sciendo

VOLUME 18, N° 2 2024
Journal of Automation, Mobile Robotics and Intelligent Systems

SoC-FPGA Based Concept of Hardware Aided Quantum Simulation

Submitted: 24th February 2023; accepted: 13th July 2023

Jacek Długopolski, Jakub Czerski, Mateusz Knapik

DOI: 10.14313/JAMRIS/2-2024/9

Abstract:
Contemporary industry and science expectations towards technological solutions set the bar high. Current approaches to increasing the computing power of standard systems are reaching the limits of physics known to humankind. Fast, programmable systems with relatively low power consumption are a different concept for performing complex calculations. Highly parallel processing opens up a number of possibilities in the context of accelerating calculations. Application of SoC (System On Chip) with FPGA (Field-Programmable Gate Array) enables the delegating of a part of computations to the gates matrix, thereby expediting processing by using parallelization of hardware operations. This paper presents the general concept of using SoC FPGA systems to support the CPU (Central Processing Unit) in many modern tasks. While some tasks might be really hard to implement on an FPGA in a reasonable time, the SoC FPGA platform allows for easy low-level interconnections, and with such virtualized access to the hardware computing resources, it is seen as making FPGAs, or hardware in general, more accessible to engineers accustomed to high-level solutions. The concept presented in the article takes into account the limited resources of cheaper educational platforms, which, however, still provide an interesting and alternative hybrid solution to the problem of parallelization and acceleration of data processing. This allows encountered limitations to be overcome and the flexibility known from high-level solutions and high performance achieved with low-level programming to be maintained without the need for a high financial background.

Keywords: *FPGA, SoC, quantum circuit, parallel computing, web service, accelerating calculations*

1. Introduction

Nowadays, practically every sector of human activity benefits from technological solutions. Accelerating and optimizing existing processes and automating new ones will improve quality and user comfort. Growing demand for computing power has caused the creation of new integrated circuits with computing cores that are clocked at even higher frequencies. It carries with it an increase in the energy required and, consequently, a rise of emitted heat that has to be dissipated.

For this reason, engineers has introduced systems composed of many computational units, enabling parallel operations. Another perspective option is the FPGA chip – a matrix of independently configurable logic gates, which provides naturally massive parallelism that increases the computing power while consuming much less energy.

Programmable gate arrays consist of a large number of advanced logical units, very fast RAM (Random Access Memory) blocks and specialized DSP (Digital Signal Processing) modules, all effectively surrounded by internal hardware connection buses. There are also programmable input/output blocks at logical arrays and boundaries for communication with other external computing units. Currently, FPGA systems are commonly used not only for digital signal processing but for regular computing tasks as well. They find applications when highly fast data computing and process parallelization are required, e.g. in satellite software, military radars, GPS and cellular phone systems, and also in image processing, emulation of physical phenomena and in solutions that require high-speed communication networks. Commercial organizations, as well as research institutions, commonly turn to FPGA in many applications.

Standard computing systems have usually separated CPUs, GPUs (Graphics Processing Units), peripherals, devices controllers, etc. In the case of mobile processors dedicated to the smartphone and tablet market, all units described above tend to be integrated and implemented on a single chip called SoC (System on Chip). SoC systems have many benefits in the era of the IoT (Internet of Things). SoC FPGAs chips, in addition to the Programmable Logic Array available inside, also contain a ready, built-in CPU processor and dedicated very fast communication channels between the two mentioned parts. They provide the following advantages: miniaturization, integration, flexibility, computing acceleration, energy savings and low TCO (Total Cost of Ownership). A hardware description language is used to configure the FPGA. SystemVerilog and VHDL are currently the most popular solutions on the market. By the use of synthesis tools, an internal communication links list is generated. Finally, the created configuration is mapped to a given chip. When the configuration is being mapped, a problem of lack of resources may occur; hence, it is important to optimize logic functions during the FPGA application development process.

The problem of limited resources may concern, in particular, cheaper platforms, e.g. educational platforms such as the one described later in this article. The main concept elaborated in this paper is a direct answer to this resource limitation. The main contributions of the paper are as follows:

- General idea of hardware support for calculations in web services, which takes into account the CPU offload, parallelization and asynchronous data processing with limited resources.
- A practical example of the implemented above concept is the use of hardware acceleration in the form of an FPGA coprocessor on the example of an educational web service designed to perform quantum calculations, which includes:
  - Programming FPGA logic for asynchronous operation;
  - Configuration of communication between FPGA and CPU processor, and a proposal on how to constrain the representation of transferred data, important in the case of cheap and resource-limited systems and
  - Using created FPGA-based coprocessor services directly from the internal main processor.

The paper is organized as follows: Section 2 provides an overview of related works; Section 3 defines the general idea of hardware support for calculations with an emphasis on SoC FPGA; Section 4 describes communication standards and hardware details; Section 5 presents a practical example – web-based quantum toolkit similar to IBM Quantum Composer [2] with FPGA-based quantum gates operations; Section 6 summarizes the paper.

2. Related Works

Heterogeneous programmable system-on-chip (SOC) FPGA devices, which combine both general-purpose processors and reconfigurable fabrics and provide a compelling platform for many systems applications using IoT technology [11]. The authors pay attention here to the hardware abstraction. According to them, FPGA should be better virtualized in order to become available for engineers that are accustomed to software API abstractions and fast product delivery.

ASICs (Application Specific Integrated Circuits) are very expensive, and therefore, cheaper solutions are usually implemented for prototyping purposes. The time of development of an ASIC platform is also long and requires expert knowledge. For example, the authors of [7] argue that "FPGAs are less dense and slower than ASICs, but their flexibility often more than makes up for these drawbacks", in particular, as far as price and availability is concerned. The SoC FPGA can provide a proper abstraction layer. Thanks to this, such systems are much cheaper to implement compared to ASICs but still give more benefits over purely CPU-based systems.

It is worth mentioning that in the times of cloud solutions many leading companies introduce hardware in their portfolio as IaaS (Infrastructure as a Service) in order to reduce their costs. An example of such a Cloud-Scale Acceleration Architecture, using FPGA technology as a main accelerator in the data centre, is presented in [6]. Many data centres deploy FPGA in their infrastructures with two main approaches: FPGA tightly coupled to the Central Processing Unit (CPU) or FPGA as a standalone component [13]. This trend shows that FPGAs are willingly used by companies providing computing solutions. In [8], the authors propose a general framework for hardware-based web applications that take advantage of cloud-based FPGAs to make this approach more accessible to users. Their aim is to make it easier to access the benefits of FPGA without any extra effort. Very often, hardware acceleration is used in AI (Artificial Intelligence) systems, especially in data processing related to the machine learning algorithms, as shown in [12].

Undoubtedly, it must be said and admitted that the best simulation effects of quantum systems, similar to real physical phenomena, can be obtained using only programmable FPGA logic. Its massive parallelism and natural real-time capability is irreplaceable. The author of [10], using a tensor network formalism, shows one of the methods of such an approach. However, with small-capacity FPGAs, their programmable logic depletes very quickly, greatly limiting the expandability of the simulator. Then the techniques of hybrid use of SoC FPGA chips described in this article can come in very handy. Thanks to them, the simulation system can be divided between the FPGA logic and the built-in processor, leaving only the key fragments of parallel calculations for the simulation on the FPGA side.

3. General Idea of Hardware Support for Calculations

Further increasing the computing power is still possible by producing multi-core microprocessors or by building computers based on many independent computing units. Certainly, this potentially enables the use of parallel algorithms in applications requiring fast calculations. Creating useful systems for such computational architectures is not easy, either because of the need to provide fast external communication channels between processors or because of the limited ability to scale hardware designed for these systems. An increasing number of cores or processors in such systems is very difficult. FPGA technology can help here. In a single programmable integrated circuit, one can configure a whole group of independent processors and then implement any algorithm of massively parallel data processing for them. It is much easier to create some parallel applications if an add-on HPS (Hard Processor System) is integrated with the FPGA matrix in one chip, the so-called SoC FPGA chip. All fast and flexible connections between FPGA hardware blocks and HPS are then optimally implemented inside the same integrated circuit.

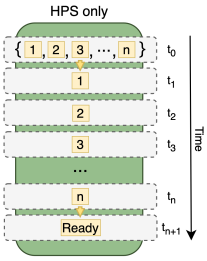

Figure 1. CPU only system

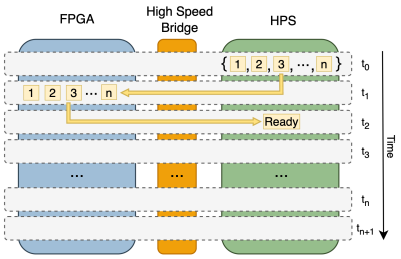

Figure 2. SoC FPGA system

Being on the same silicon, communication between CPU and FPGA consumes less power in comparison to separate chips. The integration of internal connections (bridges) between the processor and the FPGA leads to substantially higher communication bandwidth and lower latency [7]. The general concept of CPU (HPS) support through the use of parallel processing units in SoC FPGAs systems compared to CPU-only systems is presented in Figure 1 and Figure 2.

Parallel execution compared to the sequential one results in reduced runtime. Delegating some calculations to separate module results in additionally releasing the processor's resources, which, at the same time, is able to perform other tasks when programmed effectively. The FPGA matrix is a hardware component that perfectly fits this concept. A properly programmed matrix is capable of paralleling the commissioned calculations and thus returns the results faster. A sufficiently large FPGA matrix can be broken down into functional fragments, each capable of performing different tasks. This approach enables the use of hardware acceleration in a distributed system, where computation support is essential for many problems. An example of such a solution is a system consisting of many web services using support in calculations. These web services delegate their work to appropriate fragments of the FPGA matrix and then read results from them. The representation of the data sent to the matrix should be carefully considered, depending on the selected FPGA programming method. This is especially important when the system has limited resources. Processors and FPGAs integrated together can form an interesting basis for many embedded systems. This integration of broad processor functionality and the FPGA ability to perform massively parallel operations in real-time, and all these in a single chip, makes such an embedded system a much more versatile and efficient computing platform.

4. Communication Standards and Hardware Details

There are many communication standards available to transfer data between the CPU and FPGA. In general, the CPU and FPGA can be connected via PCIe (PCI Express), RIFT, CAPI or Xillybus. Also, MultiGigabit Transceivers or even Ethernet is possible [14].

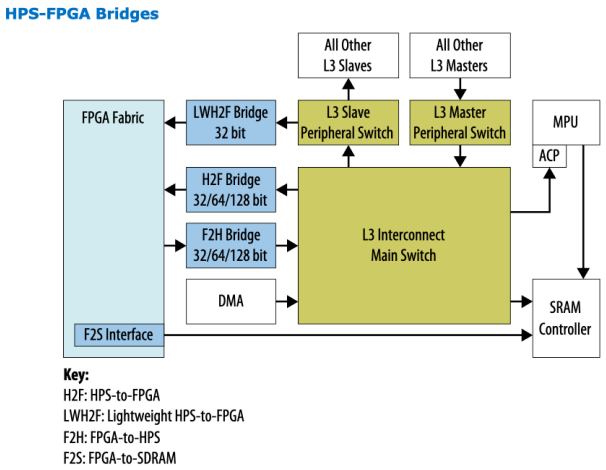

Figure 3. HPS – FPGA bridges [1]

The abundance of communication standards introduces flexibility, but at the same time, requires the implementation of an appropriate physical layer for the selected standard. In order to improve communication, SoC FPGA supports industry-standard AXI (Advanced eXtensible Interface) bus [9]. Compared to previously mentioned solutions, i.e. PCIe or Ethernet, this standard requires fewer resources and provides still better performance [14]. This can be viewed as the virtualisation of hardware resources.

The idea of hybrid data processing system based on FPGA and CPU combined in one chip, presented above, can be implemented on the Intel Cyclone V SoC FPGA chip used by the authors of this article to test the concept. The ready Atlas-SoC hardware development platform [5] was used. The system has a built-in processor called HPS (Hard Processor System) based on ARM Cortex-A9 architecture, operating at a frequency of 925 MHz, and a programmable FPGA matrix with 40 thousand logical elements. The hardware design platform has many built-in components, making it ideal for project prototyping.

The HPS has three bridges that use memory-mapped interfaces to FPGAs based on the Arm Advanced Microcontroller Bus Architecture (AMBA) and Advanced eXtensible Interface (AXI) [1]. In other words, the module responsible for the control of the FPGA system contains necessary information about the mapping of physical ports to their virtual addresses by which it has direct access to FPGA. The HPS-FPGA bridges schema is presented in Figure 3.

Lightweight AXI has a permanent 32-bit connection to FPGA and allows working in a main-agent mode, where the main is in HPS and allows access to memory-mapped FPGA ports working in agent mode. Despite the fact that AXI supports 32, 64 and 128 bits, and also works faster, according to Intel's documentation, this bridge is used for burst information transfer, therefore it is not recommended to use it to access peripheral registers in the FPGA structure. The LWH2F Bridge and the F2S Interface were used in the described here practical example.

5. Practical Example of Using the Concept

In order to demonstrate the applicability of the concept presented here, the very rapidly developing field of quantum computing has been selected. Quantum computing is characterized by a massively parallel way of processing information. Simulating this type of computation on a classic computer is more difficult the more qubits you want to simulate. Therefore, the parallel data processing potential of FPGAs can greatly facilitate this type of simulation.

Quantum computing is a field that combines computer science and quantum mechanics. It deals with using the properties of quantum systems to process information that is classically unattainable. The basic information carrier in quantum computing is the qubit, which is the quantum equivalent of the classical bit used in computer science. A qubit is described by an arbitrary linear combination of base states, in this case: '0' and '1'. Therefore, a qubit is a quantum superposition of '0' and '1'. During the calculations, a qubit value may cover the entire spectrum of such superpositions, according to the law of probability. Operations on qubits are represented by quantum gates that conceptually correspond to logic gates commonly used in almost every classical hardware. This refers to the most popular quantum model of how information is processed. Quantum gates are basic operations performed on the available qubits to implement a quantum algorithm, which can be indirectly compared to bit operations performed by classical logic gates. Dirac notation, also called bra-ket notation, is used to simply denote quantum states and to distinguish them more easily from classical states. The notation uses angle brackets and a vertical bar to construct appropriate symbols. This makes it easier to describe the processes of quantum mechanics, in particular operations performed by quantum gates.

A ket is of the form $|\Psi\rangle$ and mathematically it denotes a vector Ψ in a complex vector space. Physically, it represents a state of some qubit. Using Dirac notation, the quantum states corresponding to classical '0' and classical '1' can be expressed as $|0\rangle$ and $|1\rangle$. Mathematically they represent the following vectors (1):

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

Any qubit, which is an arbitrary linear combination of base states: $|0\rangle$ and $|1\rangle$, is described as a superposition (2):

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2}$$

where α and β are complex numbers such that $\alpha^2 + \beta^2 = 1$. Quantum gates, on the other hand, can be represented by matrices. An example of a simple quantum gate is a Pauli-X gate (X), which has the property of changing the state $|0\rangle$ to the state $|1\rangle$ and vice versa. By applying a quantum gate on a qubit, the appropriate operation is performed. In the following cases, gate X negates the quantum state (3):

$$X|0\rangle = |1\rangle \qquad X|1\rangle = |0\rangle \tag{3}$$

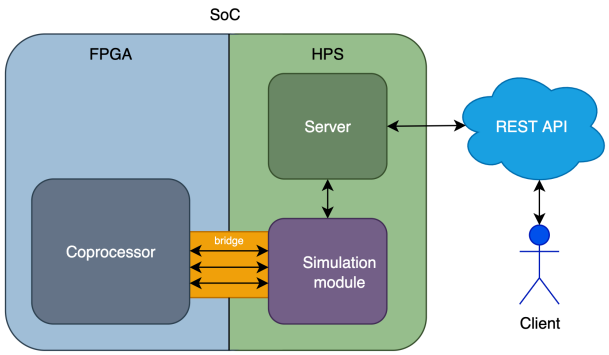

Figure 4. System architecture

Pauli-X gate expressed in matrix representation is shown below (4):

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{4}$$

Quantum gates are used to build quantum systems that process qubits in superposition states. The state of a multi-qubit system is expressed by a tensor product while applying a quantum gate to a qubit implies a matrix multiplication by a vector. The final step in the processing of a quantum algorithm is always the measurement of specific qubits, and each measurement of a qubit causes its wave function to collapse and bring its state down to one of the base states. Systems composed of multiple parallel quantum gates, when simulated on a classical computer, are calculated sequentially, gate by gate, operation by operation. Conversely, a quantum machine calculates the state of such a circuit in an instant. Contrary to a classic processor, a properly configured FPGA in many cases can significantly facilitate the efficient emulation of this quantum behaviour. An exemplary system created on the basis of the proposed concept is a simple web service implemented on the SoC FPGA platform and intended for learning the working principles of basic quantum gates. A user can create a simple quantum circuit through a web browser, and then observe the states of all qubits at many points in the circuit in real time. A dedicated mode allows part of the calculations to be performed on the FPGA.

The main components of the system are: client application, server and quantum simulation module on HPS and the coprocessor responsible for hardware emulation of quantum operations on FPGA. This system architecture is presented in Figure 4. The client sends requests to the server and waits for the results in responses. The Server communicates with the FPGA through the Simulation Module to send part of the most intensive computations to it. Then, the work is parallelized and computed asynchronously inside FPGA programmable logic circuits.

The Simulation Module manages this communication using the Bridge between the HPS processor and the FPGA logic matrix. Additionally, a user communicates with the System via the Internet using the REST API standard.

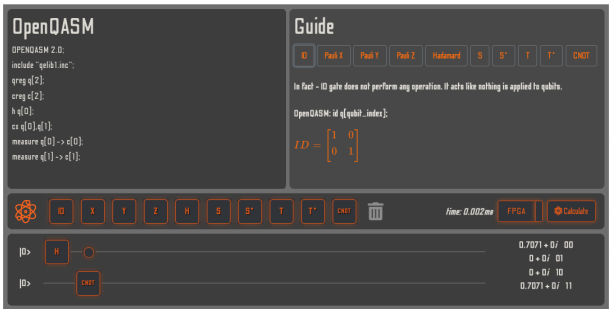

Figure 5. Client application's interface

1. Client

The client application is a website created with the React framework [3] that connects to the server via the REST protocol. Its task is to enable the graphical creation of quantum circuits and communication with the server. Figure 5 demonstrates the interface with a custom quantum circuit elaborated.

The user can compose circuits by dragging quantum gates: ID, Pauli-X, Pauli-Y, Pauli-Z, Hadamard, S, S*, T, T* and CNOT. The interface additionally provides descriptions and matrix representations of all available gates.

The proposed system is capable of viewing the results of the calculations performed after each operation by a specific quantum gate. These results are not a measurement of the system, but a raw representation of the results of gate operations on the state of the qubits. This approach has been adopted to facilitate the understanding and debugging of the created quantum circuits. In the presented example, which is only a simple proof of the concept described in the article, a typical quantum measurement operation has not been implemented but could be added if needed. The implementation of the qubits measurement operation could use a random number generator as an approximation of the quantum probability, from whose results and also based on the current quantum state of the measured qubits, the final classical state of the qubits would be determined.

The quantum circuit wizard presented here is exemplary, so the number of available quantum gates has been limited to the most basic ones. However, extending the system with further gates also would not be a demanding task, as the implementation method interprets the gate types in a generic way, based on their matrix representations. Adding more gates would only involve defining the appropriate matrices and preparing them for handling by the Client.

2. Server

The server module is a Java with Spring framework [4] application. It executes incoming requests from the client. After receiving quantum gates circuit information in the request, the data is appropriately converted and sent to the Simulation Module located in the same HPS part of the chip. Then, after receiving the results, the server sends them back to the client. The Server is responsible for starting and controlling the operation of the Simulation Module. From the Server level, passing input data to the system is possible through the use of the Simulation Module program arguments, while reading the output data is handled through standard output.

3. Simulation Module

The simulation module organises calculations of processes taking place in the simulated quantum circuit. After the appropriate transformation of the quantum gate matrix, it orders calculations to the Coprocessor, i.e. logical circuit located in the FPGA part of the chip. The results are computed column by column in the circuit. After receiving the results of these calculations, the Simulation Module reconnects them with the rest of the data and sends the simulation effects to the Server.

4. Bridge

The Bridge enables the communication between the Simulation Module and the FPGA coprocessor. It uses a lightweight interface based on the AXI (Advanced eXtensible Interface) standard. The data is sent back (read) from FPGA using the F2S interface and memory mapping. The data is immediately transferred to the FPGA via AXI and recorded in the appropriate local registers.

5. Coprocessor

The simulation process of quantum computations takes place partially on a programmable FPGA matrix, while the Simulation Module located on the HPS is responsible for the control of the entire process. The simulation inside the FPGA is based on performing basic arithmetic operations over complex numbers, in particular: addition, subtraction and multiplication. This part of the simulator is developed in the VHDL hardware description language. Note that the approach to the problem and hardware implementation of the logic on FPGA is very simple. The primary purpose of this example is to show the idea of hardware-aided simulation using the SoC-FPGA platform.

In quantum computing, all numbers are represented in the field of complex numbers. Our solution is based on simple observation and minimizes the number of required resources to represent complex numbers and allows to keep floating point precision. Most of the basic quantum gates available in the system consist of 0s and 1s. Hadamard gate, for instance, has an additional value: $\frac{1}{\sqrt{2}}$. However, it can be still broken down into absolute values and a matrix that consists of only 0s and 1s (5).

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

All values such as $\frac{1}{\sqrt{2}}$, in that case, can be stored directly on HPS; whereas, matrices can be sent to FPGA. To represent values from the set -1, 0, 1 we need only 2 bits and we do not need to implement floating-point representation on FPGA. In the first step, we need to build an initial qubit state. Next, we calculate the final state row-wise and iterate column by column. As a simple example, let's consider the circuit from Figure 5.

$$|\psi_1\psi_2\rangle = CNOT((H \otimes ID)|00\rangle) \tag{6}$$

where ID and CNOT gates are defined as follows:

$$ID = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

$$CNOT = cX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{8}$$

In the above equation from an example (6), all the elements apart from the Hadamard gate consist of 0s and 1s. To ensure that all matrices consist of only 0s and 1s, temporary Hadamard gate H' is introduced (7).

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} H' \tag{9}$$

Next, the equation can be transformed to the final form (10) that consists of vectors and matrices with only 0s and 1s (11).

$$|\psi_1\psi_2\rangle = \frac{1}{\sqrt{2}} CNOT((H' \otimes ID)|00\rangle) \tag{10}$$

$$|\psi_1\psi_2\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left(\left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) \tag{11}$$

From equation (11) all matrices are sent to FPGA. Required matrix multiplications and transformations can be performed simultaneously thanks to the use of FPGA. The final result is the FPGA output (vector) multiplied by the absolute value $\frac{1}{\sqrt{2}}$ stored on HPS (12).

$$|\psi_1\psi_2\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{12}$$

The concept of restoring appropriate values on the HPS is the basis of the implementation of the hardware coprocessor. The main disadvantage of the elaborated solution is the additional load on the processor that must recreate the result. Nevertheless, in order to transfer floating point numbers, the processor would have to convert and then recreate the appropriate form as well.

By connecting two different platforms: FPGA and HPS, it was possible to combine a high-level GUI designed in the React framework, through a Java Spring-based application, with a low-level FPGA-based parallel coprocessor. An important advantage of the demonstrated system architecture and computing concept is the ease of integration of high-level technologies and solutions commonly used commercially with their own hardware accelerator, which has now become possible thanks to the use of the SoC FPGA platform.

6. Conclusion

The hardware-aided parallel computing concept based on the SoC-FPGA could have many more applications than just the quantum simulation shown here. The demonstrated architecture of a hardware platform equipped with a programmable FPGA matrix is a versatile technical solution that has a good chance of even greater expansion in the future, especially in the field of processor development and cloud infrastructure. The flexibility of FPGA programming facilitates the prototyping of a wide range of new technological solutions. The implementation of an extensive system with many modules on a small platform is a big technical challenge. A solution like the one shown in this research, available to everyone, shows that with creativity, many limitations can be omitted and the entire system can benefit from increased computing power while still keeping lower energy consumption. This is a crucial aspect for R&D projects at the initial step of advancement when the costs of prototyping should be adjusted to the risk of the project. Definitely, more complex algorithms such as Wavelet or Fourier transformations for fast signals analysis or others can benefit from the hybrid implementations on SoC-FPGA platforms.

AUTHORS

Jacek Długopolski* – Faculty of Computer Science, AGH University of Krakow, al. A. Mickiewicza 30, 30-059 Kraków, Poland, e-mail: dlugopol@agh.edu.pl.

Jakub Czerski – European Organization for Nuclear Research CERN, Espl. des Particules 1, 1211 Meyrin, Switzerland, e-mail: jakub.marek.czerski@cern.ch.

Mateusz Knapik – CyberOwl Ltd, No 1 Colmore Square, Birmingham, United Kingdom, B4 6AA, e-mail: mateusz.knapik@cyberowl.io.

*Corresponding author

ACKNOWLEDGEMENTS

The research presented in this paper was partially supported by the funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education.

References

[1] "An 796: Cyclone® v and arria® v soc device design guidelines".

[2] "Ibm quantum composer". https://quantum-computing.ibm.com/composer/files/new. Accessed: 27/6/2022.

[3] "React library", https://reactjs.org.

[4] "Spring framework", https://spring.io.

[5] "Atlas-soc kit - user manual", 2015.

[6] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, 1–13. doi: 10.1109/MICRO.2016.7783710.

[7] M. C. Herbordt, Y. Gu, T. VanCourt, J. Model, B. Sukhwani, and M. Chiu, "Computing models for fpga-based accelerators", *Computing in Science Engineering*, vol. 10, no. 6, 2008, 35–45. doi: 10.1109/MCSE.2008.143.

[8] S. Hoover, "Hardware accelerated web applications using cloud fpgas", 2018.

[9] Z. Jiang, N. Audsley, D. Shill, K. Yang, N. Fisher, and Z. Dong, "Brief industry paper: Axi-interconnectrt: Towards a real-time axi-interconnect for system-on- chips". In: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, 437–440. doi: 10.1109/RTAS52030.2021.00046.

[10] M. Levental, "Tensor networks for simulating quantum circuits on fpgas", 2021.

[11] X. Li, C. Fei, and D. Maskell, "Fpga overlays: Hardware-based computing for the masses", 2018. doi: 10.15224/978-1-63248-144-3-12.

[12] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks", *Neural Computing and Applications*, vol. 32, 2020. doi: 10.1007/s00521-018-3761-1.

[13] R. Skhiri, V. Fresse, J. Jamont, B. Suffran, and J. Malek, "From fpga to support cloud to cloud of fpga: State of the art", *International Journal of Reconfigurable Computing*, vol. 2019, 2019, 1–17. doi: 10.1155/2019/8085461.

[14] A. Wicaksana, "Portable infrastructure for heterogeneous reconfigurable devices in a cloud-fpga environment", 2018.