DNN-based Accelerator for Intelligent Robotic Arm Control with High-Level Synthesis

Yu-Chien Chung[1*], Hao-Hsiang Lian[1*], Yong-Lun Xiao[2], Chih-Tsun Huang[1], Jing-Jia Liou[2],
[1]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300
[2]Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 300

***Abstract*—Intelligent robotics leverages deep learning to boost collaboration between humans and devices. Robotic controllers require a low-latency computation process for a real-time response when facing dynamic situations. Also, in the meantime, more controllers are designed with DNN-based reinforcement learning, which may need more computation power. In this paper, we use high-level synthesis to implement a DNN-based controller on an FPGA. The FPGA is built with an ESP SoC (System-on-Chip) platform, integrated with, and controlled through a host computer. We demonstrated the complete end-to-end controller system on a virtual robotic arm with 1041 times speedup compared with a CPU-based software implementation.**

I. Introduction

Artificial intelligence (AI)-related applications have achieved massive success recently. Still, their high demands for computing power make them challenging to deploy to an environment that lacks of enough computing resources. Meanwhile, the recent advancement of the humanoid robotic arm inspires many researchers to combine deep neural networks (DNNs) to perform human-like actions [1]–[3]. Therefore, DNN-based controllers for robotic applications have been getting more popular [4], [5] because of their flexibility and performance, as compared with the traditional rule-based programming. For example, reinforcement learning is used for a robotic to learn a particular task, e.g., simultaneous localization and mapping (SLAM) [6] by exploring the environments successively. As another example, imitation learning [7] is applied to mimic human behavior, e.g., operation skills. Both learning techniques will be implemented with a specific DNN network (usually LSTM/GRU-based [8] for generating a sequence of actions). The DNN networks can be implemented directly with CPU/GPU frameworks (Tensorflow or Pytorch). However, operation latency is another constraint to be considered for a robotic controller. A robotic controller has to collect sensor data (such as image, object, sound, pressure, etc.) from the environment to make a quick decision on the next actions. The operation cycle of collecting, analyzing, and decision-making needs to fit in a real-time limit. When we use DNN-based learning, the latency requirement may not be easily solvable with CPU/GPU frameworks since they are not optimized for latency and energy efficiency.

In this paper, we proposed to use high-level synthesis (HLS) to implement the DNN network (designed with reinforcement or imitation learning) on an FPGA to satisfy the latency requirement of a robotic controller. The contributions of this work can be summarized as follows:

- Our DNN accelerator was implemented on an open-sourced Embedded Scalable Platform (ESP) [9]. With the proposed scalable architecture, the area-latency trade-off can be done precisely and rapidly using the high-level synthesis. And then, the ESP SoC platform was integrated with a 7-axis robotic arm controller to accelerate the DNN computation process.
- We also improved the communication link (Extended ESPLINK) between the host and the FPGA platform to support the DNN accelerator. With the hardware/software co-optimization, the speedup achieves 1041 times as compared with the software approach.
- Finally, a full mechatronics system was operational to demonstrate the real-time robotic arm control on a physical simulator, Webots [10]. The demonstration also justifies the completeness of our design platform.

II. The System Architecture of Robotic Arm Control

Fig. 1 shows the system architecture of the DNN-based accelerator for robotic arm control. The host executes the main application and acts as a server for flow control. It monitors the physical status of the robotic arm and controls the DMA to transmit the data to and from the accelerator via the interface. The interface protocol between the host and accelerator is implemented on top of the Ethernet port. We also present an improved interface design to support the real-time robotic control. Our accelerator is based on the agile ESP architecture [9], which is an open-source platform for heterogeneous SoC design. The tile-based architecture consists of modular CPU tile with RISC-V Ariane, memory tile, IO tile, and accelerator tile, integrated by a scalable network-on-chip (NoC). The companion methodology enables system-level abstraction and an automated flow from software and hardware co-development to full-system prototyping. For the specialized application of robotic arm control, the DNN engine is modeled

This work was supported in part by Ministry of Science and Technology, Taiwan ROC, under contract MOST 110-2218-E-007-041, MOST 111-2218-E-007-010 and MOST 110-2622-8-007-018.

*These authors contributed equally to this work.

in SystemC and synthesized into RTL with high-level synthesis (HLS) flow. The control parameters to the robotic arm, i.e., the seven motor angles, are fed back to the robotic arm. In this prototyping, our accelerator is implemented in FPGA. The Webots is used to simulate the mechanical robotic arm with physical parameters such as mass, center of mass, velocity, gravity, friction coefficients, etc.

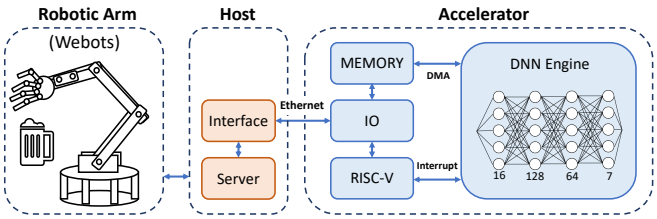

Fig. 1. The system architecture of robotic arm control.

III. The Proposed Accelerator Design

A. Proximal Policy Optimization (PPO) Algorithm

Fig. 2 shows the training model of our reinforcement learning. The environment is implemented by Webots simulator to simulate the physical behavior of the 7-axis humanoid arm. Also, the environment provides the state and reward to the agent. The agent is trained to produce the correspondent action of the robot arm. Actor-critic deep reinforcement learning (RL) with experience replay has drawn much attention from researchers for continuous control problems [11] to optimize the training reward. Among the existing deep RL techniques, Proximal Policy Optimization (PPO) [12] algorithm is simpler to implement, more general, with better sample complexity. So, we adopted PPO's actor-critic architecture as both actor and critic are separate DNN models for controlling the robotic arm to reach the goal destinations. The input state to the DNN include the motor angles on the 7-axis robotic arm, both the position and orientation of the robot palm, and the target destination (i.e., the input state is a 16-tuple). The output actions of the DNN are the target rotation angles of the seven motors for continuous arm control (i.e., the output action is a 7-tuple). Both networks consist of three layers: $16 \times 128 \times 64 \times 7$ neurons for the actor, and $16 \times 128 \times 64 \times 1$ neurons for the critic, respectively. Besides, we use the hardtanh as the non-linear activation function instead of the tanh to simplify the hardware implementation.

The reward function can be defined as follows:

$$Reward = \sqrt{S^2 + T^2} + cos^{-1}(\frac{(S \cdot T)}{\sqrt{S^2} \times \sqrt{T^2}}), \quad (1)$$

where S is the vector of the current position and T is the vector of the destination for the robotic arm. The training process begins with an initial arm position and a random target position. Then, the DNN model tries to move the robotic arm to approach the target point without exceeding the maximum trial number during the training. The success rate can reach 100% in 33 epochs on average. Then, we also applied the post-training quantization from 64-bit float-point numbers to 32-bit fixed-point numbers to improve the hardware efficiency with neglectable error rate.

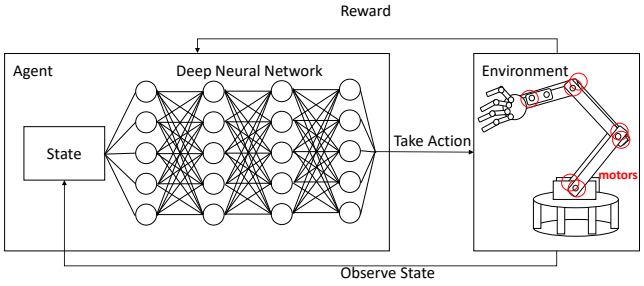

Fig. 2. Reinforcement training model of robotic arm control with PPO.

B. Accelerator Architecture

As Fig. 3 shows, our accelerator consists of three main blocks: Load, Compute, and Store. The Load block is responsible for loading weights, biases and inputs (i.e., the motor angles on the 7-axis robotic arm, both the position and orientation of the robot palm, and the target destination) from DRAM to Private Local Memory (PLM) through the DMA (Direct Memory Access) controller. It also checks the handshaking flags to ensure data readiness. The Compute block performs the DNN operations and stores the intermediate results and final PPO outputs (i.e., the target rotation angles of the seven motors) in the PLM. The number of processing elements (PEs) in the Compute block is scalable (the figure shows eight PEs in our prototyping). These PEs mainly perform the fixed-point matrix multiplication. There is also a dedicated activation module for the hardtanh function in this block. Finally, the Store block writes the result and handshaking flags back into DRAM to inform the host of the valid outputs.

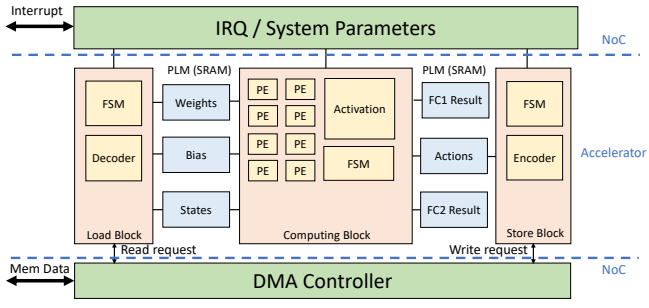

Fig. 3. The proposed accelerator design.

We designed our HLS-based accelerator and its flexible handshaking protocol with constrained resources. The HLS utilizes SystemC, a C++-based hardware description language, to describe the hardware accelerator and its synthesis constraints. Fig. 4 lists the core segment of tiling description in SystemC for a fully connected layer of M input neurons and N output neurons as shown in Fig. 5. The first for-loop tiles the

dimension of N by the number of PEs. The data movement of all inputs, weights, and biases from DRAM to PLM is also performed inside this loop. Similarly, the second for-loop partitions the dimension of M into tiles. Finally, the third for-loop loads the input features and filter weights from the PLM and executes multiplications and accumulations (MAC). The directive of HLS_UNROOL_LOOP(ON) is to control the HLS process of Cadence Stratus to unroll this loop. After the inner two loops are completed, the output activations are stored back to the PLM. The two-level tiling also enables the optimization between the PLM size and data reusability.

```
for (int n = 0; n < N; n += PE_num){
    load_from_DRAM_to_PLM();
    int acc[PE_num] = {0};
    for (int m = 0; m < M; m++){
        for (int i = 0; i < PE_num; i++){
            HLS_UNROLL_LOOP(ON);   // Unroll parallel PEs
            int ifeature = (PLM_ifeature[m * PE_num + i]);
            int weight = (PLM_weight[m * PE_num + i]);
            acc[i] += ifeature * weight;
        }
    for (int i = 0; i < PE_num; i++)
        PLM_result[n + i] = acc[i];
    }
}
```

Fig. 4. SystemC description of DNN engine.

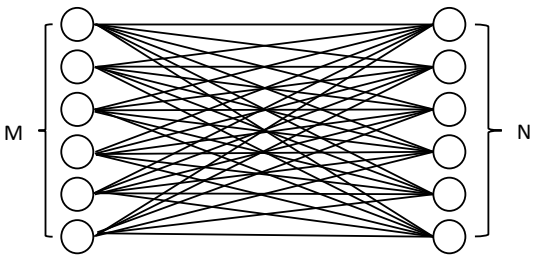

Fig. 5. Fully connected layer in deep neural network.

Due to the flexibility of HLS design, we configured three different PE numbers: 4, 8, and 16 in our accelerator design, and showed the results in Table I. We listed the latency of Load, Compute and Store in rows 2-4. Except for Store operations, both Load and Compute latency (and the total latency) decrease with more PEs added to the accelerator. However, the decreased rate does not correspond to the number of PEs, i.e., it is sub-linear. This is due to the fact that there is a limited number of ports of the PLM (private SRAM). To further reduce latency with more PEs, we need to balance the PLM bandwidth of each PE group carefully. Here we set the PLM port number to be 8 for our experiments.

The last row of Table I compares the normalized AT^2, where A is the number LUT on FPGA and T is the total latency (also refer to Fig. 6). In this prototyping, one spatial

TABLE I
LATENCY (CYCLES), AREA (LUT), AND AT^2 COMPARISON OF DIFFERENT PE NUMBERS IN RTL SIMULATION

# PEs	4	8	16
Load	24,489	20,481	18,477
Compute	16,608	12,735	11,043
Store	53	53	53
Total Latency	41,150	33,269	29,573
Area (Total LUT)	12,903	19,439	32,581
Normalized AT^2	1.02	1	13.24

tile consists of eight PEs (i.e., eight adders and eight multipliers in parallel) as the configuration has the optimized AT^2. To maximize the flexibility of the accelerator, the layer size and number of layers are configurable in the accelerator module, allowing users to adjust for various applications in the software on the RISC-V CPU. With proper hardware support, different non-linear activation functions or extended neural networks are also possible.

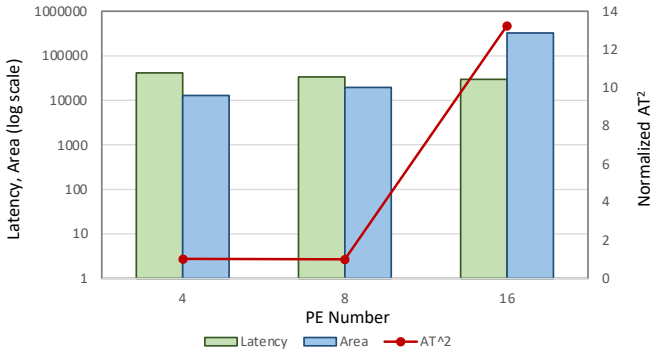

Fig. 6. The comparison of area, latency, and AT^2 for different number of PEs.

The SystemC design is synthesized into the RTL description by Cadence Stratus. Then the hardware accelerator is implemented on the FPGA by using the design flow of Xilinx Vivado. Finally, the integrated system is operated using the ESP software stack to interact with the Webots. We also implement the hardware performance counter in SystemC to profile the operating cycles of the accelerator. Table II compares the latency of a single robot action in cycles among SystemC simulation of the accelerator, RTL simulation of the accelerator, RTL simulation of the system, and the FPGA measurement of the system. The result shows that the SystemC simulation is too optimistic without considering the realistic latency of external DRAM access. However, SystemC simulation provides a fast evaluation in the early design stage. Once entering the RTL stage, our hardware counter can provide a consistent and precise performance indicator to monitor and evaluate the entire acceleration system.

IV. THE DATA COMMUNICATION

The ESP platform supported two types of data communication, Secure Copy (SCP) and ESPLINK [13]. The SCP

TABLE II
COMPARISON OF THE PERFORMANCE COUNTER (CYCLES)

Method	Load	Compute	Store	Total
ACC SystemC Simulation	20,481	203	53	20,534
ACC RTL Simulation	20,481	12,735	53	33,269
System RTL Simulation	21,956	12,735	53	34,744
System FPGA Measurement	24,439	12,745	53	37,227

is a Linux-based method based on SSH; ESPLINK is a customized protocol on top of the Ethernet port, supporting the data communication for both the bare-metal and Linux-based applications. The SCP is mainly to establish a secure data transmission, which takes about a second to establish the link. A large chunk of data with security requirements is suitable for applying the SCP. On the other hand, ESPLINK can be used for transferring many data blocks of small size.

A. Extended ESPLINK

Replacing the SCP with the ESPLINK improves the system latency significantly. However, there is still room for further advancement. Fig. 7(a) illustrates the ESPLINK communication with the FPGA. The host sends the request to the FPGA to establish the communication link. After the acknowledgment, the data is transmitted. Then, the second acknowledgment is used to close the communication. From our experience, the returning path from the FPGA usually requires a wait time of 40× longer than that of sending the data to the FPGA. Considering that the communication between the host and accelerator is local and well confined, we can further simplify the communication when there is only a single host in the system. As shown in Fig. 7(b), the data with the request can be sent by the host directly; the result with the acknowledgment can be returned. The communication can be done with single handshaking.

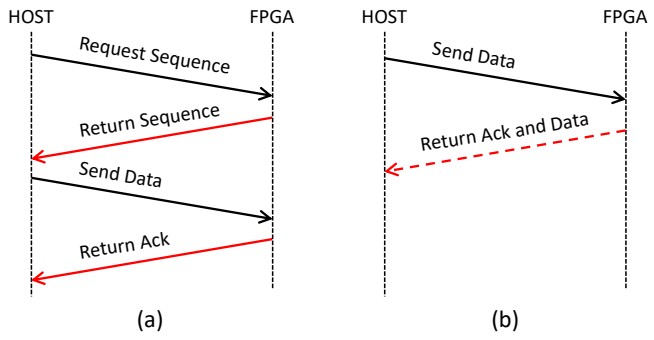

Fig. 7. ESPLINK handshaking mechanism with the FPGA: (a) original protocol; (b) extended version, with the dashed line indicates that the return only happens when the host requests to get data from the FPGA.

B. Improved Host-Accelerator Handshaking

The data movement from the host to the FPGA requires the host to write the inputs into the DRAM tile and raise the handshaking flag to indicate the data is ready. Similarly, once the FPGA updates the output to the memory, it also updates the flag status. So the host can be aware of the readiness of the result. If the data and flag locate apart, the transmission needs separate packets to compete, resulting in a significant latency overhead for the additional packet. Thus, we arrange the handshaking flag in between the input data and output data, as shown in Fig. 8. The continuous memory footprint allows a single packet to efficiently move data in and out.

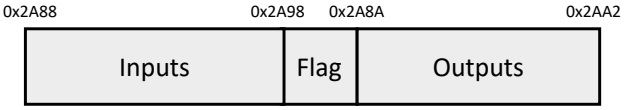

Fig. 8. Memory footprint for handshaking.

C. Performance Evaluation

Table. III compares the different data communication schemes, including the SCP, ESPLINK, and Extended ESPLINK. The data transfer of 1 Kbytes is evaluated with different block sizes of transmission, i.e., 16 bytes, 64 bytes, 256 bytes, and 1024 bytes. The total transfer time of the SCP decreases with the transmission block size increases due to the large overhead of establishing the secure connection. The ESPLINK performs a lot faster. However, the protocol overhead becomes dominant with the block size larger than 256 bytes. By contrast, the proposed Extended ESPLINK gains consistent improvement as the packet size grows. For the block size of 1024 bytes, the data bandwidth achieves 0.01 seconds per 1024 bytes, which is equivalent to 100 Kbytes per second.

TABLE III
COMPARISON OF LATENCY (SECONDS) FOR DATA OF 1024 BYTES

Block Size	SCP	ESPLINK	Extended ESPLINK
16 Bytes/Block	87.142	0.261	0.096
64 Bytes/Block	23.374	0.064	0.026
256 Bytes/Block	5.529	0.032	0.014
1024 Bytes/Block	1.431	0.040	0.010

V. EXPERIMENT RESULT

In the experiments, we measure the overall performance of the system. The latency of the system includes the Extended ESPLINK communication handshaking and computation time by the HLS-based accelerator for computing Proximal Policy Optimization (PPO). We also implement the PPO algorithm as a C program for comparison. Both the ESP platform and the accelerator are implemented on a Xilinx-VCU118 FPGA evaluation board. Also, the C program is run on a RISC-V Ariane core [14], a 6-stage 64-bit in-order RV64IMAC processor. For the test cases, we use Webots to generate 500 robotic arm configurations. The whole setup, including the FPGA board and host, is shown in Fig. 9. Note that the proposed acceleration system interacts with the humanoid robotic arm simulator in real-time. Moreover, our DNN-based acceleration system can also engage with the realistic robotic arm flawlessly thanks to the consistency of the interface.

Table. IV lists the measured average latency considering two computation modules (i.e., RISC-V CPU and Accelerator) and three communication methods (i.e., SCP, ESPLINK, and Extended ESPLINK). Since the SCP is the slowest communication method, its latency dominates overall performance. There is no difference in average iteration/s for both the CPU and Accelerator with the SCP. When replacing with the ESPLINK, we improve the throughput from 33.1 to 42.4 iteration/s with the accelerator. And finally, with the Extended ESPLINK, we can compute the PPO at 353.9 iteration/s, which justifies the efficiency of the accelerator design. The HLS-based DNN accelerator is over four times faster as compared with the CPU-based approach on the FPGA prototyping (from 85.3 iterations/sec to 353.9 iterations/sec). Together with the proposed extended ESPLINK, the speedup achieves 1041× over the baseline approach. The analysis also shows that the system-level evaluation helps identify the performance bottleneck and leads to efficient design optimization.

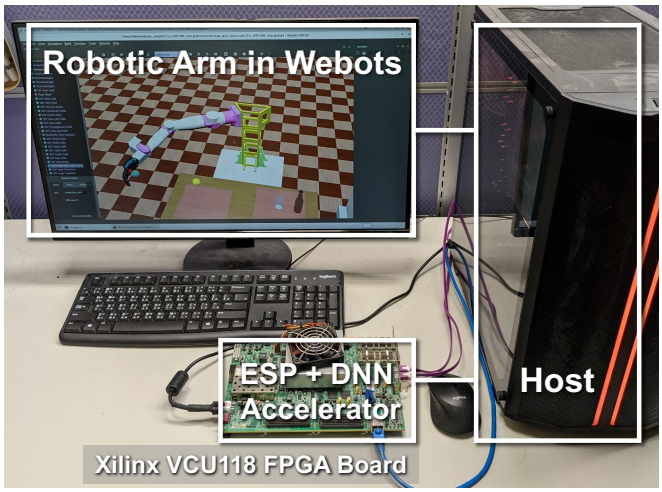

Fig. 9. System integration of FPGA and robotic arm in Webots.

TABLE IV
COMPARISON OF THE OVERALL PERFORMANCE

Module	Method	Iterations/sec	Speedup
RISC-V CPU	SCP	0.34	(Baseline) —
	ESPLINK	33.1	97.4×
	Extended ESPLINK	85.3	250.9×
Accelerator	SCP	0.35	1.0×
	ESPLINK	42.4	124.7×
	Extended ESPLINK	353.9	1040.9×

Currently, we are also working on an extended robotic arm controller that adopts the conditional variational autoencoder (CVAE) with a much improved success rate for obstacle avoidance. The accelerator engine will implement recurrent neural network (RNN) of CVAE by using the HLS flow, and again integrate into our design platform to showcase the effectiveness.

VI. CONCLUSION

This paper demonstrates a flexible platform to design application-specific DNN accelerators for an intelligent robotic controller. Leveraging the modularized design of ESP, we can plug in different accelerators to target the performance for diverse applications. By the proposed Extended ESPLINK and HLS-based accelerator, we show that the average throughput of a PPO algorithm can achieve 353.9 iteration/s, which is 1041× faster compared with the baseline of CPU-based software and SCP communication. We also present a mechatronics system, integrating the real-time DNN-based robotic arm control with a cyber-physical simulator to evaluate the effectiveness of the proposed software/hardware acceleration architecture. Our architecture not only allows the designers to build up a rapid FPGA prototyping but also enables a systematic SoC design methodology for DNN acceleration on a cyber-physical system.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] H. Modares, I. Ranatunga, F. L. Lewis, and D. O. Popa, "Optimized assistive human–robot interaction using reinforcement learning," *IEEE transactions on cybernetics*, vol. 46, no. 3, pp. 655–667, 2015.

[3] C. Kim and H.-J. Yoo, "Energy-efficient DNN processor on embedded systems for spontaneous human-robot interaction," *Journal of Semiconductor Engineering Vol*, vol. 2, no. 2, 2021.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[6] M. Klingensmith, S. S. Sirinivasa, and M. Kaess, "Articulated robot motion for simultaneous localization and mapping (arm-slam)," *IEEE robotics and automation letters*, vol. 1, no. 2, pp. 1156–1163, 2016.

[7] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann, "Imitation learning of dual-arm manipulation tasks in humanoid robots," *International Journal of Humanoid Robotics*, vol. 5, no. 02, pp. 183–202, 2008.

[8] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "Learning real manipulation tasks from virtual demonstrations using lstm," *arXiv preprint arXiv:1603.03833*, 2016.

[9] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC development with open ESP," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[10] O. Michel, "Cyberbotics Ltd. Webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[11] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, and L. P. Carloni, "Accelerator integration for open-source SoC design," *IEEE Micro*, vol. 41, no. 4, pp. 8–14, 2021.

[14] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, nov 2019.