# Integration of Hardware Acceleration Techniques in a Real-Time Framework Using FPGA Devices

C. González, *Graduate Student Member, IEEE*, M. Ruiz, *Senior Member, IEEE*, J. Nieto, A. Carpeño, A. Piñas, V. Costa, E. Barrera, G. Arranz, W. Lee, T. Tak, and A. Zagar

***Abstract*— The ITER International Fusion Experiment Organization is implementing the real-time framework (RTF) to facilitate the development, deployment, and execution of instrumentation and control (I&C) applications optimized for real-time performance using the GNU/Linux-based ITER CODAC Core System (CCS) software distribution. This contribution examines the feasibility of using hardware acceleration techniques with field-programmable gate arrays (FPGAs) to implement real-time applications in the RTF that requires specific compute-intensive functions. By combining the use of languages such as high-level synthesis (HLS) and open computing language (OpenCL) with FPGA devices, specific hardware architectures can be implemented to solve certain computational problems to gain performance and limit latency. This work shows the methodology used to integrate HLS and OpenCL in the ITER CCS and the results obtained in terms of execution time for two common processing operations, vector addition and matrix multiplication, using a commercial off-the-shelf FPGA-based device.**

***Index Terms*— CODAC Core System (CCS), field-programmable gate array (FPGA), hardware acceleration techniques, high-level synthesis (HLS), open computing language (OpenCL), real-time framework (RTF).**

## I. Introduction

THE use of specific software frameworks to develop and deploy real-time applications is essential in the control systems used in big science facilities. There are many technical approaches to implementing real-time applications, which are dependent on the final hardware target to be used (embedded system and industrial-grade computer) and the specific application (military, transport, and so on). One of these approaches is the use of specific applications running proprietary (i.e., VxWorks) or open-source real-time operating systems (i.e., FreeRTOS). A current trend in big science facilities is the use of the GNU/Linux operating system with a real-time full-preemptive kernel due to the flexibility to use the numerous software applications already implemented and integrate a great diversity of hardware platforms and the inherent cost reduction. It is important to note that GNU/Linux is not a hard real-time system and, therefore, requires specific tuning in order to meet the required real-time standards.

In the field of experimental fusion devices, some software frameworks have been implemented to simplify the development and deployment of real-time applications. The plasma control system (PCS) of the DIII-D tokamak is implemented using a distributed system based on CentOS 6 Linux OS, running real-time applications for different diagnostics and specific data analysis. This PCS is also operated in K-STAR, EAST, MAST, NSTX, and Pegasus [1]. Another important tool is the multithreaded application real-time executor (MARTe) [2], which is mainly used at the Joint European Torus (JET), RFXMod2, and other facilities [3], [4]. The ASDEX Upgrade device at the Max Planck Institute for Plasma Physics (IPP) has implemented the discharge control system (DCS) [5], which is also used at WEST [6]. ITER, the fusion energy experiment under construction, is implementing a new framework called real-time framework (RTF). ITER plasma control applications' instrumentation and control (I&C) systems and diagnostics with real-time requirements will use MARTe (version 2) and RTF as state-of-the-art solutions [7], [8], [9].

The ITER control, data acquisition, and communication (CODAC) is the system that facilitates inter-plant communication and ensures uniformity of data representation. ITER has developed a software suite designated as the CODAC Core System (CCS) to achieve this objective. The CCS's standard operating system is Red Hat Enterprise Linux (RHEL), which supports using Linux Real-Time Kernel. The ITER RTF is a suite of software tools and applications that provide common services and capabilities for creating real-time applications integrated into the CCS. The ITER RTF can execute specific tasks on isolated CPU cores, transforming them into single-task cores through the Linux Real-Time Kernel. Fig. 1 shows a simplified diagram of the ITER RTF architecture with its main elements [10]. Two nodes can be observed: one that generates information and another one that consumes it. Each node executes a real-time process that fulfills three functions: configuration of the framework, management of the framework's life cycle, and support of the RTF context. The RTF context comprises both non-real-time functions and

Received 13 May 2024; revised 26 November 2024 and 16 January 2025; accepted 22 January 2025. Date of publication 28 January 2025; date of current version 17 March 2025. This work was supported in part by Grant PID2022-137680OB-C33 funded by MCIN/AEI/10.13039/501100011033 and in part by "ERDF A way of making Europe." *(Corresponding author: C. González.)*

C. González, M. Ruiz, J. Nieto, A. Carpeño, A. Piñas, V. Costa, E. Barrera, and G. Arranz are with the Instrumentation and Applied Acoustic Research Group, Universidad Politécnica de Madrid, 28031 Madrid, Spain (e-mail: c.gonzalez@alumnos.upm.es).

W. Lee, T. Tak, and A. Zagar are with the ITER International Fusion Organization, 13067 Saint-Paul-Lez-Durance, France.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2025.3534905.

Digital Object Identifier 10.1109/TNS.2025.3534905

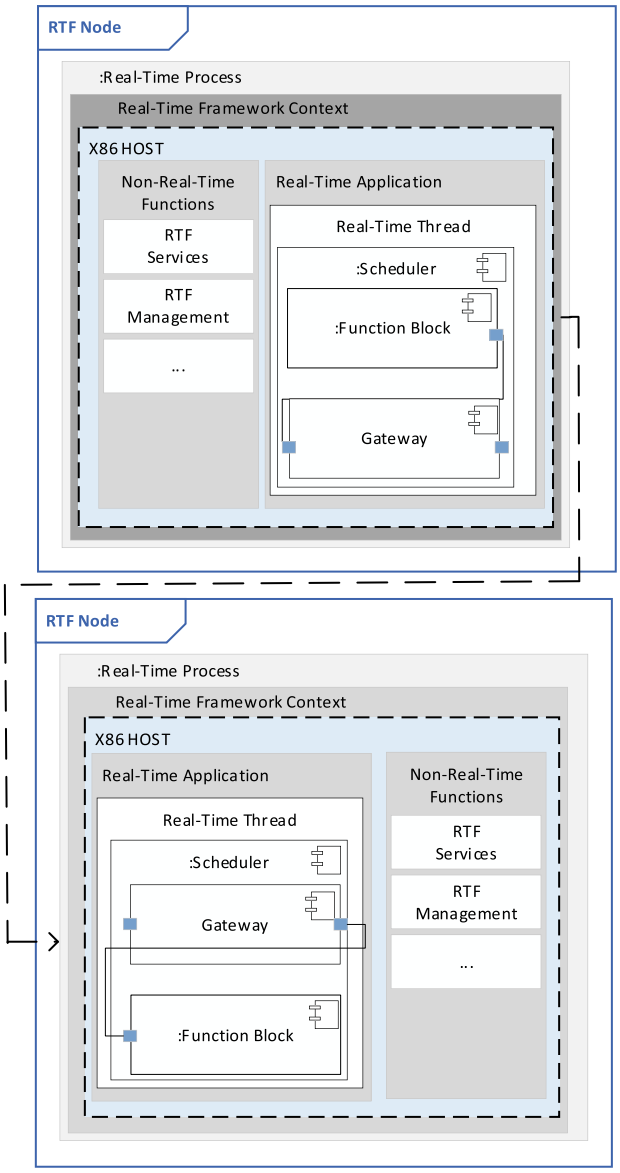


Fig. 1. Diagram with the representation of the main elements in the ITER RTF (reproduced from [10]).

the real-time application. Non-real-time functions are not part of a computational process and for which real-time constraints must not be met (i.e., logging or archiving). In contrast, the real-time application is composed of one or more threads, each of which can be assigned to an isolated CPU's core. This allows for the dedicated allocation of exclusive hardware resources to the tasks within a thread, ensuring the consistent and predictable execution of tasks within defined time constraints. Function blocks (FBs) are the fundamental units of real-time applications responsible for executing the desired operation. FBs can run in different threads, processes, or nodes, each with inputs and outputs. Each FB accepts inputs and produces outputs as they are processed. A set of preimplemented FBs is part of the RTF software framework.

It is essential to highlight that several works have already analyzed the suitability of the ITER RTF for the implementation of real applications. Another example [11] shows the use of the RTF in a Thomson Scattering diagnostic to acquire and process signal segments acquired in pulse mode with a 5-GS/s digitizer. The RTF uses several FBs to acquire and process the data using the CPUs of an industrial computer. This work also compares the performance of the FB implemented using the RTF with an independent graphical processing unit (GPU) application not connected to the RTF. Perek et al. [12] describe the evaluation of the application developed using the RTF using simulated data and also with real data acquired from a polychromator at KSTAR. The authors demonstrate that RTF is a suitable candidate for the development of I&C diagnostic systems due to its ability to meet time constraints and process data in real time. Lee et al. [13] present the details of the capabilities of RTF to be used in the ITER PCS.

The conclusions in [11] and [12] highlight the importance of reducing the computational time and latency of certain algorithms. Hardware acceleration techniques have been conceived to address this crucial problem in time-critical applications. The technique is based on the use of hardware accelerators, such as GPUs and field-programmable gate arrays (FPGAs), to implement heterogeneous applications [14]. FPGAs have demonstrated that some specific computing algorithms can be implemented, reducing the execution time and their latency using a specific design cycle proposed by the manufacturers. In particular, AMD Xilinx and Altera (IntelFPGA) have software tools to help designers in this work. This contribution aims to demonstrate the feasibility and performance of hardware acceleration techniques using AMD Xilinx FPGAs to optimize the execution time and limit the latency for RTF FBs. It also proposed the development cycle for easy integration with the ITER CCS software tools.

## II. AMD Xilinx Hardware Acceleration Design Cycle

AMD Xilinx provides two hardware acceleration design cycles corresponding to the primary product categories [15]: system-on-chip (SoC)-based solutions and peripheral component interconnect express (PCIe)-based platforms. Several manufacturers are producing PCIe cards with AMD Xilinx FPGAs. AMD Xilinx (with the Alveo family) and BittWare, among others, provide products with varying costs and performance characteristics. AMD Xilinx provides software for developing applications using these PCIe cards with the Vitis and Vivado tools and an extensive collection of libraries for developing artificial intelligence (AI) applications and audio and video processing.

Fig. 2 depicts the development flow involved in application design, as detailed in [15]. The host design flow and the device design flow are discrete operations. The section of the host design flow encompasses the software components on the host machine responsible for overseeing communication and control with the FPGA. Such components include the following: the C++ application code incorporating the logic enabling interaction with the FPGA kernels. The g++ compiler and linker are employed to compile and link the code, thereby generating an executable that is capable of communicating with the FPGA hardware. The open-source Xilinx Runtime (XRT) libraries enable low-level communication with

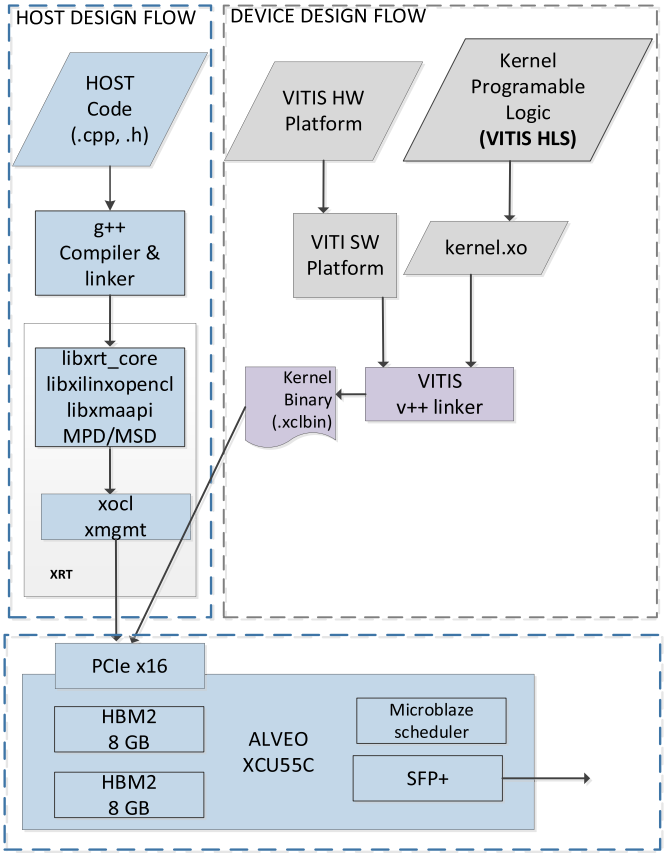


Fig. 2. AMD-Xilinx design cycle for hardware acceleration using the Vitis environment. xmgmt: PCIe management physical function. xocl: PCIe user physical function. HBM: high bandwidth memory.

the Alveo card via the PCIe interface [17]. This approach is advantageous because the software implementation is independent of the kernel implemented, except for the kernel parameters and the data types used.

The Device Design Flow section consists of kernel programming, which uses Vitis high-level synthesis (HLS). The kernel is developed using HLS in C/C++ and subsequently converted by the compiler to register transfer level (RTL) code, which is then synthesized to kernel object format (kernel.xo). The Vitis hardware platform specifies the hardware configuration, selecting the appropriate FPGA resources, including memory blocks, in accordance with the required specifications, and the Vitis software platform defines the software environment necessary for kernel integration. The v++ linker facilitates the consolidation of kernel object files and the Vitis hardware and software platforms into a binary file (.xclbin), which can then be deployed to the FPGA. The tools provided by the manufacturer allow the hardware profiling of the kernel, which simplifies the identification of performance problems and bottlenecks in the hardware-implemented.

The final section at the bottom of the figure depicts the deployment to the Alveo XCU55C FPGA accelerator card. The card has two high bandwidth memory (HBM2) stacks of 8 GB each, providing rapid access for memory-intensive applications. A MicroBlaze processor integrated into the FPGA is utilized as a scheduler for the purpose of managing kernel execution and data flow. The small form-factor pluggable plus transceiver (SFP+) permits high-speed communication through optical or copper interfaces to transfer data beyond the host system [15].

The reconfigurable functionality of the FPGA is implemented and utilized through the dynamic function exchange (DFX) feature [16]. This allows the kernel to be changed without reconfiguring all the FPGA.

## III. System Configuration and Integration in RTF

The hardware platform defined by ITER CODAC to run the real-time applications is an industrial computer based on PICMG 1.3 standards, known as an ITER fast controller, which supports the connection of PCIe devices with an external instrumentation chassis. An AMD Xilinx ALVEO U55C PCIe card has been installed, which is built around a Virtex UltraScale+ FPGA device. The fast controller runs a Red Hat Enterprise 8.5 operating system with the real-time kernel and the ITER CCS version 7.2.1 with the ITER RTF (v2.4.0). The real-time system has been configured with a profile that isolates two of the eight CPUs (0 and 1) and the interrupts of the xocl kernel driver (part of the AMD Xilinx XRT module [17]) in CPU0. The AMD Vitis Environment (version 2023.2) is also installed in the same system for development, debugging, and profiling purposes.

Fig. 3 depicts a simplified schematic of a generic RTF FB that implements a computing function using the available FPGA resources on an Alveo card. The RTF context is deployed on the fast controller. A non-real-time function, designated "Hardware Management," has been developed for the purpose of configuring the FPGA. The RTF context life cycle management service executes the Hardware Management function each time the application is loaded. This allows for the configuration of the FPGA after the setting of the FB input and output signals and parameters. Consequently, the path to the binary kernel file (.xclbin), previously obtained through the AMD-Xilinx hardware design cycle, is entered as an FB parameter. The file contains the bitstream that the FPGA will utilize to execute kernel functions. In accordance with the open computing language (OpenCL) API [17], the Hardware Management function ensures that the hardware platform is installed and operational, reads the binary kernel file, prepares the necessary data, and programs the device. To program the device, it is necessary to create an object, which is called the OpenCL program. This takes the OpenCL context, OpenCL command queues, and the binary kernel file as arguments. Both the OpenCL context and the OpenCL command queues are defined in the host before the OpenCL program is created. The Hardware Management function creates the memory buffers with their corresponding pointers for data transfer between the host and the device. The data to be transferred to the device on a one-time basis (typically parameters) are transferred by the Hardware Management function once all the configurations have been completed.

The real-time thread, which is scheduled by the RTF context life cycle management, executes the FB process method in a loop that begins upon completion of the Hardware Management function and terminates upon reception of a stop signal. Each instance of the FB process method starts by reading

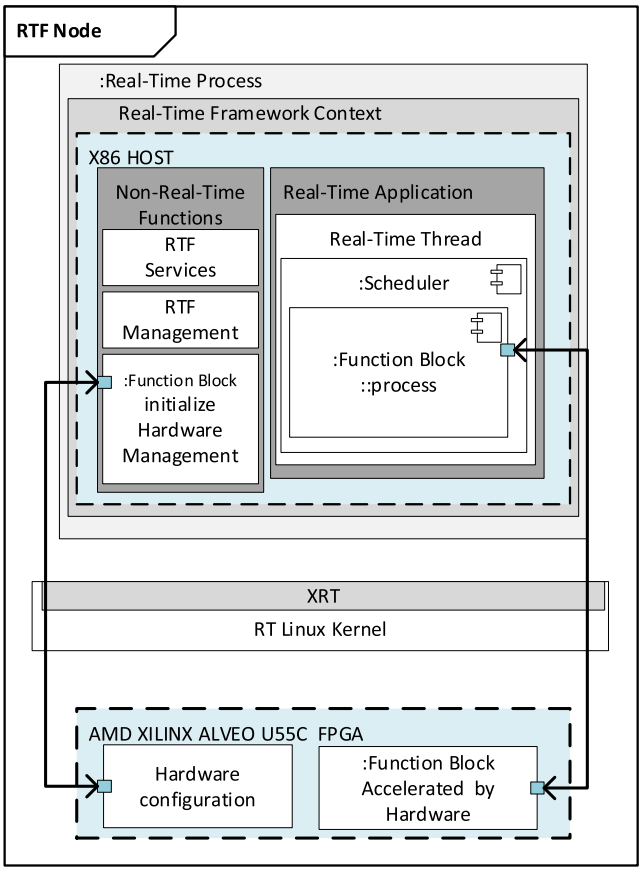


Fig. 3. Scheme of an RTF real-time process executing a single FB accelerated by an FPGA-based accelerator card.

the input data and ends with writing the output data. As the buffers created by the Hardware Management are accessible to the FB process, the input data are written into these buffers. Subsequently, the FPGA kernel is triggered, and the remaining steps are to await the FPGA results, read them, and then write them as output FB data. Given that the complexity of data processing is managed by the FPGA, in comparison, the RT FB process method is relatively simple. Even with this simplicity, the real-time thread must be assigned to an isolated CPU to ensure that no interrupts from the OS can occur and to avoid high execution time variance.

The OpenCL API is supported by the AMD Xilinx XRT (version 2.15.225) Linux Device Driver, which includes a number of libraries. Of particular relevance is the libxrt_core library, which is designed to facilitate the interaction between the OS and the FPGA device, and libxilinxopenpcl, which implements the OpenCL API for AMD-Xilinx FPGAs.

The RTF profiler is used to ascertain the execution times of FB process instances; however, this tool solely provides the cumulative time for each FB process method instance. The Vitis analyzer can be employed to ascertain the low-level execution times within the process method that is related to the use of the OpenCL runtime environment.

## IV. Results

Two use cases have been evaluated as proof of concept. The initial case study evaluates the implementation of a simple

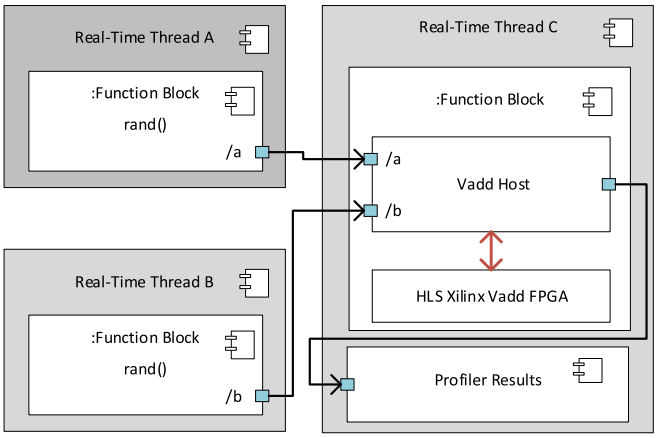


Fig. 4. Diagram with the implementation in RTF of the basic adder FB accelerated in the FPGA device.

operation and the addition of arrays of floating-point numbers by using a simple for-loop operation solution or by using the OpenBlas library function saxpy (version 0.3.26, compiled by default with $-$O2). A CPU easily solves this simple process, which is more time-consuming when the number of arrays to add is large. The RTF compiles the code using GCC g++ version 8.5.0 with the standard C++ 14 and optimized with the $-$O3 flag. No optimization techniques are applied for the HLS implementation in the FPGA, which produces a lower execution time and the use of more FPGA resources because the profiling hardware is included. This first use case helps validate the execution times obtained by both profiling methods: RTF-profiler and Vitis profiler/analyzer. The second case of study implements a more complex functionality, which is the general matrix multiply (GEMM), a more time-consuming operation for a CPU, implemented in two manners: directly in C++ without any code optimization or using the OpenBlas library function cblas_sgemm. In this second example, the FPGA implementation uses the Vitis basic linear algebra subroutine (BLAS) library for accelerating linear algebra functions (open-source) [20].

### A. Vector Add

Fig. 4 shows the deployment scheme of the RTF application, adding variable-length 1-D vectors. The RTF application executes three threads. Threads A and B are two instances of an RTF FB generating random data vectors that feed a third thread with an FB whose primary function is to compute $N$ times the addition of these vectors. Instead of using the CPU, the accelerated version writes the input data to the Alveo device's global dynamic random access memory (DRAM), coordinates the execution of the kernel, and reads the results from the Alveo DRAM. Among the RTF services, the profiler collects statistical information from each FB.

It is important to mention that the RTF applications are executed in isolated CPUs 0 and 1 (eight CPUs in total and Intel Core i7-6700K-4.00 GHz) using the SCHED_RR policy with a priority of 20, and the xocl kernel module interrupts are isolated in CPU 0. The FB using the OpenBlas library sets the number of threads to use to one. Table I

TABLE I
EXECUTION TIMES MEASURED USING THE RTF PROFILER

| N | CPU (us)[a] | CPU (us) OPENBLAS | FPGA (us) |
|---|---|---|---|
| | RTF PROF | RTF PROF | RTF PROF |
| 1 | 2 ± 1 (3) | 3 ± 1 (5) | 141 ± 18 (380) |
| 10 | 8 ± 1 (11) | 11 ± 1 (16) | 263 ± 18 (410) |
| 20 | 14 ± 1 (19) | 19 ± 1 (25) | 402 ± 30 (562) |
| 30 | 21 ± 1 (27) | 26 ± 2 (33) | 545 ± 50 (606) |
| 40 | 28 ± 2 (38) | 34 ± 2 (40) | 681 ± 50 (730) |
| 50 | 35 ± 2 (44) | 42 ± 2 (51) | 820 ± 50.0 (955) |

[a]The times measured are the mean, the standard deviation and the maximum.

TABLE II
RESULTS OF THE MEAN EXECUTION AND STANDARD DEVIATION TIMES USING AMD VITIS PROFILER FOR THE FPGA

| N | Vitis Analyzer (us) | | |
|---|---|---|---|
| | Kernel | Write | Read |
| 1 | 68 ± 20 (268) | 15 ± 20(50) | 13 ± 20 (50) |
| 10 | 191± 30 (400) | 15 ± 20(50) | 13 ± 20 (50) |
| 20 | 328 ± 50 (500) | 15 ± 20(50) | 13 ± 20 (50) |
| 30 | 472 ± 60 (600) | 15 ± 20(50) | 13 ± 20 (50) |
| 40 | 608 ± 70 (700) | 15 ± 20(50) | 13 ± 20 (50) |
| 50 | 745 ± 80 (900) | 15 ± 20(50) | 13 ± 20 (50) |

shows the results obtained with the RTF profiler for vectors of 4096 single-precision floating-point elements that added a variable number of iterations ($N$) using the CPU and the FPGA device. The FB was executed at least three thousand times to get the execution time statistics (LastExecTime) using the RTF profiler. As expected, the performance obtained with the FB using the CPU is better in all cases. The FPGA execution times include the DMA data movements (buffer writes and reads) between the host and the accelerator card's global memory and all the OpenCL runtime operations. The FPGA kernel is compiled to achieve a clock of 300 MHz.

Table II details the times measured using the AMD Xilinx Vitis profiler/analyzer (kernel execution and the data movement, write, and read). As this algorithm always uses the same amount of input and output data (32 kB for the two input buffers and 16 kB for the results), the values measured for the writing and reading operations for the different $N$ values are always the same. The results depend on the CPU performance, the computer's architecture and PCIe configuration, the Linux kernel parameters configuration, the specific configuration of the Linux real-time parameters, and other Linux applications system resource usage.

Figs. 5 and 6 display one of the timing traces obtained with the Vitis profiler/analyzer when the add operation is computed 50 times using two arrays of 4096 single floating-point elements. The profiler is employed to ascertain the sequence of operations undertaken by the host application to

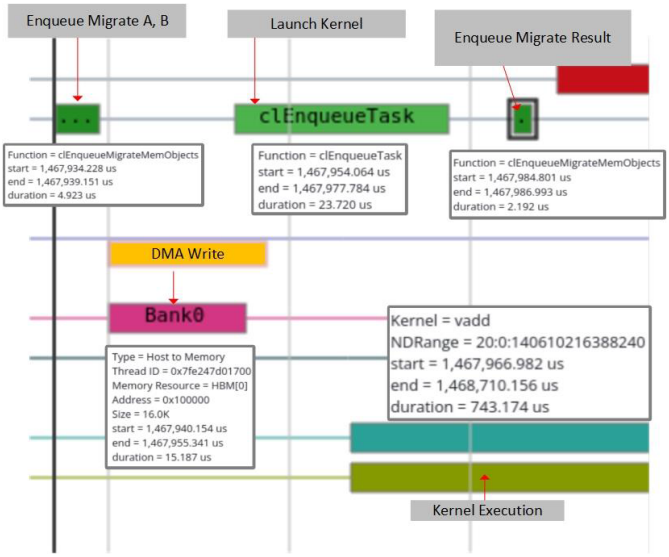


Fig. 5. Profiler measurement for the kernel implementing the adding operation 50 times. Initial part.

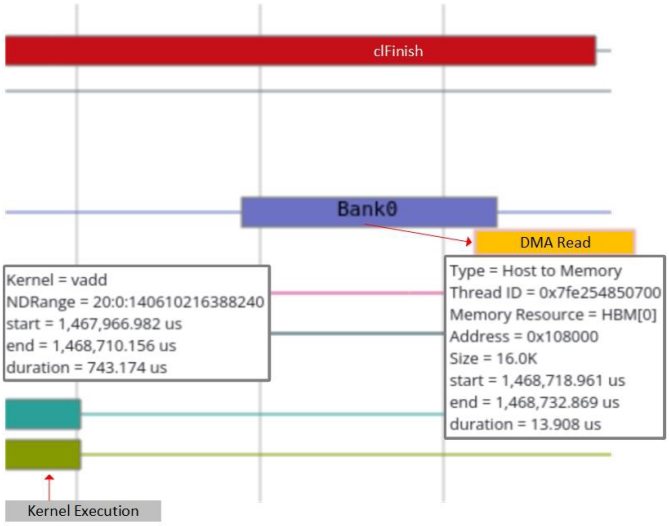


Fig. 6. Profiler measurement for the kernel implementing the adding operation 50 times. Final part.

enqueue data (OpenCL call to enqueueMigrateMemObjects), trigger the kernel execution (enqueueTask), and retrieve results when the kernel is finished (clfinish operation). It is important to detail that every enqueueMigrateMemObjects call triggers the DMA data movement between the host and the FPGA. The Vitis profiler shows the traces and the time spent in all these operations (the gray boxes in Figs. 5 and 6 detail the results reported for the DMA transfers and kernel execution). A discrepancy of 43 $\mu$s is observed between the mean value in the third column of Table I (the total kernel execution time measured with RTF Profiler for the FB) and the sum of the mean values measured with the Vitis profiles (see Table II). This discrepancy is due to the time that elapses from the call to enqueue operation until the Write DMA transfer begins and the time that elapses between the Read DMA transfer ends and the call to clFinish returns: the use of the tracer of the XRT module, the overhead added by RTF, and the data copies

TABLE III
EXECUTION TIMES OBTAINED FOR FLOAT GEMM USING OPENBLAS BENCHMARK

| Square Matrix Size | CPU execution time (µs) mean value-standard deviation[a] |
|---|---|
| 64 | 9 ± 1 |
| 128 | 53 ± 4 |
| 256 | 368 ± 17 |
| 512 | 2656 ± 23 |
| 1024 | 20475 ± 15 |
| 2048 | 160379 ± 34 |
| 3072 | 532935 ± 73 |

[a]Results obtained using OpenBlas benchmark using only one thread

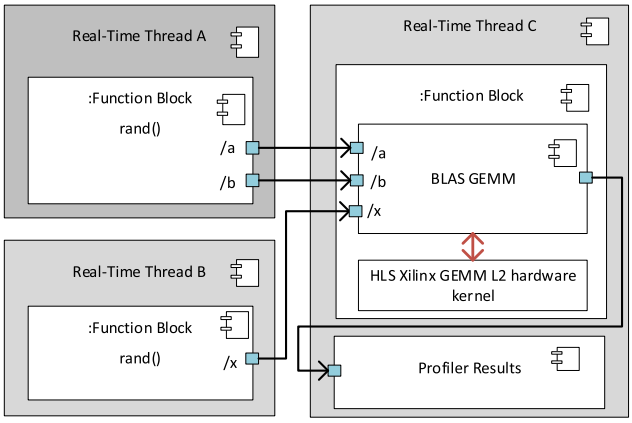


Fig. 7. Diagram with the implementation in RTF of the GEMV FB accelerated in the FPGA device.

needed between the buffers used by RTF and the buffers used by the OpenCL runtime.

### B. Matrix-Vector Multiplication

The RTF implementation follows a similar scheme to that described in the previous section (see Fig. 7). Two threads produce random data matrixes, which feed a third thread that computes the matrix multiplication using the CPU or the FPGA. The FPGA version is implemented using the open-source code provided in the AMD Xilinx Vitis library repository, which is an FPGA-accelerated implementation of the standard BLASs [20]. The library includes a C++ implementation, designated as $L2$, which are predefined kernels that demonstrate how to construct hardware for BLAS operations, such as GEMM. These $L2$ kernels combine primitive functions, designated as $L1$, with data mover modules, creating a hardware block where data are transferred efficiently between memory and computation modules on the FPGA. The GEMM kernel implements the operation (1), where $A$, $B$, and $X$ are matrices of $N \times N$ elements

$$C = A \times B + X. \tag{1}$$

In order to make the best possible comparison between the Vitis FPGA implementation and the code running in the CPU with the same functionality, the tests use a nonoptimized CPU implementation for GEMM and the implementation provided in the OpenBlas project. Table III contains the results obtained with the OpenBlas benchmark executed for only one thread [openblas_set_num_threads(1)] repeated 1000 times.

TABLE IV
RTF PROFILER RESULTS FOR NONOPTIMIZED CPU CODE, CPU-OPENBLAS, AND FPGA WITH VITIS BLAS LIBRARY

| Square Matrix Size | CPU (ms)[a,b] | CPU (µs) OpenBlas | FPGA (µs) |
|---|---|---|---|
| | RTF PROF | RTF PROF | RTF PROF |
| 64 | 0.21 ± 0.01 (0.217) | 10.75±0.1 (12.4) | 86 ± 2 (89) |
| 128 | 2.06 ± 0.01 (2.08) | 62.66±2.1 (72.2) | 189± 3 (204) |
| 256 | 21.00 ± 1.70 (24.35) | 373.64±2.6 (400.76)) | 714± 6 (744) |
| 512 | 247.61 ± 0.60 (248.75) | 2711.17±8.3 (2800.50) | 3790± 5326 (3883) |
| 1024 | 1731.33 ± 1.52 (1737.40) | 20852.50±18.5 (20978.63) | 23350± 50 (23525) |
| 2048 | 38000.00 ± 20.00 (38100) | 162000 ± 91 (162350) | 149000 ± 54 (150000) |
| 3072 | 150000.00 ± 100.00 (151000) | 539520 ± 50 (539650) | 488600 ± 100 (489050) |

[a]The non-optimize version performs very poorly and execution times are reported in ms. [b]The values reported are the mean, the standard deviation and the maximum value.

TABLE V
EXECUTION TIMES OBTAINED USING THE VITIS PROFILER/ANALYZER USING THE PROFILING COUNTERS

| Square Matrix Size | Vitis Profiler/Analyzer (µs) | | |
|---|---|---|---|
| | Kernel | Write* | Read* |
| 64 | 39± 5(234) | 26 ±5 (43) | 23 ± 5 (37) |
| 128 | 80 ± 3(95) | 53± 1(63) | 52± 1(64) |
| 256 | 310± 20(535) | 168± 5(180) | 167± 5(180) |
| 512 | 2785± 15(2794) | 626 ± 3(640) | 634± 3(641) |
| 1024 | 16766± 50 (16788) | 2416± 5 (2425) | 2436 ± 5 (2443) |
| 2048 | 119575 ± 100 (122137) | 9604 ± 3 (9610) | 9698 ±3 (9703) |
| 3072 | 423489 ± 100 (425910) | 21600 ± 4 (21613) | 21819 ± 4 (21831) |

*From Host to DRAM

Table IV shows the results obtained by running three different RTF applications: the nonoptimized GEMM implementation, the one using the OpenBlas cblas_sgemm function, and the implementation using the FPGA acceleration. The GEMM kernel consists of a systolic array of processors that parallelize the computation of each matrix element, each performing partial computations and accumulating results to produce the final matrix product efficiently. The FPGA design uses a 300-MHz clock for the kernel, and it has been implemented without hardware profiling to avoid using more area and performance penalties. The results show that RTF adds an important overhead of around 20% (of the value obtained with the benchmark) for the operation of size 64 and 128 square matrixes and a lower value (2%) for 256, 512, 1024, 2048, and 3072. The accelerated version performs poorly for square matrixes in the size range of 64–1024. For 2048 and 3072,

the FPGA implementation provides better results than the OpenBlas CPU implementation.

Table V contains the results obtained with the Vitis profiler/analyzer, which requires a specific compilation that adds the profiling hardware and the activation of the profiler traces in the XRT runtime. The design has been implemented using a 300-MHz clock. It is worth clarifying that the profiler provides valuable information about the time spent to move the data using the DMA. It is important to highlight again the influence on the total execution time of the data movement between the host and the accelerator, which is around 100% for small-size matrixes (64, 128, and 256) and around 10%–15% for the bigger ones, 2048 and 3072.

## V. Conclusion

No previous work has been published showing the integration of hardware acceleration techniques into the ITER RTF. The research and development work completed has demonstrated the following key aspects.

- The acceleration methodology proposed by AMD Xilinx has been successfully integrated and tested into the ITER CCS (version 7.2.1) and its RTF framework (version 2.4.0).
- An ITER fast controller has been configured to use an ALVEO U55C device using the Xilinx XRT open-source module and the Red Hat real-time preemptive kernel (4.18.0-348.23.1.rt7.153).
- The profiling results obtained using the RTF and the Vitis profilers show accurate and reproducible timing values with a delimited latency but require a specific configuration of the kernel parameters and the isolation of the specific interrupts of the xocl kernel module to reduce the maximum latency in the operations.
- Two RTF applications have been implemented, showing the performance of the hardware accelerator using the FPGA and validating the timing measurements done with the RTF profiler and the Vitis profiler/analyzer. The RTF added overhead has been estimated. Compared with optimized libraries for CPUs, such as OpenBlas, the performance of the FPGA is only superior when using matrixes of the largest values ($1024 \times 1024$ and $2048 \times 2048$). This comparison has been made by running the OpenBlas benchmark using only one thread. It is obvious that the gain obtained compared with no optimized libraries is very notable, as expected.
- It is necessary to investigate in more detail how AMD Xilinx is implementing the control logic that executes the kernel hardware implementation to identify the origin of the variability observed in the kernel execution time.

Finally, the development and validation cycle used in this research can be directly applied to the instrumentation systems implemented with the MTCA platforms, which allow the use of boards with UltraScale MPSoC [21], which supports the use of this acceleration technique. This implementation will require analyzing the implementation of the XRT module for this ARM-based platform.

## Acknowledgment

The views and opinions expressed herein do not necessarily reflect those of the ITER Organization, Saint-Paul-Lez-Durance, France.

## References

[1] M. Margo et al., "Current state of DIII-D plasma control system," *Fusion Eng. Des.*, vol. 150, Jan. 2020, Art. no. 111368, doi: 10.1016/j.fusengdes.2019.111368.

[2] A. C. Neto et al., "MARTe: A multiplatform real-time framework," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 479–486, Apr. 2010, doi: 10.1109/TNS.2009.2037815.

[3] G. Manduchi, A. Rigonii, T. W. Fredian, J. A. Stillerman, A. Neto, and F. Sartori, "MARTe2 and MDSplus integration for a comprehensive fast control and data acquisition system," *Fusion Eng. Des.*, vol. 161, Dec. 2020, Art. no. 111892, doi: 10.1016/j.fusengdes.2020.111892.

[4] G. Cseh, G. Kocsis, B. Kovács, E. Skáre, and T. Szepesi, "Integrating EDICAM into the MARTe framework," *Fusion Eng. Des.*, vol. 191, Jun. 2023, Art. no. 113516, doi: 10.1016/j.fusengdes.2023.113516.

[5] W. Treutterer et al., "ASDEX upgrade discharge control system—A real-time plasma control framework," *Fusion Eng. Des.*, vol. 89, no. 3, pp. 146–154, Mar. 2014, doi: 10.1016/j.fusengdes.2014.01.001.

[6] J. Colnel et al., "Adapting DCS real time framework for WEST plasma control," *Fusion Eng. Des.*, vol. 129, pp. 24–28, Apr. 2018, doi: 10.1016/j.fusengdes.2018.02.050.

[7] F. X. R. Alvarez, "Design of a distributed data acquisition system for the ITER's neutral beam," M.S. thesis, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, 2023. Accessed: Apr. 25, 2024. [Online]. Available: http://hdl.handle.net/2117/395518

[8] W. Lee et al., "Real-time framework for ITER control systems," in *Proc. ICALEPCS*, Shanghai, China, Oct. 2021, pp. 45–49.

[9] J. A. Snipes et al., "ITER plasma control system final design and preparation for first plasma," *Nucl. Fusion*, vol. 61, no. 10, Sep. 2021, Art. no. 106036, doi: 10.1088/1741-4326/ac2339.

[10] D. Soklic, "Real-time framework user manual for real-time application developers and deployment engineers," ITER, Saint-Paul-lez-Durance, France, Tech. Rep. 8G45ZU, 2024.

[11] M. Kadziela, B. Jablonski, P. Perek, and D. Makowski, "Evaluation of the ITER real-time framework for data acquisition and processing from pulsed gigasample digitizers," *J. Fusion Energy*, vol. 39, no. 5, pp. 261–269, Nov. 2020, doi: 10.1007/s10894-020-00264-3.

[12] P. Perek et al., "Evaluation of ITER real-time framework in plasma diagnostics applications," *Fusion Eng. Des.*, vol. 192, Jul. 2023, Art. no. 113623, doi: 10.1016/j.fusengdes.2023.113623.

[13] W. Lee et al., "A case study of the real-time framework for the implementation of the ITER plasma control system," *Fusion Eng. Des.*, vol. 193, Aug. 2023, Art. no. 113702, doi: 10.1016/j.fusengdes.2023.113702.

[14] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, vol. 10, pp. 131788–131828, 2022, doi: 10.1109/ACCESS.2022.3229767.

[15] *Vitis Unified Software Platform Documentation Application Acceleration Development*, document UG1393 (v2023.2), Dec. 13, 2023. Accessed: Apr. 25, 2024. [Online]. Available: https://docs.amd.com/r/2023.2-English/ug1393-vitis-application-acceleration/Getting-Started-with-Vitis

[16] *Vivado Design Suite User Guide: Dynamic Function EXchange (UG909)*. Accessed: Apr. 25, 2024. [Online]. Available: https://docs.amd.com/r/en-U.S./ug909-vivado-partial-reconfiguration

[17] *Xilinx Runtime (XRT)*. Accessed: Apr. 25, 2024. [Online]. Available: https://github.com/Xilinx/XRT

[18] *The OpenCLTM Specification*. Accessed: Apr. 25, 2024. [Online]. Available: https://www.khronos.org/opencl/

[19] *OpenBLAS : An Optimized BLAS Library*. Accessed: Nov. 2, 2024. [Online]. Available: http://www.openblas.net/

[20] *Vitis Blas Library*. Accessed: Apr. 25, 2024. [Online]. Available: https://docs.amd.com/r/en-U.S./Vitis_Libraries/blas/overview.html

[21] M. Ruiz, A. Carpeño, D. Rivilla, M. Astrain, A. Piñas, and V. Costa, "Using FPGA-based AMC carrier boards for FMC to implement intelligent data acquisition applications in MTCA systems using OpenCL," *IEEE Trans. Nucl. Sci.*, vol. 70, no. 6, pp. 993–1000, Jun. 2023, doi: 10.1109/TNS.2023.3255554.