IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 1, 2019, accepted December 17, 2019, date of publication December 20, 2019,
date of current version December 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2961174

An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm

SERGIO SPANÒ[1], GIAN CARLO CARDARELLI[1], (Member, IEEE), LUCA DI NUNZIO[1], ROCCO FAZZOLARI[1], DANIELE GIARDINO[1], MARCO MATTA[1], ALBERTO NANNARELLI[2], (Senior Member, IEEE), AND MARCO RE[1], (Member, IEEE)

[1]Department of Electronic Engineering, University of Rome "Tor Vergata," 00133 Rome, Italy
[2]Department of Applied Mathematics and Computer Science, Danmarks Tekniske Universitet, 2800 Kgs. Lyngby, Denmark

Corresponding author: Sergio Spanò (spano@ing.uniroma2.it)

ABSTRACT In this paper we propose an efficient hardware architecture that implements the Q-Learning algorithm, suitable for real-time applications. Its main features are low-power, high throughput and limited hardware resources. We also propose a technique based on approximated multipliers to reduce the hardware complexity of the algorithm. We implemented the design on a Xilinx Zynq Ultrascale+ MPSoC ZCU106 Evaluation Kit. The implementation results are evaluated in terms of hardware resources, throughput and power consumption. The architecture is compared to the state of the art of Q-Learning hardware accelerators presented in the literature obtaining better results in speed, power and hardware resources. Experiments using different sizes for the Q-Matrix and different wordlengths for the fixed point arithmetic are presented. With a Q-Matrix of size 8×4 (8 bit data) we achieved a throughput of 222 MSPS (Mega Samples Per Second) and a dynamic power consumption of 37 mW, while with a Q-Matrix of size 256×16 (32 bit data) we achieved a throughput of 93 MSPS and a power consumption 611 mW. Due to the small amount of hardware resources required by the accelerator, our system is suitable for multi-agent IoT applications. Moreover, the architecture can be used to implement the SARSA (State-Action-Reward-State-Action) Reinforcement Learning algorithm with minor modifications.

INDEX TERMS Artificial intelligence, hardware accelerator, machine learning, Q-learning, reinforcement learning, SARSA, FPGA, ASIC, IoT, multi-agent.

I. INTRODUCTION

Reinforcement Learning (RL) is a Machine Learning (ML) approach used to train an entity, called *agent*, to accomplish a certain task [1]. Unlike the classic supervised and unsupervised ML techniques [2], RL does not require two separated *training* and *inference* phases being based on a *trial & error* approach. This concept is very close to the human learning.

As depicted in Fig. 1, the agent "lives" in an *environment* where it performs some *actions*. These actions may affect the environment which is time-variant and can be modelled as a Markovian Decision Process (MDP) [1]. An *interpreter* observes the scenario returning to the agent the *state* of the environment and a *reward*. The reward (or reinforcement) is a quality figure for the last action performed by the agent and it is represented as a positive or negative number. Through this iterative process, the agent learns an optimal *action-selection policy* to accomplish its task. This policy indicates which is the best action the agent should perform when the environment is in a certain state. Eventually, the interpreter may be integrated into the agent that becomes self-critic.

Thanks to this approach, RL represents a very powerful tool to solve problems where the operating scenario is unknown or changes over time.

Recently, the applications of RL have become increasingly popular in various fields such as robotics [3]–[5], Internet of Things (IoT) [6], power management [7], financial trading [8] and telecommunications [9], [10]. Another research area in RL is multi-agent and swarm systems [11]–[14].

The associate editor coordinating the review of this manuscript and approving it for publication was Ahmed M. Elmisery.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

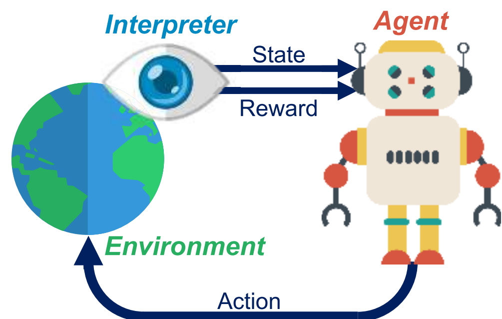

FIGURE 1. Reinforcement learning framework.

This kind of applications require powerful computing platforms able to process very large amount of data as fast as possible and with limited power consumption. For these reasons, software-based implementations performance is now the main limitation in further development of such systems and the use of hardware accelerators based on FPGAs or ASICs can represent an efficient solution for implementing RL algorithms.

The main contribution of this work is a flexible and efficient hardware accelerator for the Q-Learning algorithm. The system is not constrained to any specific application, RL policy or environment. Moreover, for IoT target devices, a low-power version of the architecture based on approximated multipliers is presented.

The paper is organized as follows.

- Section I is a brief survey on Reinforcement Learning and its applications. Q-Learning algorithm, and the related work in the literature are presented.
- Section II describes the proposed hardware architecture, detailing its functional blocks. A technique to reduce the hardware complexity of the arithmetic operations is also proposed.
- Section III presents the implementation results and the comparisons with the state of the art.
- In sec. IV final considerations and future developments are given.
- Appendix shows how the architecture can be exploited to implement the SARSA (State-Action-Reward-State-Action) RL algorithm [15] with minor modifications.

A. Q-LEARNING ALGORITHM

Q-Learning [16] is one of the most known and employed RL algorithms [17] and belongs to the class of off-policy methods since its convergence is guaranteed for any agent's policy. It is based on the concept of *Quality Matrix*, also known as Q-Matrix. The size of this matrix is $N \times Z$ where N is the number of the possible agent's states to sense the environment and Z is the number of possible actions that the agent can perform. This means that Q-Learning operates in a discrete state-action space $S \times A$. Considering a row of the Q-Matrix that represents a particular state, the best action to be performed is selected by computing the maximum value in the row.

At the beginning of the training process, the Q-Matrix is initialized with random or zero values, and it is updated by using (1).

$$Q_{new}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a)\right) \tag{1}$$

The variables in (1) refer to:

- s_t and s_{t+1}: current and next state of the environment.
- a_t and a_{t+1}: current and next action chosen by the agent (according to its policy).
- γ: discount factor, $\gamma \in [0, 1]$. It defines how much the agent has to take into account long-run rewards instead of immediate ones.
- α: learning rate, $\alpha \in [0, 1]$. It determines how much the newest piece of knowledge has to replace the older one.
- r_t: current reward value.

In [16] it is proved that the knowledge of the Q-Matrix suffices to extract the optimal action-selection policy for a RL agent.

B. RELATED WORK

Despite the growing interest for RL and the need for systems capable to process large amount of data in very short time, just a few works can be found in the literature about the hardware implementation of RL algorithms. Moreover, the comparison is hard due to the lack of implementation details and homogeneous benchmarks. In this section we show the most prominent researches in this field.

In 2005, Hwang *et al.* [18] proposed a hardware accelerator for the "Flexible Adaptable Size Topology" (FAST) algorithm [19]. The system was implemented on a Xilinx XCV800 FPGA and was validated using the cart-pole problem [20]. The architecture is well described but few details about the implementation are given.

In 2007, Shao *et al.* [21] proposed a smart power management application for embedded systems based on the SARSA algorithm [15]. The systems was implemented on a Xilinx Spartan-II FPGA. Although the authors proved its functionality, neither the architecture nor the implementation details are given.

One of the most relevant work in the field is [22] by Gankidi et al. that, in 2017, proposed a RL accerelerator for space rovers. The authors implemented the Deep Q-Learning technique [23] on a Xilinx Virtex-7 FPGA. They obtained a throughput of 2.34 MSPS (Mega Samples Per Second) for a 4×2 state-action space.

Also in 2017, Su *et al.* [24] proposed another Deep Q-Learning hardware implementation based on an Intel Arria-10 FPGA. The architecture was compared to a Intel i7-930 CPU and a Nvidia GTX-760 GPU implementation. They achieved a throughput of 25 KSPS with 32 bit fixed point representation for a 27×5 state-action space.

In 2018, Shao *et al.* [21] proposed a hardware accelerator for robotic applications based on "Trust Region Policy Optimization" (TRPO) [25]. The architecture was implemented

on different devices: FPGA (Intel Stratix-V), CPU (Intel i7-5930K) and GPU (Nvidia Tesla-C2070). With respect to the CPU, the authors obtained a speed-up factor of 4.14× and 19.29× for the GPU and the FPGA implementation, respectively.

The most recent works (published in 2019) include Cho *et al.* [26]. They propose a hardware accelerator for the "Asynchronous Advantage Actor-Critic" (AC3) algorithm [27], describing an implementation based on a Xilinx VCU1525 FPGA. The system was validated using 6 Atari-2600 videogames.

In the work by Li *et al.* [28] another Deep Q-Learning network was implemented on a Digilent Pynq development board for the cart-pole problem. The system is meant only for inference mode and, consequently, cannot be used for real-time learning.

One of the most advanced hardware accelerator for Q-Learning was proposed by Da Silva *et al.* [29]. The authors presented an implementation based on a Xilinx Virtex-6 FPGA. Moreover, they performed a fixed-point analysis to confirm the convergence of the algorithm. Different comparisons with state of the art implementations were made. Since this is one of of best performing Q-Learning accelerators at today, we provide an extensive comparison with our architecture (sec. III-B).

II. PROPOSED ARCHITECTURE

The Q-Learning agent shown in Fig. 2 is composed by two main blocks: the Policy Generator (PG) and the Q-Learning accelerator.

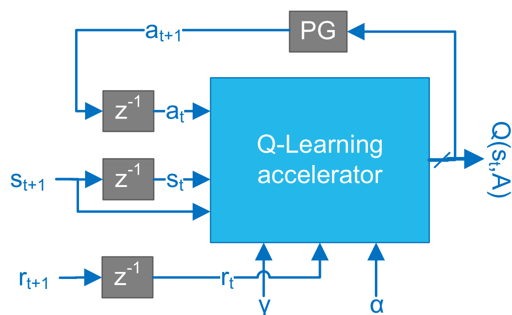

FIGURE 2. High level harchitecture of the Q-Learning agent.

The agent receives the state s_{t+1} and the reward r_{t+1} from the observer, while the next action is generated by the PG according to the values of the Q-Matrix stored into the Q-Learning accelerator.

Note that s_t, a_t and r_t are obtained by delaying s_{t+1}, a_{t+1} and r_{t+1} by means of registers. s_t and a_t represent the indices of the rows and columns of the Q-Matrix, respectively. These delays do not affect the convergence of the Q-Learning algorithm, as proved in [30].

With the aim to design a general purpose hardware accelerator, we do not provide a particular implementation for the PG since it is application-defined. The PG has been included only in the experiments for the comparison with the state of the art (sec. III-B).

Figure 3 shows the Q-Learning accelerator.

The Q-Matrix is stored into Z Dual-Port RAMs, named Action RAMs. Consequently, we have one memory block per action. Each RAM contains an entire column of the Q-Matrix and the number of memory locations corresponds to the number of states N. The read address is the next state s_{t+1}, while the write address is the current state s_t. The enable signals for the Action RAMs, generated by a decoder driven by the current action a_t, select the value $Q(s_t, a_t)$ to be updated. The Action RAMs outputs correspond to a row of the Q-Matrix $Q(s_{t+1}, A)$.

The signal $Q(s_t, a_t)$ is obtained by delaying the output of the memory blocks and then selecting the Action RAM through a multiplexer driven by a_t. A MAX block fed by the output of the Action RAMs generates $\max_a Q(s_{t+1}, A)$.

The Q-Updater (Q-Upd) block implements the Q-Matrix update equation (1) generating $Q_{new}(s_t, a_t)$ to be stored into the corresponding Action RAM.

The accelerator can be also used for Deep Q-Learning [23] applications if the Action RAMs are replaced with Neural Network-based approximators.

A. MAX BLOCK

An extensive study about this block has been proposed in [30]. In the paper, the authors proved that the propagation delay of this block is the main limitation for the speed of Q-Learning accelerators when a large number of actions is required. Consequently, they propose an implementation based on a tree of binary comparators (M-stages) that is a good trade-off in area and speed [31].

This architecture is employed by the Q-Learning accelerators presented in [22], [29] and has also been used in our architecture (Fig. 4).

Moreover, in [30] it is proved that, when pipelining is used to speed up the MAX block, the latency does not affect the convergence of the Q-Learning algorithm. This means that, when an application requires a very high throughput, it is possible to use pipelining.

B. Q-UPDATER BLOCK

Equation (1) can be rearranged as

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{2}$$

to obtain an efficient implementation. Equation (2) is computed by using 2 multipliers, while (1) requires 3 multipliers.

The Q-Updater block in Fig. 5 is used to compute (2), generating $Q_{new}(s_t, a_t)$.

The critical path consists in 2 multipliers and 2 adders. In the next section (II-B1) a method to reduce the hardware complexity for the multipliers is illustrated.

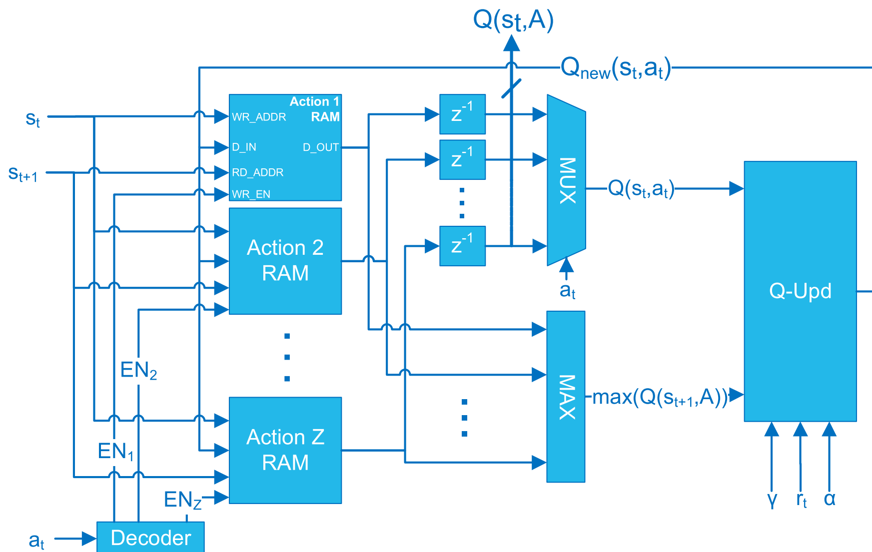

FIGURE 3. Q-Learning accelerator architecture.

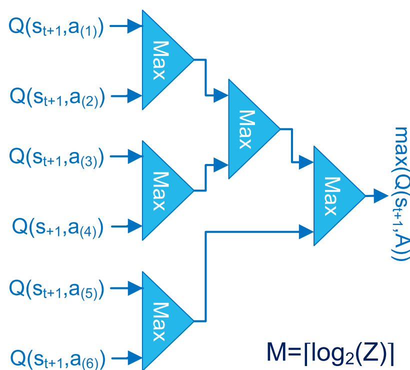

FIGURE 4. MAX block tree architecture for Z = 6.

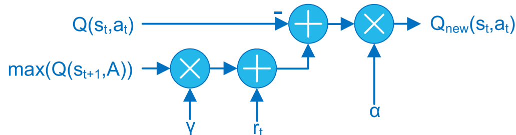

FIGURE 5. Q-matrix updater block architecture.

1) APPROXIMATED MULTIPLIERS

The main speed limitation in the updater block is the propagation delay of the multipliers. Using a similar approach to [32], it is possible to replace the full multipliers shown in Fig. 5 with approximated multipliers based on barrel shifters [33]. In this way, we are approximating α and γ with a number equal to their nearest power of two (single shifter), or to the nearest sum of powers of two (two or more shifters). Due to the fact that $\alpha, \gamma \in [0, 1]$, only right shifts have been used.

Considering a number $x \leq 1$, its binary representation using M bits for the fractional part is:

$$x = x_0 2^0 + x_{-1} 2^{-1} + x_{-2} 2^{-2} + \ldots + x_{-M} 2^{-M} \quad (3)$$

where $x_0, \ldots, x_{-M}$ are the binary digits. Let i, j, k be the positions of the first, second and third '1' in the binary representation of x starting from the most significant bit. Moreover, we define $< x >_{OP_n}$ the approximation of x with the n most significant powers of two in the $M + 1$ bits representation. That is

$$\begin{aligned} < x >_{OP_1} &= 2^{-i} \\ < x >_{OP_2} &= 2^{-i} + 2^{-j} \\ < x >_{OP_3} &= 2^{-i} + 2^{-j} + 2^{-k} \end{aligned} \quad (4)$$

for the approximation with one, two and three powers of two. The concept can be extended to more power of two terms.

For example, $x = 0.101101_{(2)} = 0.703125$ can be approximated as:

$$\begin{aligned} < 0.101101_{(2)} >_{OP_1} &= 2^{-1} = 0.5 \\ < 0.101101_{(2)} >_{OP_2} &= 2^{-1} + 2^{-3} = 0.625 \\ < 0.101101_{(2)} >_{OP_3} &= 2^{-1} + 2^{-3} + 2^{-4} = 0.6875. \end{aligned} \quad (5)$$

Some examples of the approximated values for different powers of two are presented in Fig. 6 ($x \leq 1$).

Consequently, the product $z = x \cdot y$ can be approximated as:

$$\begin{aligned} < z >_{OP_1} &= 2^{-i} \cdot y \\ < z >_{OP_2} &= 2^{-i} \cdot y + 2^{-j} \cdot y \\ < z >_{OP_3} &= 2^{-i} \cdot y + 2^{-j} \cdot y + \cdot 2^{-k} \cdot y. \end{aligned} \quad (6)$$

The approximated multipliers are implemented by one or more barrel shifters in the Q-Updater block, depending on the approximation, as shown in Fig. 7 and 8.

The position of the leading ones i and j in the representation of α and γ can be given as input if constant for the whole computation, or determined by Leading-One-Detectors (LOD) [34] if the values are modified at run time.

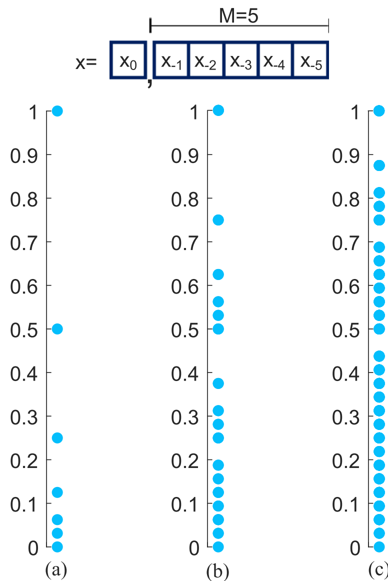

FIGURE 6. Approximated values for a 6-bit number using M = 5 bits for the fractional part. (a) 1 power of two, (b) 2 powers of two, (c) 3 powers of two.

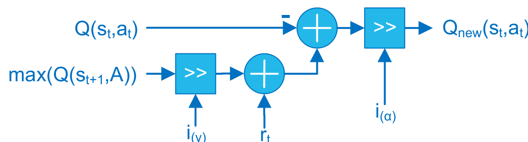

FIGURE 7. Q-Matrix updater block with multipliers implemented by a single barrel shifter.

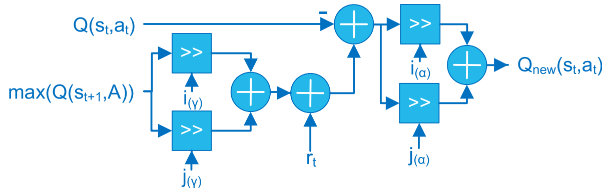

FIGURE 8. Q-Matrix updater block with multipliers implemented by two barrel shifters.

The error introduced by this approximation does not effect the convergence of the Q-Learning algorithm [16] and, as a side effect, we obtain a shorter critical path and lower power consumption (sec. III-A). Moreover, we tested the system in different applications which proved to be almost insensitive to the approximation error since the convergence conditions of Q-Learning are still satisfied ($\alpha,\gamma \leq 1$).

By using approximated multipliers, we avoid to use FPGAs with DSP blocks and we can implement the accelerator in small ultra low power FPGAs suitable for IoT applications [35], [36].

III. IMPLEMENTATION EXPERIMENTS

In order to validate the proposed architecture, we implemented different versions of the Q-Learning accelerator. In the experiments, we used a Xilinx Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit featuring the XCZU7EV-2FFVC1156 FPGA. All the results in this section were obtained using the Vivado 2019.1 EDA tool with default implementation parameters and setting a timing constraint of 2 ns. The system was coded in VHDL.

The design exploration was implemented for the following range of parameters:

- Number of bits for the Q-Matrix values : 8, 16 and 32 bit.
- Number of states N: 8, 16, 32, 64, 128 and 256.
- Number of actions Z: 4, 8 and 16.

We focused the implementation analysis on the following resources [37]:

- Look Up Tables (LUT);
- Look Up Tables used as RAM (LUTRAM);
- Flip-Flops (FF);
- Digital Signal Processing slices (DSP);

For every resource of the device, we also provide the percent usage respect to the total available.

The performances were measured in terms of maximum clock frequency (CLK) and dynamic power consumption (PWR). The latter was evaluated using Vivado after the Place&Route considering the maximum clock frequency and a worst case scenario with a 0.5 activity factor on the circuit nodes [38].

All the implementation examples in this section do not make use of pipelining in the MAX block (sec. II-A). Unless otherwise stated, no approximated multipliers are used.

Tables 1 to 9 show the implementation results for different number of states, actions and data-width for the Q-Matrix values (tables header color: blue 8-bit, red 16-bit, green 32-bit data-widths).

TABLE 1. Implementation results for Q-Matrices with 8 bit data and Z = 4.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	193 0.08%	32 0.03%	154 0.03%	0 0.00%	222	37
16	192 0.08%	32 0.03%	156 0.03%	0 0.00%	240	37
32	193 0.08%	32 0.03%	158 0.03%	0 0.00%	234	41
64	214 0.09%	64 0.06%	160 0.03%	0 0.00%	227	38
128	271 0.12%	80 0.08%	162 0.04%	0 0.00%	239	45
256	362 0.16%	160 0.16%	164 0.04%	0 0.00%	233	61

The first consideration is related to the number of DSPs. Since only one Q-Matrix element is updated per clock cycle, the only parameter that affects the number of required DSPs is the bit-width. For a Q-Matrix with 8-bit data, we obtain the fastest implementations that do not require any DSP slice. For 16-bit and 32-bit data, 3 DSPs and 5 DSPs are required respectively.

Another consideration comes with the maximum clock frequency (that corresponds exactly to the throughput of the

TABLE 2. Implementation results for Q-Matrices with 8 bit data and Z = 8.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	290 0.13%	64 0.06%	252 0.05%	0 0.00%	190	64
16	293 0.13%	64 0.06%	254 0.06%	0 0.00%	168	68
32	294 0.13%	64 0.06%	256 0.06%	0 0.00%	187	65
64	343 0.15%	128 0.16%	258 0.06%	0 0.00%	183	71
128	451 0.20%	160 0.16%	260 0.06%	0 0.00%	210	90
256	632 0.27%	320 0.31%	268 0.06%	0 0.00%	188	109

TABLE 3. Implementation results for Q-Matrices with 8 bit data and Z = 16.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	497 0.22%	128 0.13%	318 0.07%	0 0.00%	147	88
16	498 0.22%	128 0.13%	320 0.07%	0 0.00%	130	94
32	496 0.22%	128 0.13%	322 0.07%	0 0.00%	141	91
64	587 0.25%	256 0.25%	330 0.07%	0 0.00%	149	81
128	717 0.36%	320 0.31%	344 0.07%	0 0.00%	160	114
256	1192 0.52%	640 0.63%	346 0.08%	0 0.00%	167	162

TABLE 4. Implementation results for Q-Matrices with 16 bit data and Z = 4.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	179 0.06%	64 0.06%	250 0.05%	3 0.17%	152	56
16	178 0.06%	64 0.06%	252 0.05%	3 0.17%	152	58
32	180 0.08%	64 0.06%	254 0.06%	3 0.17%	149	60
64	204 0.09%	96 0.09%	256 0.06%	3 0.17%	153	57
128	333 0.14%	160 0.16%	258 0.06%	3 0.17%	158	80
256	513 0.22%	320 0.31%	272 0.06%	3 0.17%	156	88

TABLE 5. Implementation results for Q-Matrices with 16 bit data and Z = 8.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	382 0.17%	128 0.13%	444 0.10%	3 0.17%	129	107
16	383 0.17%	128 0.13%	446 0.10%	3 0.17%	127	108
32	380 0.16%	128 0.13%	448 0.10%	3 0.17%	132	110
64	417 0.18%	192 0.19%	450 0.10%	3 0.17%	127	108
128	691 0.30%	320 0.31%	458 0.10%	3 0.17%	135	124
256	1048 0.45%	640 0.63%	478 0.10%	3 0.17%	136	180

TABLE 6. Implementation results for Q-Matrices with 16 bit data and Z = 16.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	759 0.33%	256 0.25%	577 0.13%	3 0.17%	117	133
16	759 0.33%	256 0.25%	580 0.13%	3 0.17%	117	139
32	761 0.33%	256 0.25%	583 0.13%	3 0.17%	117	142
64	828 0.36%	384 0.38%	592 0.13%	3 0.17%	115	143
128	1386 0.60%	640 0.63%	606 0.13%	3 0.17%	121	187
256	2101 0.91%	1280 1.26%	632 0.13%	3 0.17%	115	284

TABLE 7. Implementation results for Q-Matrices with 32 bit data and Z = 4.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	383 0.17%	96 0.09%	586 0.13%	5 0.29%	136	116
16	383 0.17%	96 0.09%	588 0.13%	5 0.29%	125	113
32	382 0.17%	96 0.09%	590 0.13%	5 0.29%	106	128
64	417 0.18%	160 0.16%	592 0.13%	5 0.29%	116	125
128	682 0.30%	320 0.31%	606 0.13%	5 0.29%	112	142
256	1035 0.45%	640 0.63%	620 0.13%	5 0.29%	110	183

system). Given a certain data-path bit-width and number of actions, the clock frequency remains almost unaltered. This can be ascribed to the different solutions found by routing tool. For this reason, in Fig. 9 we use the average clock frequencies. The frequency drop, when the number of actions increases, is greater for 8 bit data-paths with respect to the 16 and 32 bit cases. This behaviour can be justified by taking into account the major role of FPGA interconnections when a large number of bits is used.

For what concerns the hardware resources, the number of required LUT RAMs is related to the size of the Q-Matrix $N \times Z$. From $N = 8$ to $N = 32$ the resources remain the same, from $N = 64$ a higher number of LUT RAMs is required.

As expected, the power consumption is proportional to the number of required LUTs (considering architectures with the same parameters). The trend can be observed in Figs. 10 and 11.

Even for the largest implementation considered ($N = 256$, $Z = 16$, 32-bit Q-Matrix values), the required FPGA resources are moderate. This suggests that the architecture can be easily employed in applications requiring a large number of states or actions and applications where multiple agents must be implemented on the same device.

The main result of the design exploration shows that we can implement fast Q-Learning accelerators with small amount of resources and low power consumption.

TABLE 8. Implementation results for Q-Matrices with 32 bit data and Z = 8.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	752 0.33%	192 0.19%	940 0.20%	5 0.29%	97	209
16	759 0.33%	192 0.19%	942 0.20%	5 0.29%	102	205
32	763 0.33%	192 0.19%	944 0.20%	5 0.29%	100	220
64	845 0.37%	320 0.31	958 0.21%	5 0.29%	92	233
128	1387 0.60%	640 0.63%	972 0.21%	5 0.29%	103	263
256	2095 0.91%	1280 1.26%	1004 0.22%	5 0.29%	104	329

TABLE 9. Implementation results for Q-Matrices with 32 bit data and Z = 16.

N	LUT	LUTRAM	FF	DSP	CLK (MHz)	PWR (mW)
8	1366 0.59%	384 0.38%	1092 0.24%	5 0.29%	95	244
16	1362 0.59%	384 0.38%	1096 0.24%	5 0.29%	96	260
32	1365 0.59%	384 0.38%	1100 0.24%	5 0.29%	95	248
64	1525 0.66%	640 0.63	1116 0.24%	5 0.29%	97	251
128	2579 1.12%	1280 1.26%	1148 0.25%	5 0.29%	91	408
256	4017 1.74%	2560 2.52%	1210 0.26%	5 0.29%	93	611

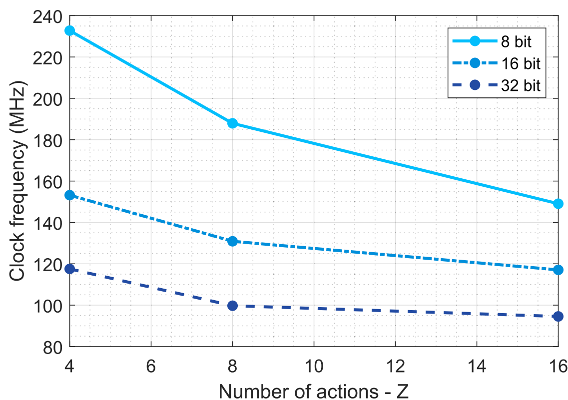

FIGURE 9. Average clock frequency for different Q-Matrix data bit-width vs number of actions.

A. Q-UPDATER BASED ON APPROXIMATED MULTIPLIERS

As discussed in sec. II-B1, to allow the use of IoT devices, the hardware complexity of the Q-Updater block can be reduced by replacing the full multipliers with approximated multipliers based on barrel shifters.

In order to evaluate the benefits of such approach, we implemented the multipliers by using the single power-of-two approach (1 barrel shifter per multiplier) and the more precise approach based on the linear combination of two powers-of-two (two barrel shifters per multiplier), as depicted in Figs. 7 and 8. We considered 8, 16 and 32 bit operands.

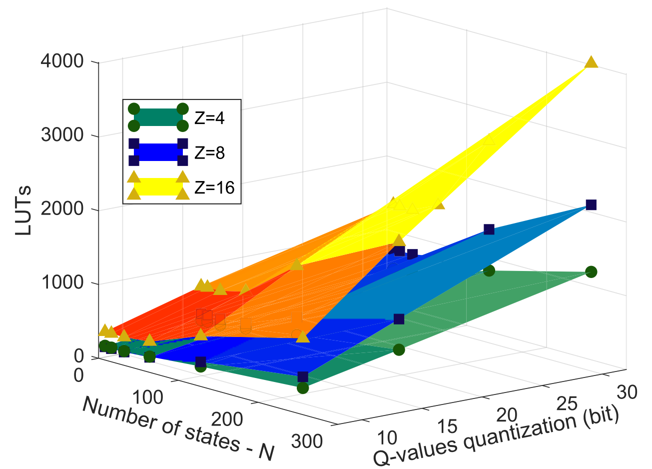

FIGURE 10. Number of LUTs for different implementations.

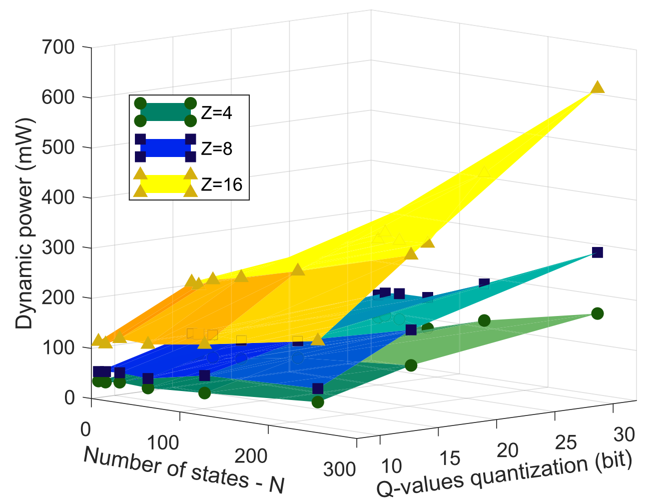

FIGURE 11. Dynamic power consumption for different implementations.

Since the dynamic power consumption is directly proportional to the clock frequency [38], for a fair comparison we provide the energy required to update one Q-Matrix element and the percentage of energy saved respect to the traditional implementation. Tables 10, 11 and 12 show the comparison between the implementations of approximated and full multipliers. Note that for the 8 bit architectures the power dissipation was too low to be accurately estimated. In the traditional implementation, we forced the Vivado synthesizer not to use any DSP block.

TABLE 10. Implementation comparisons: approximated and full multipliers with 8 bit operands.

Impl. type	LUT	CLK (MHz) SpeedUp	PWR (mW)	Energy (nJ)	Energy saving
Full	35 0.01%	795 1×	2	0.002516	*-Ref-*
1 Shift	11 <0.01%	1644 2.07×	<1	<0.000608	>76%
2 Shift	27 0.01%	864 1.09×	<1	<0.004116	>54%

TABLE 11. Implementation comparisons: approximated and full multipliers with 16 bit operands.

Impl. type	LUT	CLK (MHz) SpeedUp	PWR (mW)	Energy (nJ)	Energy saving
Full	155 0.07%	527 1×	9	0.017082	*-Ref-*
1 Shift	33 0.01%	1412 2.68×	2	0.001416	92%
2 Shift	66 0.02%	700 1.33×	4	0.005716	67%

TABLE 12. Implementation comparisons: approximated and full multipliers with 32 bit operands.

Impl. type	LUT	CLK (MHz) SpeedUp	PWR (mW)	Energy (nJ)	Energy saving
Full	731 0.32%	414 1×	49	0.118286	*-Ref-*
1 Shift	87 0.04%	808 1.95×	5	0.006190	95%
2 Shift	184 0.08%	637 1.54×	9	0.014130	73%

The barrel shifter-based architectures do not require any DSP slice, they use less hardware resources, they are faster and more power-efficient than their full multiplier-based counterparts, especially for the 16 and 32 bit implementations. For these reasons, they are suitable for Q-Learning applications on very small and low-power IoT devices at the cost of a reduced set of possible α and γ values.

B. STATE OF THE ART ARCHITECTURE COMPARISON

The architecture proposed in this paper has been compared with one of best performing Q-Learning hardware accerelerators at today [29].

In their paper, Da Silva et al. proposed a parallel implementation based on the number of states N, while in our work the parallelization is based on the number of actions Z. Since in most of the RL applications $Z \ll N$ (see examples in sec. I), our approach results in a smaller architecture.

Another important difference consists in the earlier selection of the Q-matrix value to be updated. This allows to implement a single block for the computations of $Q_{new} = (s_t, a_t)$, while in [29] $N \times Z$ blocks are required. Moreover, in case of FPGA implementations, our architecture allows to employ distributed RAM or embedded block-RAM. This gives an additional degree of freedom compared to [29] where only registers are considered for storing the Q-Matrix values.

To obtain a fair comparison:

- We implemented the same RL environment of [29] and stored the reward values in a Look-Up Table.
- We implemented a random PG as described in [29].
- We considered 16-bit Q-Matrix values.
- We implemented the architectures on the same Virtex-6 FPGA ML605 Evaluation Kit (using the ISE 14.7 Xilinx suite).

The experimental results are shown in Tables 13, 14, 15 and 16. We can only make comparisons with $Z = 4$ and $Z = 8$ since they are the only values implemented in [29]. The implementation results are given in terms of [39]:

- DSP blocks (DSP)
- Slice Registers (REG)
- Slice LUT (LUT)
- Maximum clock frequency (CLK)
- Power consumption (PWR)
- Energy required to update one Q-Matrix element (Energy)

As expected, our architecture employs a constant number of DSP slices, while in [29] this number is proportional

TABLE 13. Da Silva et al. [29] implementation results for 16 bit Q-Matrix values and Z = 4.

N	DSP	REG	LUT	CLK (MHz)	PWR (mW)	Energy (nJ)
6	34 4.16%	548 0.18%	1734 1.15%	24	3	0.127443
12	58 7.55%	1029 0.34%)	3387 2.24%	22	6	0.269421
20	90 11.71%	1670 0.55%	5594 3.71%	20	10	0.496032
30	130 16.92%	2470 0.81%	8872 5.88%	20	13	0.660905
56	234 30.46%	4552 1.51%	15526 10.30%	15	29	1.914191
132	370 48.17%	10533 3.49%	70311 46.65%	13	67	5.018727

TABLE 14. Proposed implementation results for 16 bit Q-Matrix values and Z = 4.

N	DSP	REG	LUT	CLK (MHz)	PWR (mW)	Energy (nJ)
8	3 0.39%	186 0.06%	148 0.09%	72	14	0.193798
16	3 0.39%	188 0.06%	142 0.09%	74	15	0.202922
32	3 0.39%	190 0.06%	147 0.09%	72	14	0.193157
64	3 0.39%	192 0.06%	191 0.12%	74	12	0.163088
128	3 0.39%	194 0.06%	304 0.20%	71	16	0.226053
256	3 0.39%	196 0.06%	512 0.33%	72	20	0.279330

TABLE 15. Da Silva et al. [29] implementation results for 16 bit Q-Matrix values and Z = 8.

N	DSP	REG	LUT	CLK (MHz)	PWR (mW)	Energy (nJ)
12	106 13.80%	1797 0.59%	6960 4.61%	21	10	0.483793
20	170 23.13%	2950 0.97%	11561 7.67%	18	21	1.189128
30	250 32.55%	4390 1.45%	17208 11.41%	16	26	1.580547
56	378 49.21%	8136 2.69%	48536 32.20%	14	48	3.331020

TABLE 16. Proposed implementation results for 16 bit Q-Matrix values and Z = 8.

N	DSP	REG	LUT	CLK (MHz)	PWR (mW)	Energy (nJ)
8	3 0.39%	316 0.10%	316 0.20%	62	20	0.322529
16	3 0.39%	318 0.10%	324 0.21%	62	18	0.289902
32	3 0.39%	320 0.10%	316 0.20%	63	18	0.28454
64	3 0.39%	322 0.10%	423 0.28%	64	22	0.346075
128	3 0.39%	324 0.10%	655 0.43%	63	24	0.380288
256	3 0.39%	326 0.10%	1064 0.70%	59	35	0.591816

to $N \times Z$. The number of Slice Registers required by our implementations remains almost unaltered when the number of states increases, while in [29] it grows with N.

Figure 12 compares the maximum clock frequency for different number of states and actions. Our system is more than 3 times faster and the speed is almost independent to the Q-Matrix number of states.

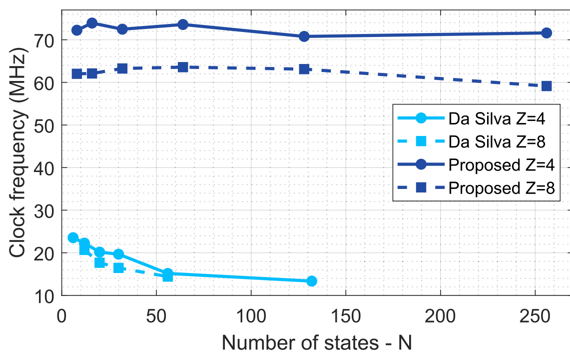

FIGURE 12. Clock frequency comparison between Da Silva et al. [29] and proposed architecture, 16 bit Q-Matrix values.

Figure 13 compares the energy required to update a single Q-Matrix element for different number of states and actions. Also in this case, our architecture, except for the $N = 6$ $Z = 4$ case, presents a better energy efficiency which remains almost unaltered increasing the number of states.

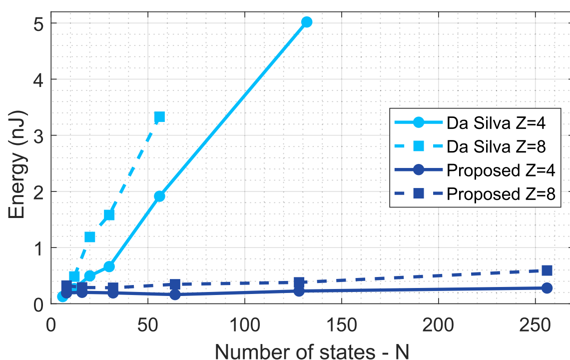

FIGURE 13. Energy required to update one Q-Matrix element comparison between Da Silva et al. [29] and proposed architecture, 16 bit values.

It is important to highlight that the most evident difference between the proposed architecture and [29] is its independence from the environment and agent's policy. This happens because the system in [29] cannot be used as a general-purpose hardware accelerator since the RL environment is mapped on the FPGA. Our system does not have such limitation.

IV. CONCLUSION

In this paper we proposed an efficient hardware implementation for the Reinforcement Learning algorithm called Q-Learning. Our architecture exploits the learning formula by a-priori selecting the required element of the Q-Matrix to be updated. This approach made possible to minimize the hardware resources.

We also presented an alternative method for reducing the computational complexity of the algorithm by employing approximated multipliers instead of full multipliers. This technique is an effective solution to implement the accelerator on small ultra low-power FPGAs for IoT applications.

Our architecture has been compared to the state of the art in the literature, showing that our solution requires a smaller amount of hardware resources, is faster and dissipates less power. Moreover, our system can be used as a general-purpose hardware accelerator for the Q-Learning algorithm, not being related to a particular RL environment or agent's policy.

With little effort, the proposed approach can be also exploited to implement the on-policy version of the Q-Learning algorithm: SARSA. This aspect is further explored in Appendix.

For the above reasons, our architecture is suitable for high-throughput and low-power applications. Due to the small amount of required resources, it also allows the implementation of multiple Q-Learning agents on the same device, both on FPGA or ASIC.

APPENDIX

SARSA ACCELERATOR ARCHITECTURE

The proposed architecture for the acceleration of the Q-Learning algorithm can be easily exploited to implement the SARSA (State-Action-Reward-State-Action) [15] algorithm. Equation (7) shows the SARSA update formula for the Q-Matrix.

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha\,(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{7}$$

Comparing (2) to (7), it is straightforward to note the similarities between the two equations. Since the update of the Q-Matrix depends on the agent's next action a_{t+1}, SARSA algorithm is the on-policy version of the Q-Learning algorithm (which is off-policy).

The resulting architecture is presented in Fig. 14. The main difference between the Q-Learning implementation in Fig. 3

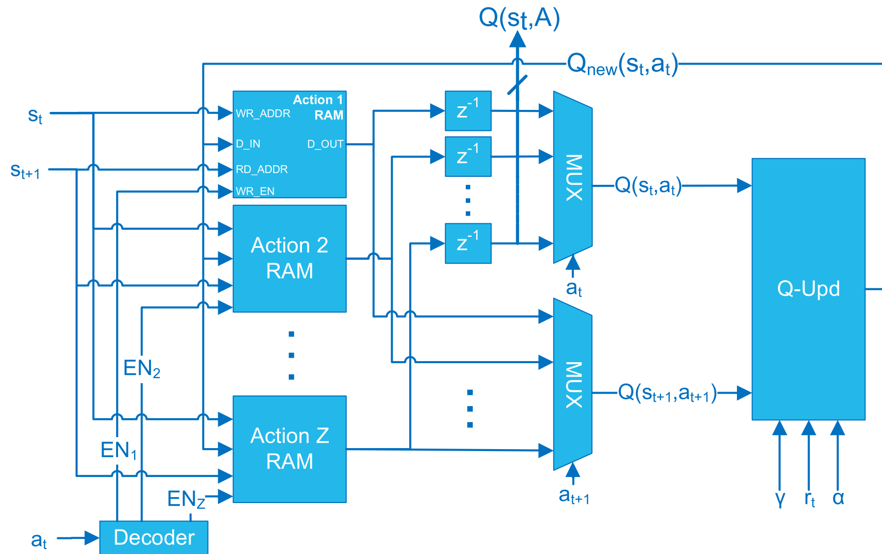

FIGURE 14. SARSA accelerator architecture.

consists in the replacement of the MAX block with a multiplexer driven by the next action a_{t+1}.

The analysis about the Q-Learning architecture can also be extended to the SARSA accelerator.

ACKNOWLEDGMENT

The authors would like to thank Xilinx Inc., for providing FPGA hardware and software tools by Xilinx University Program.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[2] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*, vol. 4. New York, NY, USA: AMLBook, 2012.
[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.
[4] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 5, pp. 1141–1153, Sep. 2013.
[5] J.-L. Lin, K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, "Gait balance and acceleration of a biped robot based on Q-learning," *IEEE Access*, vol. 4, pp. 2439–2449, 2016.
[6] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2375–2385, Aug. 2017.
[7] C. Wei, Z. Zhang, W. Qiao, and L. Qu, "Reinforcement-learning-based intelligent maximum power point tracking control for wind energy conversion systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 10, pp. 6360–6370, Oct. 2015.
[8] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.
[9] M. Matta, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, and S. Spanò, "A reinforcement learning-based QAM/PSK symbol synchronizer," *IEEE Access*, vol. 7, pp. 124147–124157, 2019.
[10] A. He, K. K. Bae, T. R. Newman, J. Gaeddert, K. Kim, R. Menon, L. Morales-Tirado, and J. J. Neel, "A survey of artificial intelligence for cognitive radios," *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1578–1592, May 2010.
[11] Q. Wang, H. Liu, K. Gao, and L. Zhang, "Improved multi-agent reinforcement learning for path planning-based crowd simulation," *IEEE Access*, vol. 7, pp. 73841–73855, 2019.
[12] M. Jiang, T. Hai, Z. Pan, H. Wang, Y. Jia, and C. Deng, "Multi-agent deep reinforcement learning for multi-object tracker," *IEEE Access*, vol. 7, pp. 32400–32407, 2019.
[13] M. Matta, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, F. Silvestri, and S. Spanò, "Q-RTS: A real-time swarm intelligence based on multi-agent Q-learning," *Electron. Lett.*, vol. 55, no. 10, pp. 589–591, 2019.
[14] X. Gan, H. Guo, and Z. Li, "A new multi-agent reinforcement learning method based on evolving dynamic correlation matrix," *IEEE Access*, vol. 7, pp. 162127–162138, 2019.
[15] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Dept. Eng., Univ. Cambridge, Cambridge, U.K., Tech. Rep. CUED/F-INFENG/TR 166, 1994.
[16] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[17] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019.
[18] K.-S. Hwang, Y.-P. Hsu, H.-W. Hsieh, and H.-Y. Lin, "Hardware implementation of FAST-based reinforcement learning algorithm," in *Proc. IEEE Int. Workshop VLSI Design Video Technol.*, May 2005, pp. 435–438.
[19] A. Pérez and E. Sanchez, "The FAST architecture: A neural network with flexible adaptable-size topology," in *Proc. 5th Int. Conf. Microelectron. Neural Netw.*, 1996, pp. 337–340.
[20] S. Geva and J. Sitte, "A cartpole experiment benchmark for trainable controllers," *IEEE Control Syst.*, vol. 13, no. 5, pp. 40–51, Oct. 1993.
[21] S. Shao, J. Tsai, M. Mysior, W. Luk, T. Chau, A. Warren, and B. Jeppesen, "Towards hardware accelerated reinforcement learning for application-specific robotic control," in *Proc. IEEE 29th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2018, pp. 1–8.
[22] P. R. Gankidi and J. Thangavelautham, "FPGA architecture for deep learning and its application to planetary robotics," in *Proc. IEEE Aerosp. Conf.*, Mar. 2017, pp. 1–9.
[23] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, pp. 219–354, 2018.
[24] J. Su, J. Liu, D. B. Thomas, and P. Y. Cheung, "Neural network based reinforcement learning acceleration on FPGA platforms," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 4, pp. 68–73, 2017.
[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy otimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[26] H. Cho, P. Oh, J. Park, W. Jung, and J. Lee, "FA3C: FPGA-accelerated deep reinforcement learning," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 499–513.

[27] A. K. Mackworth, "Consistency in networks of relations," *Artif. Intell.*, vol. 8, no. 1, pp. 99–118, 1977.

[28] M.-J. Li, A.-H. Li, Y.-J. Huang, and S.-I. Chu, "Implementation of deep reinforcement learning," in *Proc. 2nd Int. Conf. Inf. Sci. Syst.*, 2019, pp. 232–236.

[29] L. M. D. Da Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of reinforcement learning Q-learning technique for FPGA," *IEEE Access*, vol. 7, pp. 2782–2798, 2018.

[30] Z. Liu and I. Elhanany, "Large-scale tabular-form hardware architecture for Q-Learning with delays," in *Proc. 50th Midwest Symp. Circuits Syst.*, Aug. 2007, pp. 827–830.

[31] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dündar, "Fast and efficient circuit topologies for finding the maximum of n k-bit numbers," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1868–1881, Aug. 2014.

[32] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, M. Re, and S. Spanò, "AW-SOM, an algorithm for high-speed learning in hardware self-organizing maps," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, to be published.

[33] M. R. Pillmeier, M. J. Schulte, and E. G. Walters, III, "Design alternatives for barrel shifters," *Proc. SPIE*, vol. 4791, pp. 436–447, Jul. 2002.

[34] K. H. Abed and R. E. Siferd, "VLSI implementations of low-power leading-one detector circuits," in *Proc. IEEE SoutheastCon*, Mar./Apr. 2006, pp. 279–284.

[35] H. Qi, O. Ayorinde, and B. H. Calhoun, "An ultra-low-power FPGA for IoT applications," in *Proc. IEEE SOI-3D-Subthreshold Microelectron. Technol. Unified Conf. (S3S)*, Oct. 2017, pp. 1–3. s

[36] Microsemi. *FPGAs*. [Online]. Available: https://www.microsemi.com/product-directory/fpga-soc/1638-fpgas

[37] Xilinx. *Vivado Design Suite User Guide—Synthesis*. Accessed: Jun. 12, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf

[38] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[39] Xilinx. *Synthesis and Simulation Design Guide*. Accessed: Dec. 18, 2012. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/sim.pdf

SERGIO SPANÒ received the B.S. degree in electronic engineering and the M.S. degree *(summa cum laude)* in electronic engineering from the University of "Tor Vergata", Rome, Italy, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering, as a member of the DSPVLSI research group. He had industrial experiences in the space and telecommunications field. His interests include digital signal processing, machine learning, telecommunications, and ASIC/FPGA hardware design. His current research topics related to machine learning hardware implementations for embedded and low-power systems.

GIAN CARLO CARDARILLI (S'79–M'81) was born in Rome, Italy. He received the Laurea degree *(summa cum laude)* from the University of Rome "La Sapienza," in 1981. From 1992 to 1994, he was with the University of L'Aquila. From 1987 to 1988, he was with the Circuits and Systems team, EPFL, Lausanne, Switzerland. He has been with the University of Rome "Tor Vergata," since 1984. He is currently a Full Professor of digital electronics and electronics for communication systems with the University of Rome "Tor Vergata". He has regular cooperation with companies, such as Alcatel Alenia Space, Italy, STM, Agrate Brianza, Italy, Micron, Italy, and Selex S.I., Italy. He works in the field of computer arithmetic and its application to the design of fast signal digital processor. His interests are in the area of VLSI architectures for signal processing and IC design. In this field, he published over than 160 articles in international journals and conferences. His scientific interest concerns the design of special architectures for signal processing.

LUCA DI NUNZIO received the master's degree *(summa cum laude)* in electronics engineering and the Ph.D. degree in systems and technologies for the space from the University of Rome "Tor Vergata," in 2006 and 2010, respectively. He has a working history with several companies in the fields of electronics and communications. He is currently an Adjunct Professor with the Digital Electronics Laboratory, University of Rome "Tor Vergata" and an Adjunct Professor of digital electronics with University Guglielmo Marconi. His research activities are in the fields of reconfigurable computing, communication circuits, digital signal processing, and machine learning.

ROCCO FAZZOLARI received the master's degree in electronic engineering and the Ph.D. degree in space systems and technologies from the University of Rome Tor Vergata, Italy, in May 2009 and in June 2013, respectively. He is currently a Postdoctoral Fellow and an Assistant Professor with the Department of Electronic Engineering, University of Rome "Tor Vergata". He works on hardware implementation of high-speed systems for digital signals processing, machine learning, array of wireless sensor networks, and systems for data analysis of acoustic emission (AE) sensors (based on ultrasonic waves).

DANIELE GIARDINO received the B.S. and M.S. degrees in electronic engineering from the University of Rome "Tor Vergata," Italy, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering and is a member of the DSPVLSI research group. He works on digital development for wideband signals architectures, telecommunications, digital signal processing, and machine learning. In specific, he is focused on the digital implementation of MIMO systems for wideband signals.

MARCO MATTA was born in Cagliari, Italy, in 1989. He received the B.S. and M.S. degrees in electronic engineering from the University of Rome "Tor Vergata", Italy, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree in electronic engineering. Since 2017, he has been a member of the DSPVLSI research group, University of Rome "Tor Vergata". His research interests include the development of hardware platforms, and low-power accelerators aimed machine learning algorithms and telecommunications. In particular, he is currently focused on the implementation of reinforcement learning models on FPGA.

ALBERTO NANNARELLI (S'94–M'99–SM'13) graduated in electrical engineering from the University of Roma "La Sapienza," Roma, Italy, in 1988, and received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California at Irvine, CA, USA, in 1995 and 1999, respectively. He worked for SGS-Thomson Microelectronics and Ericsson Telecom as a Design Engineer and for Rockwell Semiconductor Systems as a summer Intern. From 1999 to 2003, he was with the Department of Electrical Engineering, University of Roma "Tor Vergata," Italy, as a Postdoctoral Researcher. He is currently an Associate Professor with the Technical University of Denmark, Lyngby, Denmark. His research interests include computer arithmetic, computer architecture, and VLSI design. He is a Senior Member of the IEEE Computer Society.

MARCO RE (M'92) received the Ph.D. degree in microelectronics from the University of Rome "Tor Vergata." He is currently an Associate Professor with the University of Rome "Tor Vergata," where he teaches digital electronics and hardware architectures for DSP. He is the Director of a master in audio engineering with the Department of Electronic Engineering, University of Rome Tor Vergata. He was awarded with two NATO fellowships at the University of California at Berkeley, working as a Visiting Scientist with Cadence Berkeley Laboratories. He has been awarded with the Otto Moensted fellowship as a Visiting Professor with the Technical University of Denmark. He collaborates in many research projects with different companies in the field of DSP architectures and algorithms. He is author of about 200 articles on international journals and conferences. His main scientific interests are in the field of low power DSP algorithms' architectures, hardware-software codesign, fuzzy logic and neural hardware architectures, low power digital implementations based on non-traditional number systems, computer arithmetic, and CAD tools for DSP. He is member of Audio Engineering Society (AES).