Received June 30, 2021; revised December 17, 2021; accepted January 10, 2022; date of publication January 27, 2022;
date of current version February 15, 2022.

Digital Object Identifier 10.1109/TQE.2022.3143997

Versatile and Concurrent FPGA-Based Architecture for Practical Quantum Communication Systems

ANDREA STANCO[1], FRANCESCO B. L. SANTAGIUSTINA[1,2], LUCA CALDERARO[1,3], MARCO AVESANI[1], TOMMASO BERTAPELLE[1], DANIELE DEQUAL[4], GIUSEPPE VALLONE[1,5], AND PAOLO VILLORESI[1]

[1]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Padova, 35131 Padova, Italy
[2]Dipartimento di Matematica "Tullio Levi-Civita,", Università degli Studi di Padova, 35121 Padova, Italy
[3]ThinkQuantum S.r.l., 36030 Sarcedo, Italy
[4]Unità Telecomunicazioni e Navigazione, Agenzia Spaziale Italiana, 75100 Matera, Italy
[5]Dipartimento di Fisica e Astronomia, Università degli Studi di Padova, 35131 Padova, Italy

Corresponding author: Andrea Stanco (e-mail: andrea.stanco@unipd.it).

This work was supported in part by the Italian Ministry of Education, University and Research under the initiative "Departments of Excellence" (Law 232/2016).

ABSTRACT This article presents a hardware and software architecture, which can be used in those systems that implement practical quantum key distribution (QKD) and quantum random-number generation (QRNG) schemes. This architecture fully exploits the capability of a System on a Chip (SoC), which comprehends both a field-programmable gate array (FPGA) and a dual-core CPU unit. By assigning the time-related tasks to the FPGA and the management to the CPU, we built a flexible system with optimized resource sharing on a commercial off-the-shelf (COTS) evaluation board, which includes an SoC. Furthermore, by changing the dataflow direction, the versatile system architecture can be exploited as a QKD transmitter, QKD receiver, and QRNG control-acquiring unit. Finally, we exploited the dual-core functionality and realized a concurrent stream device to implement a practical QKD transmitter, where one core continuously receives fresh data at a sustained rate from an external QRNG source, while the other operates with the FPGA to drive the qubit transmission to the QKD receiver. The system was successfully tested on a long-term run proving its stability and security. This demonstration paves the way toward a more secure QKD implementation, with fully unconditional security as the QKD states are entirely generated by a true random process and not by deterministic expansion algorithms. Eventually, this enables the realization of a standalone quantum transmitter, including both the random numbers and the qubit generation.

INDEX TERMS Commercial off-the-shelf (COTS), embedded system, field-programmable gate array (FPGA), quantum communication (QC), quantum key distribution (QKD), quantum random number generator (QRNG), System on a Chip (SoC).

I. INTRODUCTION

Quantum communication (QC) is one of the promising applications of quantum technology and has recently received a relevant boost toward commercial applications. Quantum key distribution (QKD) and quantum random-number generation (QRNG) are the two leading technologies of QC since their combination allows us to realize the perfect secrecy protocol, resistant to any external attack. The realization of such a system requires the design and development of several components: from the optical setup to the driving electronics, from the digital control board to the management software. Besides the particular quantum implementations, which can vary according to different security protocols, an essential component of the whole setup is a supervision board capable to provide a deterministic behavior, high temporal resolution, and high-speed computation. Within this framework, a field-programmable gate array (FPGA) is almost a mandatory choice for such applications [1], [2]. The FPGA also offers an advantage in terms of power consumption [3], which can be a key feature for critical applications, such as CubeSat missions for satellite QC [4]. However, the literature on the usage of an FPGA for QC is mostly focused on the QKD postprocessing hardware acceleration [5]–[8] and lacks detailed presentations of control system architectures. Moreover, the exploitation of both FPGA and CPU capabilities was proposed only in [9], which, however, investigated encryption and network functionalities. In this article, we present a generalized FPGA-based architecture for control and readout functions of QC systems. It exploits the integrated CPU counterpart, forming a System on a Chip (SoC), to improve flexibility and unleash continuous operability of a QKD scheme fed, without any expansion, by an external QRNG. The architecture can be easily applied to discrete-variable QKD (DV-QKD) and discrete-variable QRNG (DV-QRNG) applications, implementing the control functionalities of those systems. Moreover, the schematic has the potentiality to be used even in continuous-variable QKD (CV-QKD) and continuous-variable QRNG (CV-QRNG), provided that an auxiliary digital-to-analog converter (DAC) and analog-to-digital converter (ADC) should be included. The architecture was implemented and tested on an entry-level evaluation board: ZedBoard by Avnet, which mounts the Zynq-7020 SoC. Besides representing an optimal tradeoff between costs and performances, the FPGA chip (Artix-7) showed good results in radiation environments [10], and it can be considered a good choice for both terrestrial and satellite QC applications. Moreover, the ZedBoard was also successfully used in a recent quantum computer scheme [11], demonstrating excellent reliability. The exploitation of both the SoC's layers (FPGA and CPU[1]) leads to a high level of flexibility, allowing to scale the application functionalities to the specific part of the chip. According to the specific application, the system can be set in a top-down (dataflow from the PC/user to the quantum system) or bottom-up (dataflow from the quantum system to the PC/user) configuration. The system has the flexibility to be used with several protocol configurations and successfully contributed to different experiments over the past few years [12]–[21]. Recently, it has also been tested in the prototype of a QKD transmitter for CubeSat mission [22], making it suitable for satellite QCs, a key area of QC. Given the presence of a dual-core CPU, we also designed, developed, and successfully tested a dual-core application capable to sustain a continuous data transfer from an external source to the CPUs and then to the FPGA. This feature is the key to implement a provably secure QKD system since it combines a QRNG output stream with a QKD stream without the need of a random number expansion (which allowed very high transmission rates at the price of undermined security [7], [8]), paving the way to commercial QKD devices with full unconditional security.

The rest of this article is structured as follows. In Section II, an overall overview of the architecture is presented. In Sections III and IV, the FPGA and CPU layers are described. In Section V, the dual-core architecture for a QKD transmitter is presented along with the system test results.

[1]Note that from now on, the term “CPU” always refers to the CPU counterpart of the SoC.

II. ARCHITECTURE OVERVIEW

Current FPGA-based QKD systems do not exploit the capability of an SoC architecture and implement a single-layer architecture on the FPGA chip to execute every operation, even the ones that can be performed by a CPU processor and are not time intensive [5], [7]. While this approach may lead to a higher performance system, realizing the entire design on an FPGA can reduce flexibility as even trivial changes require a proper hardware review, increasing the complexity of the design workflow. On the contrary, our system architecture is organized in two different layers thanks to the SoC capability. The lower layer is the FPGA one, where all the deterministic and high-resolution operations are carried out. The higher layer is the CPU(s) one, which is responsible for the parameters and data management operations as well as the communication with the outside world. Besides the functions' separation, this subdivision is also a key point for the maintenance and the upgrading of the architecture since the two layers require different programming languages (VHDL/Verilog versus C/C++) and different design teams. Two additional layers, i.e., end user/external source and quantum system, enclose the previous ones completing the whole practical system. The architecture is designed to control a given quantum apparatus and can be easily switched between top-down and bottom-up workflows, which represents the distinction between QKD transmitter and QKD receiver/QRNG applications. For QKD applications, the current version of the architecture does not implement the postprocessing and temporal synchronization between the transmitter and the receiver. As a matter of fact, an FPGA-based postprocessing can be considered effective only if implemented both on Alice and Bob as accelerating the processing on just one side may not bring to practical benefits due to possible bottlenecks on the other end. Nevertheless, on DV-QKD, a time-to-digital converter (TDC) is required to distinguish the time of arrival of the incoming single photon. Thus, an efficient FPGA implementation of a DV-QKD receiver should include both the TDC and the postprocessing functionalities. This approach can result in significant FPGA design efforts without bringing valuable benefits as a standard commercial personal computer (PC) may be sufficient to keep up with the postprocessing rate, like in our implementations. Furthermore, in an R&D/Academic scenario, having a PC-based postprocessing can be an advantage as it allows a more straightforward realization and upgrade of the software and, thus, an easier adaptation to a given QKD test, where the goal is to investigate different protocols and performances. Therefore, we chose to implement a PC-based postprocessing as it can be easily adapted to different experiments. Nevertheless, while being more targeted to commercial deployment, where higher performances should be preferred over flexibility, an FPGA-based postprocessing can be part of future integration and investigation. On the same PC, we also implement the Qubit4Sync, a recent postprocessing synchronization method for the temporal alignment of the two parties [17] making unnecessary any dedicated module

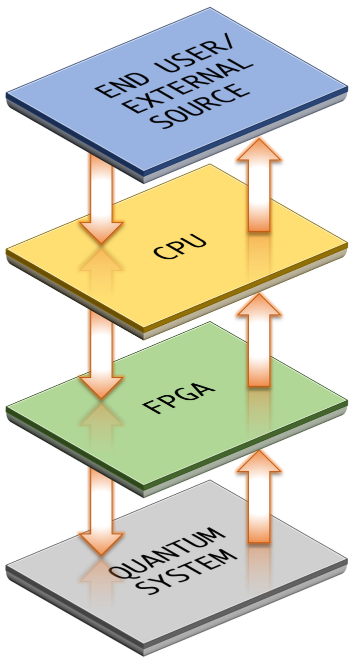

FIGURE 1. **Overview of the four-layer structure of the system. The embedded architecture is divided into two different layers (FPGA+CPU), which are enclosed by the two outside-world layers (end user/external source and quantum system).**

on the FPGA. Nevertheless, if required, standard synchronization solutions, such as direct clock transmission, can be added straightfowardly to our architecture. An overview of the whole schematic is given in Fig. 1. In the following, we provide a general description of the top-down and bottom-up applications.

A. TOP-DOWN APPLICATION

In this configuration, the dataflow starts from an external device, e.g., a QRNG or PC, goes through the CPU, then to the FPGA, and finally to the chip input–output (I/O) pins. This layout is suitable for a QKD transmitter, as the raw cryptographic key, either generated in real time by a QRNG or previously stored in a PC, is fed through the CPU to the FPGA, which drives the hardware dedicated to quantum state generator accordingly. As detailed in the following, the first communication step, from the external device to the CPU, has been performed over Gigabit Ethernet. This choice provides both a high throughput of the data transfer ($>$600 Mbit/s) and a great flexibility, being Ethernet a widespread standard. As there is no encryption of the dataflow, it is of paramount importance to protect the communication channel from eavesdropping. This can be done by setting up a private local area network (LAN), physically disconnected from other networks, between the SoC and the external device. The second layer in the stack is from the CPU to the FPGA. For this step, two solutions have been implemented. The configuration parameters, e.g., qubit frequency or total transmission length, which have a very slow refresh rate, are exchanged with a direct communication via the Advanced eXtensible Interface (AXI) protocol. Instead, for the raw key exchange, which can reach 400 Mb/s of steady data transmission, we exploited both the onboard DDR-RAM, accessible from the CPU, and the block RAM (BRAM), integrated in the FPGA but accessible from the CPU through the AXI protocol. The BRAM, with a maximum length in the order of Mbits, has been divided in two halves, so that while the FPGA is reading from one section, the CPU can update the contents of the other with the data stored in the RAM and previously received via Ethernet. This allows for a continuous and synchronized dataflow from the external device to the FPGA and, hence, its I/Os. According to the specific quantum system, the output signals are routed to either Peripheral Module Interface (LVCMOS33 standard) or FPGA Mezzanine Card ports (LVCMOS18 standard) of the ZedBoard and then properly amplified by an external driving stage.

B. BOTTOM-UP APPLICATION

In this configuration, the dataflow starts from the chip I/Os controlled by the FPGA and is then transmitted from the FPGA to the CPU and finally from the CPU to the external device. This layout can be used either for a QKD receiver or for a QRNG, where the electrical signal coming from external devices, i.e., single-photon detectors, is sampled by the FPGA I/Os. The sampled and stored signal is then transferred, via the CPU, to an external computer, for the implementation of the postprocessing phases of the QKD protocol, i.e., parameter estimation, error correction, and privacy amplification. The communication interfaces for the bottom-up configuration are the same as for the top-down. It must be mentioned that also in this case, the communication between the CPU and the external computer must be performed over a secure LAN, as the data stream at this level is not encrypted. To guarantee a high level of security, the LAN used for the PC–CPU communication has to be reserved and, therefore, physically separated from the one used for communication between the transmitter and receiver PCs.

III. FPGA LAYER

The FPGA implementation allows us to have a perfect time control over the optical system. Indeed, the capability to schedule every operation according to a system clock is a key feature for the realization of a QKD/QRNG system. When used in a QKD transmitter configuration, the FPGA is responsible for the rightful generation of the electrical pulses, which drive the electrooptical elements of the setup. When used as a QRNG/QKD receiver, the FPGA takes care of the read-out operations of the external electrical signals coming from single-photon detectors. For the sake of completeness, it is also possible to consider the application of the architecture for CV-QKD [23] and CV-QRNG [24], [25]. The difference would be that the FPGA needs to interface with proper external DAC (for the CV-QKD transmitter) and external ADC (for the CV-QKD receiver and CV-QRNG).

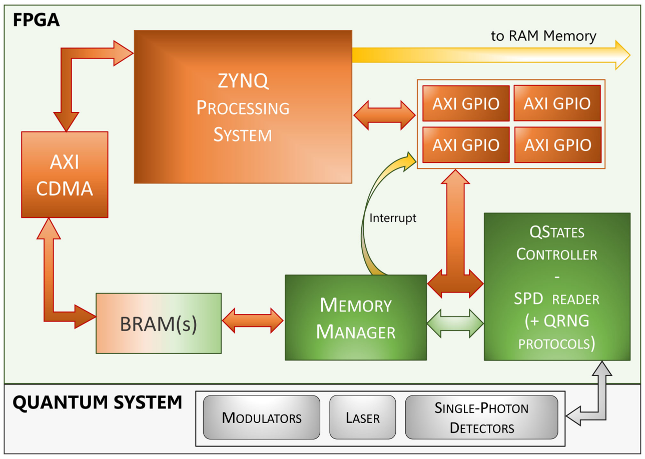

FIGURE 2. Schematic view of the FPGA system. The modules in green are custom VHDL blocks, which control the memory management, the qubit generation, and the single-photon detector readout. The orange modules are the AXI-based ones, and they are AXI-CDMA and AXI-GPIO. The BRAM module has a mixed color since it is controlled both by AXI and by custom VHDL modules. The CPU part is initialized in the FPGA design and is identified by the ZYNQ Processing System module, which can also access to RAM. The versatile schematic is the same over different applications. The only modification concerns the output interfacing module: QStates Controller triggers the qubit emission in a QKD transmitter configuration; SPD Reader implements the sampling function of the external signal from single-photon detectors in a QRNG/QKD receiver configuration. In the case of QRNG configuration, the implementation of random generation protocols is also included.

The general and simplified structure of the FPGA design is shown in Fig. 2. The schematic shows the versatility of the architecture as it can be exploited in different QC applications without changing anything but the output interfacing module (QStates Controller in the QKD transmitter configuration; SPD Reader and QRNG protocols in the QRNG/QKD receiver one) and the data flow direction. The design uses AXI-capable blocks for communication and data transfer to (from) the CPU along with BRAMs and custom VHDL blocks for FPGA data management, Memory Manager block, and external signal generation (readout), QStates Controller, and SPD Reader blocks. The QRNG application also includes dedicated modules implementing random generation protocols [26], [27]. AXI-GPIOs allow us to set parameters from the CPU and to read out interrupt signals asserted from custom VHDL blocks. AXI-CDMA enables the possibility to move data to (from) the board RAM from (to) the BRAM(s). The Memory Manager is responsible for managing the data transfer between the BRAM(s) and the other custom blocks. Being the BRAM divided into two halves, the Memory Manager asserts a signal every time it reaches the end of one of the two halves (while reading or writing) and, in turn, the signal is read by an AXI-GPIO and interpreted as an interrupt from the CPU, which writes (reads) new data to (from) the BRAM. The system clock was set in a range between 100 and 200 MHz depending on the specific application. Future improvements will consider pushing further this frequency in order to increase the overall system speed. Indeed, apart from tighter timing constraints, a higher frequency clock implies a higher data throughput to/from the external device, which may exceed the gigabit range of the board.

A. QKD TRANSMITTER

The previously described general architecture is meant to be protocol blind and can be exploited to set up a QKD-transmitter controller regardless of the exact QKD protocol and qubit encoding. The only module that is protocol dependent and that requires a specific design is the one responsible for encoding the raw key data into electrical output pulses. This module is the QStates Controller, which, according to what is usually done in literature (see, for instance, the quantum state selection module of [8]), operates the qubit emission by encoding the raw key data into electrical output pulses, which drive the laser and electrooptical modulators. We designed several variations of this module according to the chosen QKD protocol [28]–[31], derived from the well-known BB84 [32], and implementation [16], [18], [20], [22]. Here, we give a brief description of one of the most recent versions, presented in [22]. With a system clock set to 200 MHz, a pulse of 5 ns can be provided at the output. The encoding of every qubit requires, nominally, no more than 15 ns time slot since the polarization encoding describes three different polarization states and necessitates an output pulse in three different temporal positions (0–5 ns, 5–10 ns, and 10–15 ns). The decoy implementation works in a similar way, describing three different intensity levels [33], [34], but requests an output pulse in only two temporal position (0–5 ns and 5–10 ns) in combination with a possible laser switch OFF. The pulse for the laser driver is sent at the begin of every slot (0–5 ns) unless the case of a specific decoy state. To compensate and synchronize the output signals with the optical path length, a dedicated time offset can also be applied to each signal. While similar temporal diagrams, which belong to specific QKD protocols and qubit encoding, are well known in the literature [35], the advantage of our solution is that it requires only two voltage levels, which are provided by the FPGA itself (and a proper amplification stage) avoiding the usage of a DAC or multilevel devices.

Two different BRAMs were instantiated: one for the polarization encoding data and the other for the decoy one. Since every qubit requires 2 bits to distinguish among three polarization states as well as other 2 bits to discriminate among three decoy states, the BRAMs were set to the same size and operated with the same interrupt routine. For the sake of clearness, it is possible to optimize the overall qubit encoding using just three bits for the polarization+decoy encoding. Nevertheless, we chose to use two+two encoding for mainly two reasons. The first reason is that a three-bit encoding would have required a quite complicated, and in a certain way inefficient, routine to distinguish the data within each byte as more than two but less than three qubits encoding data would have been stored in one byte. The second reason is that a two+two bit encoding allows us to separate the paths, the memories, and, in turn, the transmission control protocol (TCP) sockets of the polarization and decoy data enhancing the robustness and flexibility of the overall system.

B. QKD RECEIVER/QRNG

In principle, the FPGA schematic for a QKD receiver is similar to the one for a DV-QRNG. In both cases, the I/Os are connected to the output signals of single-photon detectors, and the FPGA implements the sampling process to produce a bitstring containing the digital temporal description of the single-photon events. First of all, the input signal is translated to the FPGA clock domain by using a proper async-to-sync hardware module included in the SPD Reader. Then, in the case of a QRNG application, the sample bits are temporarily accumulated and later processed by proper modules, which apply random generation protocols to a small set of data, as described in [14]. The random bit is then stored and managed by the Memory Manager, which, in turn, transfers a 32-bit array to the ith address of a BRAM and calls an interrupt whenever it reaches half or the end of it. Moreover, this architecture was also the perfect option to implement a synchronized QRNG, which was needed for the realization of [21]. This particular application required to output a random number generated only after a specific trigger event. Therefore, the QRNG had to be synchronized with the experiment apparatus, which required a modification of the architecture to allow a resetting of all the random data, including the SPD samples, at a specific time instant. The resetting was triggered by an external electrical signal coming from the experiment setup. The random bit was then used to produce an auxiliary output port, which set a specific optical component of the experiment. To improve the randomness of the output number, the architecture was also doubled and produced two random bits, which were XORed. For further details, refer to [21]. As a matter of fact, by removing the generation protocol modules, the architecture becomes suitable to be used as a QKD receiver. However, a drawback of this implementation is the low time resolution provided by the system clock, which, even in a high-range FPGA-chip case scenario, does not exceed 1 GHz. For a high-performance QKD, a subnanosecond time resolution at the receiver is required. Therefore, this implementation can be a solution only for low-cost QKD systems. Nevertheless, the design and integration of a TDC FPGA module (such as [36]–[38]) or the exploitation of an external integrated circuit (such as [39]) would allow for a subnanosecond time resolution and, thus, the use in high-performance QKD systems. Indeed, future steps will investigate such solutions.

IV. CPU LAYER

The CPU software is implemented as a standalone/bare-metal application, and no operative system is required. This has the great advantage of having a very light and fast software at a higher design cost. The software, developed in C and C++ languages, has the role to interface the FPGA layer with the external source and the final user. It mainly implements the interrupt routine to move (read) data from (to) the RAM to (from) the BRAM anytime the Memory Manager reaches the half of the end of a BRAM. It also implements the TCP connection sockets to receive commands and data from the external source or user.

A. TCP CONNECTION

To communicate with the outside world, the TCP protocol was chosen. Given the robustness of this protocol over any possibility of losing data packets, this choice has to be preferred over the user datagram protocol (UDP), which cannot guarantee a reliable communication to the application layer, thus undermining the validity of the QKD implementation. Moreover, TCP can meet the requirement on data synchronization between parties thanks to its acknowledgment and hand-shaking structure. For a QRNG application, where the data is output to an external receiver, a UDP protocol might be suitable in any case since any (negligible) data losses do not affect the overall quality and performance of the QRNG device. Nevertheless, for QKD postprocessing purposes, one can consider to send the random stream to two different devices, e.g., a QKD transmitter and a computer, and in this case, a data loss would jeopardize the whole system. Moreover, changing the protocol only for the QRNG application would reduce the overall system flexibility.

The connection between the PC and the CPU applications is structured in two or more different TCP server–client sockets: one is for commands and parameters exchange, while the other(s) is for data exchange. Indeed, the data socket has the only purpose to receive new data from the external source. Thus, the required operations and statement conditions in the *data received callback* are quite few allowing to have optimal performance over the TCP bandwidth (>600 Mbit/s).

V. QKD-TRANSMITTER CONTINUOUS STREAM IMPLEMENTATION

A fundamental feature of any future commercial QC device is the capability to continuously sustain the transmission of fresh random data. That is, the external randomness source, such as a QRNG device or a computer where random keys are stored, must provide new data with a sufficient bitrate, and the QKD transmitter must be able to perform at the same time both the reading/storing of these data and the transmission of electrical pulses. Nowadays, specific workarounds allow us to avoid the implementation of such functionality [7] but necessarily reduce the overall security of the system. For instance, one can connect a low-bitrate QRNG to the QKD source, where the random data are expanded to reach the required bitrate [8]. The expansion process, although implemented via standard cryptographic primitives, does not offer an unconditional type of security and can represent a security breach in the entire QKD system. Hence, we developed a dual-core architecture able to sustain the required data rate for a secure QKD implementation, allowing a random data stream generated entirely by a QRNG. This approach has

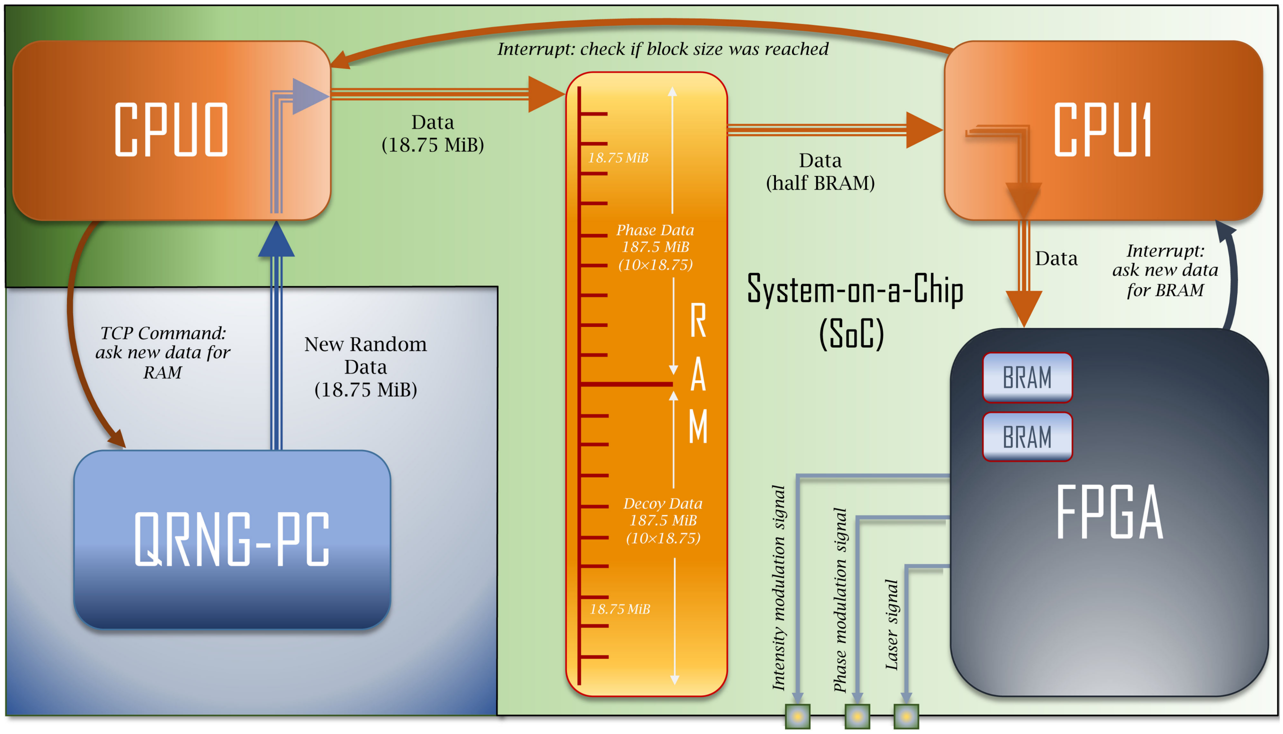
Interrupt: check if block size was reached
CPU0
Data (18.75 MiB)
CPU1
Data (half BRAM)
18.75 MiB
Phase Data 187.5 MiB (10×18.75)
RAM
Decoy Data 187.5 MiB (10×18.75)
18.75 MiB
System-on-a-Chip (SoC)
Data
Interrupt: ask new data for BRAM
TCP Command: ask new data for RAM
New Random Data (18.75 MiB)
QRNG-PC
BRAM
BRAM
FPGA
Intensity modulation signal
Phase modulation signal
Laser signal

FIGURE 3. Schematic view of the dual-core system, representing the flow of data from the QRNG-PC to the FPGA. The request of new data is triggered by the FPGA, each time it reads half of the BRAM, by means of an interrupt to CPU1, which then moves the data from the buffer to the RAM. Each time CPU1 reads 18.75 MiB of data from the buffer it sends an interrupt to CPU0 to update the new block with fresh data from the QRNG-PC.

the advantage of being unconditionally secure. Moreover, keeping the random data stored on the PC eases the QKD postprocessing procedure or, alternatively, lowers the memory resources of the SoC needed to store the transmitted bitstring until the receiver communicates the detected qubits. Furthermore, the bias of the random bits, required by some efficient QKD protocols [30], can be adjusted without the need of programming or setting the FPGA, allowing to optimize it according to the current quantum channel condition.

The data streamflow is represented in Fig. 3. The data generated from the QRNG-PC are received by a TCP socket and then moved by CPU0 into a buffer in the RAM. Meanwhile, CPU1 reads the data from the RAM and moves it to the BRAM, which can be read by the FPGA. The buffer size is set to 187.5 MiB and divided into ten blocks, which are written atomically by CPU0. Hence, when a block has been moved to the FPGA, an interrupt from CPU1 is sent to CPU0 to notify that a new block of the buffer can be written. CPU0 forwards a request of a new block of 18.75 MiB to the QRNG-PC. CPU1 reads smaller chunks from the buffer, whose size is half of the BRAM's size, when it receives an interrupt from the FPGA. Compared to the BRAM size, the buffer is larger to avoid an unwanted stop of the continuous feed of data to the FPGA, which may happen due to the temporary loss of speed of the TCP channel, or the latency of CPU0. The whole architecture is doubled in order to manage both the stream for polarization and decoy data.

A. SYSTEM TESTS

We implemented this system on a QKD transmitter to test the top-down application in a continuous stream mode. We performed a double-stress test: the first one was with a real QRNG device, based on the scheme of [40] offering high security and bitrate, while the second test was carried out with a cryptographically secure pseudorandom number generator (CSPRNG) able to provide a sufficient data rate as well. We chose to perform the test also with a CSPRNG to show the system capability in a fallback scenario, where no QRNG device is available. The (pseudo)random data were stored in a buffer of the PC, to ease the retrieval of the quantum states sent to the receiver needed for the raw key sifting. Indeed, once the QKD transmitter has produced and sent the quantum states, the QKD receiver measures and detects a subset of these states due to unavoidable channel losses. The QKD receiver communicates to the transmitter the list of states it detected (without revealing the outcome of the measurement). Hence, the QKD transmitter selects the subset of random data that will be used for the sifting.

The stream of data at the PC is managed by three threads: one for the production of blocks of random sequences; one for the system managing the transfer, upon request, of the random data from the buffer to the board via TCP; and one for the selection of the states detected by the QKD receiver. As anticipated, the (pseudo)random data can either be received from a QRNG device or generated internally by a CSPRNG. In the former case, the first thread is used to receive (via

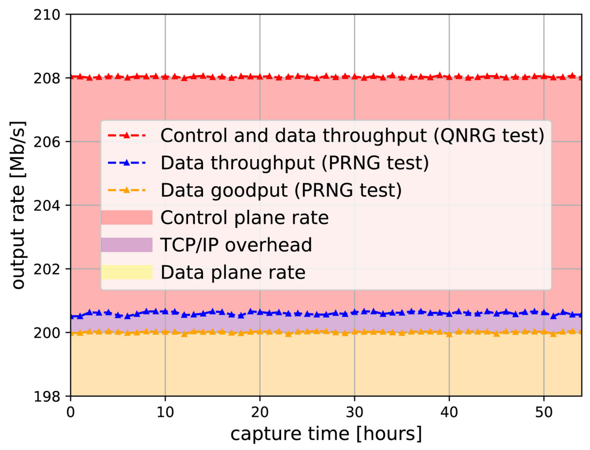

FIGURE 4. Plot of the TCP traffic from the PC to the Zynq-7020 for the transmission of the random sequences and control messages, for two 55-h-long tests using either the QRNG or CSPRNG as randomness source. We plot in red the aggregated traffic of the data and control plane for the QRNG test, amounting totally to about 208 Mb/s. The transmission of states and intensities sequences accounts for 200 Mb/s. This stream is represented in yellow with the data from the PRNG test. Finally, we can appreciate a small TCP/IP overhead (as seen from the PC OS) due to TCP segmentation offload.

UDP or TCP) the random bits from the QRNG and to bias them according to the desired Bernoulli distribution. In the latter case, the thread carries on the generation of the pseudorandom data by using a Chacha20-based CSPRNG, seeded with the Intel Secure Key hardware random number generator embedded in the recent generations of Intel CPUs. We synchronized the write and read operations on the buffer of these multiple threads by using semaphores. The buffer was divided in chunks that could be either written or read at a time. Hence, one semaphore is needed to allow the writing of new chunks of data by the RNG thread, which can be done only on those chunks that have been read by the thread selecting the subset of states arrived at the QKD receiver. Another semaphore is needed to ensure that the chunks of data sent to the system are those that have been rewritten by the RNG thread.

In our test, the QKD repetition rate was set to 50 MHz. Since the state encoding uses two bits for the state polarization and two bits for the mean photon number, we have two data streams of 100 Mb/s, plus a third stream for control communications. The outbound traffic was monitored from the transmitting PC during the two tests and is reported in Fig. 4. Given this steady input, the system was able to carry on all the needed operations seamlessly along all 55 h of the tests, resulting in a successful execution of the QKD protocol.

After the completion of the present manuscript, we became aware of a recent work presenting a similar scheme [41]. The latter implements the QRNG and QKD control hardware on a single FPGA chip. While having the advantage of being a more compact device without the needs of a continuous stream between parties, this solution is less flexible as it is not possible to combine a given QRNG with an arbitrary QKD device.

VI. CONCLUSION

In this article, we presented a versatile architecture based on FPGA technology exploiting also a CPU counterpart, forming an SoC, for the implementation of practical QC systems. The SoC architecture was developed in different layers with different tasks and was easily interchangeable among different QC applications, such as DV-QKD transmitter, DV-QKD receiver, and DV-QRNG. We also implemented and tested a dual-core functionality performing a TCP continuous stream between a QRNG source and a QKD transmitter without the need of data expansion to reach the amount of data required to encode every qubit. This allows us to strengthen the security of the QKD implementation, as the random settings needed by the QKD protocol are guaranteed to be unconditionally secure, unlike those generated by expansion algorithms. The system was implemented on a low-budget COTS evaluation board, and it was successfully tested to continuously provide four bits to encode a qubit every 20 ns. Future steps will consider higher frequency implementation as well as continuous-variable applications by including proper DAC and ADC hardware.

ACKNOWLEDGMENT

The authors Andrea Stanco and Daniele Dequal would like to thank Dr. S. Gaiarin for useful help and hints in the general architecture design. The authors Andrea Stanco and Luca Calderaro would like to thank F. Berra for useful discussions about dual-core architecture verification. The author Andrea Stanco would like to thank Dr. D. G. Marangon for useful help and discussion for the DV-QRNG architecture design.

REFERENCES

[1] D. Bacco, M. Canale, N. Laurenti, G. Vallone, and P. Villoresi, “Experimental quantum key distribution with finite-key security analysis for noisy channels,” *Nature Commun.*, vol. 4, no. 1, Sep. 2013, Art. no. 2363, doi: 10.1038/ncomms3363.

[2] K. Wei *et al.*, “High-speed measurement-device-independent quantum key distribution with integrated silicon photonics,” *Phys. Rev. X*, vol. 10, Aug. 2020, Art. no. 031030, doi: 10.1103/PhysRevX.10.031030.

[3] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, “Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels,” in *Proc. IEEE Int. Conf. Embedded Softw. Syst.*, 2019, pp. 1–8, doi: 10.1109/ICESS.2019.8782524.

[4] D. K. L. Oi *et al.*, “CubeSat quantum communications mission,” *EPJ Quantum Technol.*, vol. 4, no. 1, Apr. 2017, Art. no. 6, doi: 10.1140/epjqt/s40507-017-0060-1.

[5] H.-F. Zhang *et al.*, “A real-time QKD system based on FPGA,” *J. Lightw. Technol.*, vol. 30, no. 20, pp. 3226–3234, Oct. 2012, doi: 10.1109/JLT.2012.2217394.

[6] A. R. Dixon and H. Sato, “High speed and adaptable error correction for megabit/s rate quantum key distribution,” *Sci. Rep.*, vol. 4, no. 1, Dec. 2014, Art. no. 7275, doi: 10.1038/srep07275.

[7] N. Walenta *et al.*, “A fast and versatile quantum key distribution system with hardware key distillation and wavelength multiplexing,” *New J. Phys.*, vol. 16, no. 1, 2014, Art. no. 013047, doi: 10.1088/1367-2630/16/1/013047.

[8] J. Constantin *et al.*, “An FPGA-based 4 Mbps secret key distillation engine for quantum key distribution systems,” *J. Signal Process. Syst.*, vol. 86, no. 1, pp. 1–15, Jan. 2017, doi: 10.1007/s11265-015-1086-1.

[9] T. Lorunser *et al.*, "Security processor with quantum key distribution," in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Processors*, 2008, pp. 37–42, doi: 10.1109/ASAP.2008.4580151.

[10] B. LaMeres *et al.*, "Next on the Pad: RadSat—A radiation tolerant computer system," in *Proc. 31th Annu. AIAA/USU Conf. Small Satell.*, 2017, Art. no. SSC17-III-11. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2017/all2017/87/.

[11] I. Pogorelov *et al.*, "Compact ion-trap quantum computing demonstrator," *PRX Quantum*, vol. 2, Jun. 2021, Art. no. 020343, doi: 10.1103/PRXQuantum.2.020343.

[12] M. Avesani *et al.*, "Deployment-ready quantum key distribution over a classical network infrastructure in Padua," *J. Lightw. Technol.*, to be published, doi: 10.1109/JLT.2021.3130447.

[13] H. Tebyanian, M. Zahidy, M. Avesani, A. Stanco, P. Villoresi, and G. Vallone, "Semi-device independent randomness generation based on quantum state's indistinguishability," *Quantum Sci. Technol.*, vol. 6, 2021, Art. no. 045026, doi: 10.1088/2058-9565/ac2047.

[14] A. Stanco, D. G. Marangon, G. Vallone, S. Burri, E. Charbon, and P. Villoresi, "Efficient random number generation techniques for CMOS single-photon avalanche diode array exploiting fast time tagging units," *Phys. Rev. Res.*, vol. 2, Jun. 2020, Art. no. 023287, doi: 10.1103/PhysRevResearch.2.023287.

[15] M. Avesani *et al.*, "Resource-effective quantum key distribution: A field trial in Padua city center," *Opt. Lett.*, vol. 46, no. 12, pp. 2848–2851, Jun. 2021, doi: 10.1364/OL.422890.

[16] M. Avesani, C. Agnesi, A. Stanco, G. Vallone, and P. Villoresi, "Stable, low-error, and calibration-free polarization encoder for free-space quantum communication," *Opt. Lett.*, vol. 45, no. 17, pp. 4706–4709, Sep. 2020, doi: 10.1364/OL.396412.

[17] L. Calderaro *et al.*, "Fast and simple qubit-based synchronization for quantum key distribution," *Phys. Rev. Appl.*, vol. 13, May 2020, Art. no. 054041, doi: 10.1103/PhysRevApplied.13.054041.

[18] C. Agnesi *et al.*, "Simple quantum key distribution with qubit-based synchronization and a self-compensating polarization encoder," *Optica*, vol. 7, no. 4, pp. 284–290, Apr. 2020, doi: 10.1364/OPTICA.381013.

[19] M. Avesani *et al.*, "Full daylight quantum-key-distribution at 1550 nm enabled by integrated silicon photonics," *npj Quantum Inf.*, vol. 7, no. 1, Jun. 2021, Art. no. 93, doi: 10.1038/s41534-021-00421-2.

[20] C. Agnesi, M. Avesani, A. Stanco, P. Villoresi, and G. Vallone, "All-fiber self-compensating polarization encoder for quantum key distribution," *Opt. Lett.*, vol. 44, no. 10, pp. 2398–2401, May 2019, doi: 10.1364/OL.44.002398.

[21] F. Vedovato *et al.*, "Extending wheeler's delayed-choice experiment to space," *Sci. Adv.*, vol. 3, no. 10, 2017, Art. no. e1701180, doi: 10.1126/sciadv.1701180.

[22] A. Balossino *et al.*, "SeQBO—A miniaturized system for quantum key distribution," in *Proc. 71st Int. Astronaut. Congr.*, vol. 2020, Oct. 2020, Art. no. 166680. [Online]. Available: http://iafastro.directory/iac/paper/id/59867/summary/

[23] F. Laudenbach *et al.*, "Continuous-variable quantum key distribution with Gaussian modulation—The theory of practical implementations," *Adv. Quantum Technol.*, vol. 1, no. 1, Aug. 2018, Art. no. 1800011, doi: 10.1002/qute.201870011.

[24] D. G. Marangon, G. Vallone, and P. Villoresi, "Source-device-independent ultrafast quantum random number generation," *Phys. Rev. Lett.*, vol. 118, no. 2, 2017, Art. no. 060503, doi: 10.1103/PhysRevLett.118.060503.

[25] M. Avesani, D. G. Marangon, G. Vallone, and P. Villoresi, "Source-device-independent heterodyne-based quantum random number generator at 17 Gbps," *Nature Commun.*, vol. 9, no. 1, Dec. 2018, Art. no. 5365, doi: 10.1038/s41467-018-07585-0.

[26] H. Fürst, H. Weier, S. Nauerth, D. G. Marangon, C. Kurtsiefer, and H. Weinfurter, "High speed optical quantum random number generation," *Opt. Exp.*, vol. 18, no. 12, pp. 13029–13037, 2010, doi: 10.1364/OE.18.013029.

[27] M. Stipčević and B. M. Rogina, "Quantum random number generator based on photonic emission in semiconductors," *Rev. Sci. Instrum.*, vol. 78, no. 4, 2007, Art. no. 045104, doi: 10.1063/1.2720728.

[28] K. Tamaki, M. Curty, G. Kato, H.-K. Lo, and K. Azuma, "Loss-tolerant quantum cryptography with imperfect sources," *Phys. Rev. A*, vol. 90, no. 5, Nov. 2014, Art. no. 052314, doi: 10.1103/PhysRevA.90.052314.

[29] W.-Y. Hwang, "Quantum key distribution with high loss: Toward global secure communication," *Phys. Rev. Lett.*, vol. 91, Aug. 2003, Art. no. 057901, doi: 10.1103/PhysRevLett.91.057901.

[30] H.-K. Lo, H. F. Chau, and M. Ardehali, "Efficient quantum key distribution scheme and a proof of its unconditional security," *J. Cryptol.*, vol. 18, no. 2, pp. 133–165, Apr. 2005, doi: 10.1007/s00145-004-0142-y.

[31] D. Rusca, A. Boaron, F. Grünenfelder, A. Martin, and H. Zbinden, "Finite-key analysis for the 1-decoy state QKD protocol," *Appl. Phys. Lett.*, vol. 112, no. 17, 2018, Art. no. 171104, doi: 10.1063/1.5023340.

[32] H. C. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theor. Comput. Sci.*, vol. 560, no. P1, pp. 7–11, Dec. 2014, doi: 10.1016/j.tcs.2011.08.039.

[33] H.-K. Lo, X. Ma, and K. Chen, "Decoy state quantum key distribution," *Phys. Rev. Lett.*, vol. 94, Jun. 2005, Art. no. 230504, doi: 10.1103/PhysRevLett.94.230504.

[34] X. Ma, B. Qi, Y. Zhao, and H.-K. Lo, "Practical decoy state for quantum key distribution," *Phys. Rev. A*, vol. 72, Jul. 2005, Art. no. 012326, doi: 10.1103/PhysRevA.72.012326.

[35] B. Korzh, N. Walenta, R. Houlmann, and H. Zbinden, "A high-speed multi-protocol quantum key distribution transmitter based on a dual-drive modulator," *Opt. Exp.*, vol. 21, no. 17, pp. 19579–19592, Aug. 2013, doi: 10.1364/OE.21.019579.

[36] J. Song, Q. An, and S. Liu, "A high-resolution time-to-digital converter implemented in field-programmable-gate-arrays," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 1, pp. 236–241, Feb. 2006, doi: 10.1109/TNS.2006.869820.

[37] M. Fishburn, L. H. Menninga, C. Favi, and E. Charbon, "A 19.6 ps, FPGA-based TDC with multiple channels for open source applications," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 2203–2208, Jun. 2013, doi: 10.1109/TNS.2013.2241789.

[38] H. Chen and D. D.-U. Li, "Multichannel, low nonlinearity time-to-digital converters based on 20 and 28 nm FPGAS," *IEEE Trans. Ind. Electron.*, vol. 66, no. 4, pp. 3265–3274, Apr. 2019, doi: 10.1109/TIE.2018.2842787.

[39] *TDC-GPX High-End Time-to-Digital Converter*, ScioSense, Eindhoven, The Netherlands. Accessed: Feb. 7, 2022. [Online]. Available: https://www.sciosense.com/products/time-to-digital-converters/tdc-gpx-high-end-time-to-digital-converter/

[40] C. Gabriel *et al.*, "A generator for unique quantum random numbers based on vacuum states," *Nature Photon.*, vol. 4, no. 10, pp. 711–715, Oct. 2010, doi: 10.1038/nphoton.2010.197.

[41] K. Taofiq *et al.*, "A photonic integrated quantum secure communication system," *Nature Photon.*, vol. 15, no. 11, pp. 850–856, Nov. 2021, doi: 10.1038/s41566-021-00873-0.