allow the user to determine more closely how the hardware is implemented in the FPGA. To this regard, it is important to note that the attributes used by the Intel SDK for OpenCL are oriented explicitly to OpenCL, while the Xilinx pragmas are shared by HLS and OpenCL applications. As a result, the code is less portable to other OpenCL hardware platforms, such as CPUs or GPUs.

IV. Implementations for the NAMC Arria10 and NAMC Zynq Boards

This section illustrates the application of the acceleration design cycle for both manufacturers by means of two paradigmatic use cases. Fig. 7 shows a slightly simplified block diagram of the hardware implementation for the NAMC-ARRIA10-FMC use case. This example meets two objectives: first, it supports using the Analog Devices AD-FMCDAQ2-EBZ [23] module containing two ADC and DAC channels accessible through the JESD204B interface, and second, it implements the BSP using the Intel FPGA methodology to deploy acceleration hardware kernels developed in OpenCL.

Inside the dotted red square are the modules comprising the BSP. Blocks in green are available in the OpenCL reference model for Arria10-GX provided by Intel FPGA [18], [19]. This part comprises the elements needed to allow the management and control of kernel execution and the movement of data from the host to the global memory and vice versa. Among these blocks is the "Kernel Interface," which allows the control of kernels from the host application. The "Mem Controller" includes the External Memory Interface (EMIF) to manage the external DDR memory (global memory shared by the host and the device) and elements to manage arbitration such as the snoop to ensure coherency between the host and device writings and readings. The "PCIe" block performs the communication between the host and the device through the PCIe interface, together with the necessary blocks to construct Avalon-MM transfers to the rest of the elements of the BSP and to ensure safe clock domain crossing. The "JTAG" block allows user interaction with the system with the System Console in Quartus, mainly for debugging purposes.

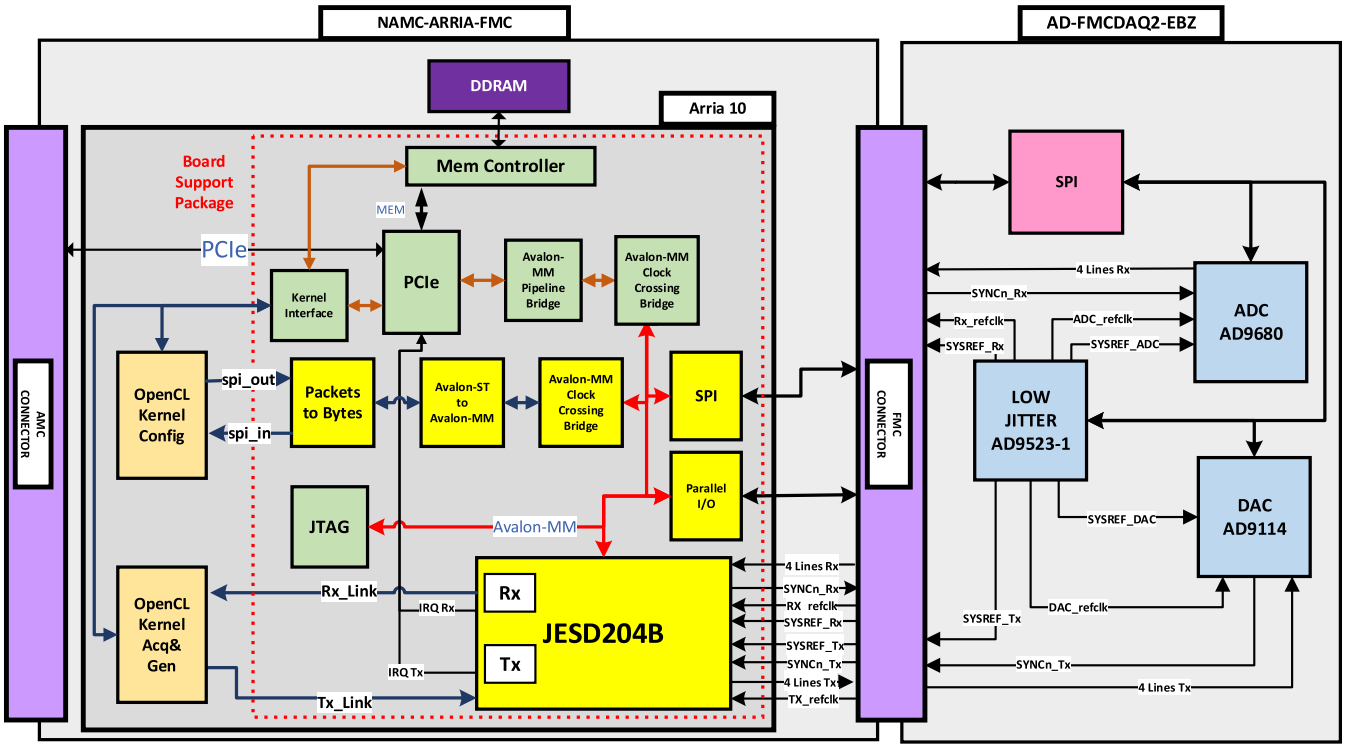

Fig. 7. Details of the hardware implementation for the NAMC-ARRIA10-FMC card.

This basic OpenCL reference design has been ported to meet the specific configuration of the AMC board and to allow the management and control of the FMC module. Considering that the resources available in most acquisition modules are very similar, a set of generic hardware blocks has been developed aiming at reusing them with other commercial cards.

These blocks can be found in yellow in Fig. 7.

1) The JESD204B Rx and Tx block receives and sends a high amount of data at a very high speed from/to the FMC.
2) The SPI interface to configure the common resources in an ADC FMC module (low jitter clock generator, ADC, and DAC).
3) Several GPIO lines controlling some specific FMC module signals.

This part of the BSP, as stated in previous sections, should only be designed once during the process of porting. Due to the use of RTL code and IP integration skills, this task demands an engineer with expertise in low-level FPGA design. There are several points that must be considered while designing this logic. The kernel-related logic uses its specific clock domain, while the FMC module, SPI interfaces, and JESD204 interface use their respective clocks. Therefore, safe clock domain crossings must be implemented. This task is not difficult because it can be achieved by using Avalon Clock Bridge IPs. Kernels can only communicate with the IO elements through resources included in the OpenCL specification, namely, IO channels or pipes, which Intel implements by means of Avalon-ST interfaces. Therefore, Avalon-MM to Avalon-ST adapters must be used [24]. All these resources are accessible, for control and management, via PCIe from the host, JTAG, or the kernel itself. This allows a very versatile way of controlling the FMC module according to the necessities of each application.

The kernels developed in the use case can be found in orange. One is in charge of the configuration of the FMC module; the other is responsible for receiving samples from the ADCs and sending them to the host, taking the samples from the host, and sending them to the DACs. The OpenCL developer generates these kernels without knowing the BSP details.

Fig. 8 shows a simplified block diagram of the hardware implementation for the NAMC-ZYNQ-FMC use case. This example meets two objectives: supports the use of a custom FMC module including the ADC device, an AFE5808A by Analog Devices [25], and implements the platform using Xilinx methodology to deploy acceleration hardware kernels developed in HLS or OpenCL code.

Inside the dotted red square can be seen the modules comprising the platform. Blocks in green are available in the reference model for ZynqMP provided by Xilinx [21], and blocks in yellow are the responsibility of the platform designer. This part includes the processing system interacting with the block responsible for the DDR global memory management. Other generic blocks are clock generators, reset controllers, and AXI4 interconnects used to communicate kernels with the host for control and data movement. Two additional blocks are needed: the LVDS interface with the ADC and the SPI controller (to configure both the ADC and other specific resources in the FMC). The necessary logic to control the kernel execution and the movement of data from the host to the kernels and vice versa is more straightforward than the one of Intel FPGA. The reason is that while Intel introduces in the BSP blocks performing the complex task of control of kernel

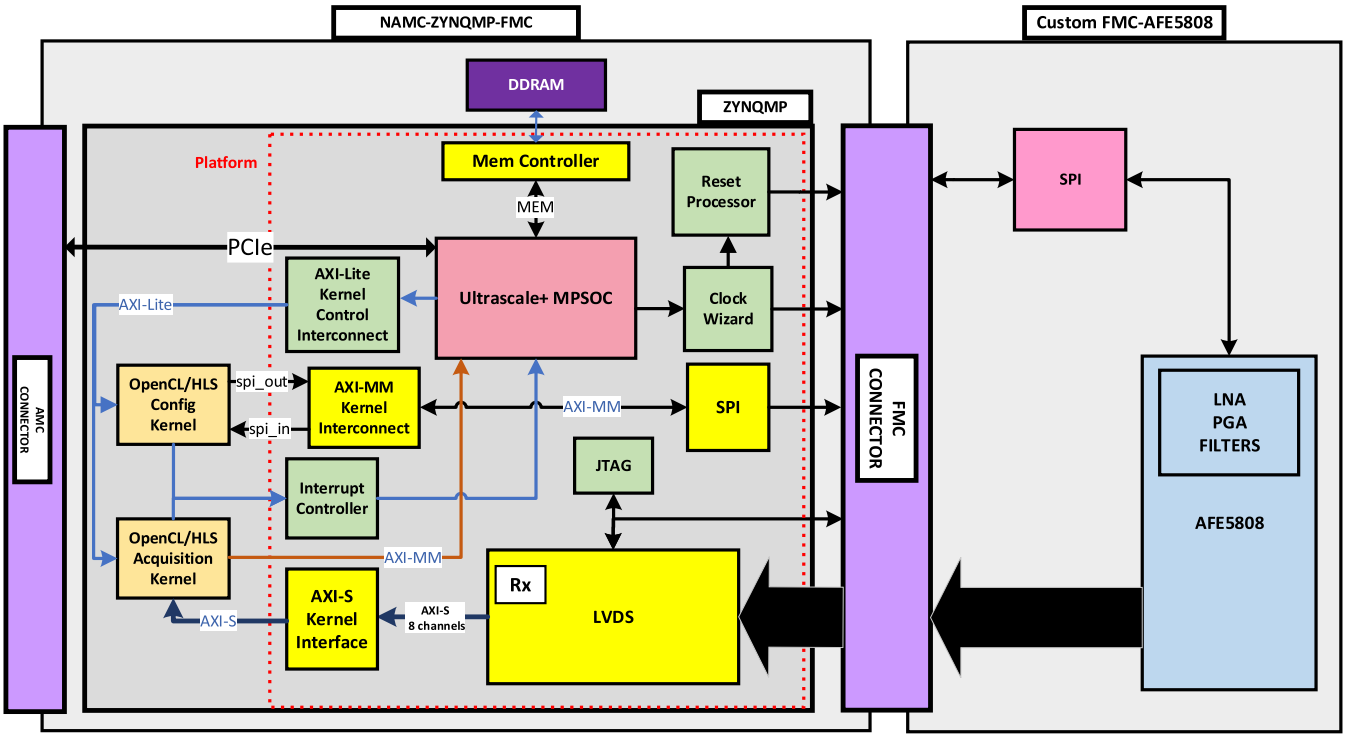

Fig. 8. Details of the hardware implementation in the NAMC-ZYNQ-FMC card.

execution or direct communication between kernels and memory, Xilinx prefers to perform these control tasks in the host through the XRT library [26]. In the FPGA part, it is enough to implement AXI-MM interfaces regardless of whether the kernels are implemented in HLS or OpenCL. In fact, one advantage of the Xilinx design cycle is that the same platform can be used for kernels written in HLS or OpenCL, which adds flexibility for the algorithm development. In both cases, the fixed part of the design, the whole system except for the kernels, should not be modified when implementing different applications, though this cannot be surely stated in this case, as explained later in this section.

The advantage of an HLS/OpenCL design is that the application developers can focus on the algorithm implementation in a c-like code, forgetting about complexities, such as the communication with the host, the FMC module management, or the LVDS/JESD204B interface implementation.

The design of the JESD204B block can be accomplished by using Intel or Xilinx-licensed IPs. These IPs are very powerful and flexible, and thus easily configurable to match the FMC module needs. However, Analog Devices provides an open-source solution to implement the interface, the JESD204 framework [28]. Though cheaper, it is more difficult to create an OpenCL-compatible system using this option because it demands deeper FPGA and JESD204B specification knowledge.

Once both systems have been reviewed and the elements that make up the BSP or the platform have been described, it is necessary to highlight an important difference between both design cycles regarding the final user, the OpenCL or HLS developer. In the case of Intel FPGA, once the BSP has been ported, it can be used to develop acceleration applications based on OpenCL without modification. Thus, there is no need for the final user to know FPGA details or complex tools such as Quartus Prime. In the case of Xilinx, the platform needs small modifications in the hardware, even for common issues such as the use of additional buffers to communicate kernels among them, without the intervention of the host, for example. This is easily achieved by Intel using channels or pipes, resources contemplated by the standard but which are not implemented by Xilinx. This prevents the use of the methodology from being completely unaware of the hardware that is working behind the scenes running in the FPGA. Ultimately, this circumstance has a significantly negative impact on the portability of the OpenCL code to be run on other hardware platforms.

V. Acquiring Data Using OpenCL and HLS-Based Kernels

As mentioned above, the acceleration methodology focuses on implementing the kernels that are added to the BSP hardware in the final implementation. The following paragraphs describe the solutions implemented for both AMC boards using a ping-pong scheme for continuous data acquisition and processing and a second implementation for single-event data acquisition and processing.

A. Ping-Pong Solution Using Intel FPGA OpenCL for AD9680

The ping-pong implementation (see Fig. 9) mechanism uses a set of pipes or channels [24] (a resource available in OpenCL to communicate kernels efficiently) to notify the interested kernels that the operation is finished, and a buffer is ready

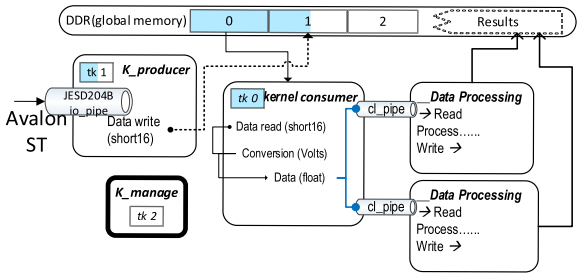

Fig. 9. Kernel logic implemented to support the ping-pong strategy for continuous data acquisition and processing.

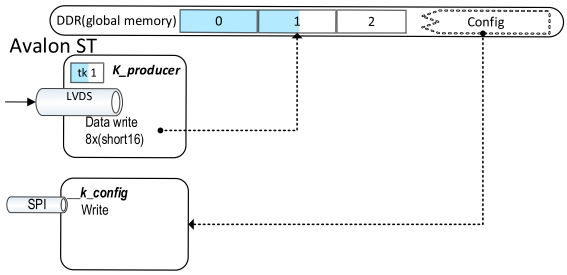

Fig. 10. Kernel logic implemented to support single-event data acquisition.

for use. The producer reads a new data packet from the ADC, comprising eight samples (128 bits), every kernel clock cycle at a rate of 140 MHz, and stores it on global memory. The global memory buffer is divided into subbuffers, and a token is generated when a subbuffer is filled. The consumer kernel reads the data from global memory, takes the samples out of the packet, and distributes the proper samples to as many pipes as processing kernels expecting these subbuffers. The consumer kernel runs at max kernel clock speed, while the producer is bound JESD204B link clock, at a rate of 125 MHz. This is an advantage because the processing kernels can process data at higher speeds than they are produced. The processing kernels read data from the consumer and write the results to global memory, from which they are sent to the host; a banked memory architecture ensures enough throughput for the data to reach the host memory. There is a specific kernel synchronizing the producer–consumer mechanism that waits for the consumer to send the token of the buffer just used to send this token (which is the offset that marks the beginning of the buffer) to the producer, so it can be filled with new data.

The resources used by this ping-pong implementation interfacing with the ADC at a sampling rate of 1 GS/s are given as follows:

1) 15% of logic elements and 9% of memory blocks for the BSP;
2) 2% of logic elements and 3% of memory blocks for the consumer–producer kernels logic.

B. Acquiring Data From AFE5808

Implementing single-event data acquisition using the HLS language in Vitis is straightforward in the platform generated for the NAMC-ZYNQ-FMC (see Fig. 10). This application only requires two kernels. One kernel configures the ADC parameters through the SPI interface (LNA, PGA gains, and other front-end analog parameters), and a second kernel waits for the specific number of samples received through the AXI stream. The samples acquired are saved in the global DDR memory to be analyzed by the HOST application. The resources needed in this application are given as follows:

1) 7% of LUT, 1.44% of LUTASMEM, 7% of REG, and 18% of BRAM for the BSP;
2) 2% of LUT, 0.5% of LUTASMEM, 1.21% of REG, and 20% of BRAM for the configuration and data acquisition kernels.

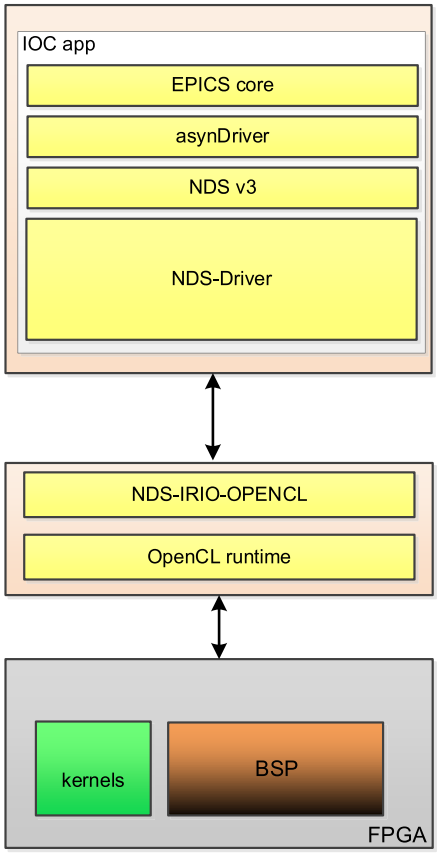

Fig. 11. Software layers implemented to manage the hardware implementation from EPICS.

VI. Interfacing the Hardware Implementation With EPICS Using NDS

The software layers developed (see Fig. 11) to manage the functionally deployed with the kernels in both AMC solutions have been implemented using EPICS and the NDS framework [7]. Using the NDS layer, there is a specific software driver, NDS-IRIO-OpenCL [9], to manage the kernels implemented in the hardware with the help of the OpenCL Runtime. The NDS driver connects with EPICS SCADA using the NDS-EPICS layer.

VII. Conclusion

The main conclusion derived from the implementation and the methodology followed are given next.

1) Simplification of the development of advanced ad complex hardware applications using OpenCL and HLS. Comparing the effort to implement the whole solution using RTL design using HDL languages, using

OpenCL/HLS notably reduces the development time. The effort to develop the BSP is related to the interface to the specific hardware elements (FMC interface, basically) and still needs the use of a design flow based on HDL. However, the data movement and processing implementation is eased due to the functionality provided by HLS and OpenCL.

2) Simplifying the development of specific software modules to manage the transfers to move data from the host to the device and vice versa, and synchronize each system element's operation. In addition, the use of OpenCL runtime avoids the use of custom Linux kernel modules and API to interface with the FPGA programmable logic.
3) There are reference designs maintained and updated by Xilinx and Intel FPGA for the BSPs. This has an impact on the maintenance cycle and the management of hardware obsolescence. In addition, these reference designs can be easily adapted to add custom interfaces, according to the needs of a specific use case.
4) The performance obtained and the resources used are expected and acceptable for the use cases implemented in this work.

REFERENCES

[1] *MicroTCA Base Specification*, PICMG, Wakefield, MA, USA, Jan. 2020.

[2] R. S. Larsen, "PICMG xTCA standards extensions for physics: New developments and future plans," in *Proc. 17th IEEE-NPSS Real Time Conf.*, May 2010, pp. 1–7.

[3] S. M. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," *Proc. IEEE*, vol. 103, no. 3, pp. 318–331, Mar. 2015, doi: 10.1109/JPROC.2015.2392104.

[4] J. Hasler, "The rise of SoC FPAA devices," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2022, pp. 1–8, doi: 10.1109/CICC53496.2022.9772732.

[5] T. Leppanen, P. Mousouliotis, G. Keramidas, J. Multanen, and P. Jaaskelainen, "Unified OpenCL integration methodology for FPGA designs," in *Proc. IEEE Nordic Circuits Syst. Conf. (NorCAS)*, Oct. 2021, pp. 1–7, doi: 10.1109/NorCAS53631.2021.9599861.

[6] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis," *IEEE Access*, vol. 5, pp. 2747–2762, 2017, doi: 10.1109/ACCESS.2017.2671881.

[7] M. Astrain, M. Ruiz, A. Stephen, R. Sarwar, A. Carpeo, and S. Esquembri, "Real-time implementation of the neutron/gamma discrimination in an FPGA-based DAQ MTCA platform using a convolutional neural network," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 2173–2178, Aug. 2021, doi: 10.1109/TNS.2021.3090670.

[8] M. Astrain, M. Ruiz, A. Carpeño, S. Esquembri, E. Barrera, and J. Vega, "A methodology to standardize the development of FPGA-based high-performance DAQ and processing systems using OpenCL," *Fusion Eng. Des.*, vol. 155, Jun. 2020, Art. no. 111561, doi: 10.1016/j.fusengdes.2020.111561.

[9] M. Astrain, M. Ruiz, A. Carpeño, S. Esquembri, and D. Rivilla, "Development of deep learning applications in FPGA-based fusion diagnostics using IRIO-OpenCL and NDS," *Fusion Eng. Design*, vol. 168, Jul. 2021, Art. no. 112393, doi: 10.1016/j.fusengdes.2021.112393.

[10] S. Esquembri et al., "Application of heterogeneous computing techniques for the development of an image-based hot spot detection system using MTCA," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 2151–2158, Aug. 2021, doi: 10.1109/TNS.2021.3087124.

[11] *Advanced Mezzanine Card Base Specification*, PICMG, Wakefield, MA, USA, Nov. 2006.

[12] *FPGA Mezzanine Card (FMC) Standard*. Standard ANSI/VITA 57.1-2019, 2021.

[13] "Understanding serial LVDS capture in high-speed ADCs," Texas Instrum. Incorporated, Dallas, TX, USA, Appl. Rep., SBAA205, Jul. 2013.

[14] *AN 803: Implementing Analog-to-Digital Converter Multi-Link Designs With Intel Arria 10 JESD204B RX IP Core*, Intel, Santa Clara, CA, USA, Feb. 2020.

[15] *Intel FPGA SDK for OpenCL Proedition Best Practices Guide*, document UG-OCL003, Intel, Santa Clara, CA, USA, May 2016.

[16] *Vitis High-Level Synthesis User Guide*, document UG1399, San Jose, CA, USA, Dec. 2022.

[17] R. Lange et al., "Nominal device support (NDSv3) as a software framework for measurement systems in diagnostics," in *Proc. 18th Int. Conf. Accel. Large Exp. Phys. Control Syst.*, Shanghai, China, Oct. 2021, pp. 1–6, doi: 10.18429/JACoW-ICALEPCS2021-TUBR01.

[18] *SDK for OpenCL Intel Arria 10 GX FPGA Development Kit Reference Platform Porting Guide*, document UG-OCL010, Intel FPGA, San Jose, CA, USA, 2022.

[19] *FPGA SDK for OpenCL Intel Arria 10 SoC FPGA Development Kit Reference Platform Porting Guide*, document UG-20052, Intel FPGA, San Jose, CA, USA, 2019.

[20] *RocketBoards Embedded Design Site*. Intel. Accessed: Feb. 10, 2022. [Online]. Available: https://www.rocketboards.org/foswiki/Main/WebHome

[21] *Xilinx Vitis Embedded Platform Source*. Accessed: Mar. 15, 2022. [Online]. Available: https://github.com/Xilinx/Vitis_Embedded_Platform_Source.Vitis_Embedded_Platform_Source/Xilinx_Official_Platforms

[22] *Petalinux Tools Reference Guide*. document UG1144, Xilinx, San Jose, CA, USA, Jun. 2021.

[23] Analog Devices. *AD-FMCDAQ2-EBZ Documentation*. Accessed: Apr. 5, 2022. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcdaq2-ebz.html

[24] *Avalon Interface Specifications*, Intel, Santa Clara, CA, USA, Jan. 2022.

[25] *AFE5808A.8-Channel Low Noise Analog Front End With Passive CW Mixer for Ultrasound*, Texas Instruments, Dallas, TX, USA, Jan. 2022.

[26] XILINX. *XRT Github*. Accessed: Feb. 10, 2022. [Online]. Available: https://github.com/Xilinx/XRT

[27] JESD204 Interface Framework. *Analog Devices*. Accessed: Dec. 22, 2023. [Online]. Available: https://wiki.analog.com/resources/fpga/peripherals/jesd204