2024 IEEE International Conference on Rebooting Computing (ICRC) | 979-8-3315-4127-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRC64395.2024.10937017

A Configurable CPG Controller using Connectome based SNN on FPGA for Robot Locomotion

Joseph Ereifej
Electrical & Computer Engineering
University of Pittsburgh
Pittsburgh, Pennsylvania
JHE7@pitt.edu

Kevin Araujo
Neuraville Inc.
Pittsburgh, Pennsylvania
kevin.araujo@neuraville.com

Mohammad Nadji-Tehrani
Neuraville Inc.
Pittsburgh, Pennsylvania
nadji@neuraville.com

Rajkumar Kubendran
ECE & Biongineering
University of Pittsburgh
Pittsburgh, Pennsylvania
rajkumar.ece@pitt.edu

***Abstract*—Movement control in autonomous robots requires low-power, real-time models, especially for bio-mimetic locomotion in challenging terrains. Human intervention is often impractical in such environments, making specialized neural networks like central pattern geterators critical for offloading computational resources. This paper presents a controller for a bipedal robot using a modified spiking neuron model, adapted for efficient deployment on field-programmable gate arrays. Key modifications were made to ensure lightweight, real-time performance. Additionally, we leverage a unique open-source neuromorphic software platform for the network design and deployment, making the technology accessible to developers aiming to implement autonomous robot locomotion.**

***Index Terms*—Central Pattern Generators, Spiking Neural Networks, Field Programmable Gate Array, Neuromorphic Computing, Neuromorphic Hardware, Robotics**

I. Introduction

In the swiftly advancing field of robotics, locomotion continues to be a point of constant breakthroughs. In a competitive field, successful deployment often depends on their ability to traverse in diverse environmental conditions. This can improve through two factors – adaptability and power consumption. This is especially true for robots aiming to be highly autonomous or deployed in situations where it is unsafe for humans, such as space or collapsing buildings. Here, robots need to dynamically change their trajectory, speed, or even means of locomotion. Furthermore, these systems need to operate at low power, such that the robot does not deplete its battery in a location where it cannot communicate or return safely. Central pattern generators (CPGs) offer a control system that delivers both adaptability and efficient power consumption.

The concept of CPGs first emerged in the early 1900s by physiologist Thomas Graham Brown as a network of self-governing neurons [1]. These neural networks – within the nervous system – could take input from external inputs and execute control signals for locomotion. CPGs could provide dynamically evolving movement patterns independent of the brain, allowing the organism to put more attention into higher need tasks. In terms of hardware systems, the decoupling of locomotion from the brain is important for biological systems as it allows movement to occur as a subtask for the body rather than requiring full attention. This concept led CPGs to be an ideal motor controller for robotic locomotion.

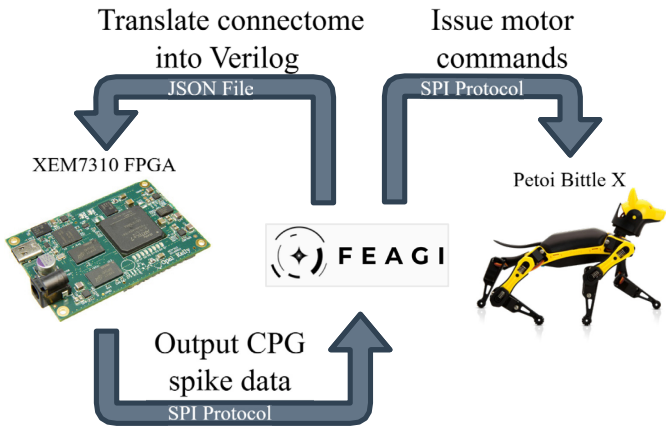


Fig. 1. Top Level Diagram showing the connections between the XEM7310, FEAGI interface, and Petoi Bittle X over SPI

In the past decade, researchers have implemented and deployed CPGs with different methods, namely spiking CPGs. Furthermore, research has identified Field Programmable Gate Arrays (FPGAs) as an ideal platform for a controller as they are reconfigurable, high-speed, and low power [2]. This paper aims to build on those ideas in previous works to develop a more streamlined and accessible FPGA-based solution using the open-source neuromorphic platform FEAGI (Framework for Evolutionary Artificial General Intelligence) as a basis for the network. FEAGI is an open-source spiking neural network simulator designed for rapid prototyping of neural circuits to control embodied systems. Its no-code interface was used to design, visualize, and fine-tune the CPG network, simplifying the development process. The FEAGI platform enables the development of a CPG from a connectome for a unique model grounded in neuroscience principles. For this purpose, we have chosen the Petoi Bittle X for its open-source code and ESP-32 microcontroller, enabling serial communication via physical and Bluetooth connection. Moreover, we have chosen the Opal Kelly XEM7310 for its high-abstraction communication methods over serial protocol.

978-8-3315-4127-9/24/31.00©2024IEEE

In summary, we propose an approach to robotic locomotion found in biological control systems for spiking neurons. An overview of this design is show in Fig. 1. To our knowledge, this is the first design process of this nature with support from a large and ever-growing user base of open-source tools with FEAGI and Petoi.

This paper is organized as follows. In Section II, we discuss the background on the neuron model and specific CPG network. Section III will provide details on the implementation using the FPGA development board and Petoi robot, as well as the overall design of the system. In Section IV, we discuss performance and benchmarking results. Finally, in section V, we reflect on our results.

II. Background

A core motivation in the exploration of this work is to develop a bio-mimetic CPG system that can be easily scaled to high-volume applications. For this, we have taken care to design a system at multiple scales of abstraction.

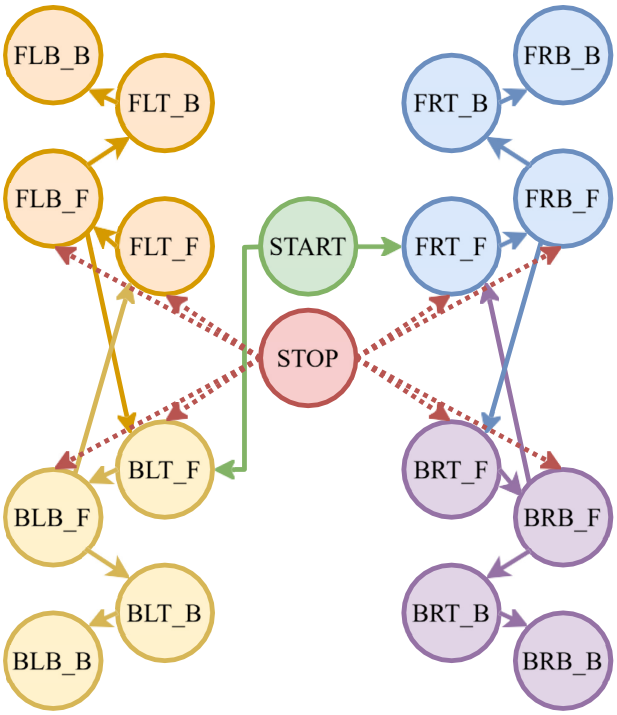
First, we discuss the decision to implement the Leaky Integrate and Fire (LIF) neuron over other more accurate models. Furthermore, we extend this to an 18-neuron CPG network for a bipedal robot with discussion on the manner of control and capacity for reconfiguration.

A. Leaky Integrate-and-Fire Neuron Model

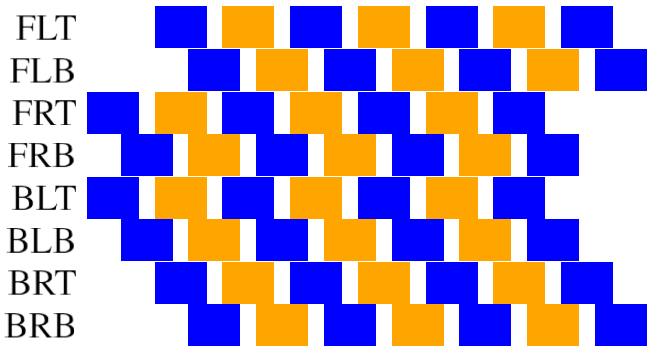
In both neuroscience and neuromorphic engineering, the study of neuron models has been a thoroughly discussed topic. For neuroscientists, a model that provides an elegant yet exhaustive description of the underlying mechanisms within and around the neuron is key. In related work [3], we explore a bursting neuron model for a CPG controller with the Petoi robot. This model, while more bio-realistic, presents a burden on the computational resources of it's platform. In this vein, engineers must work with the design constraints of their given domain. For this work, it is critical to balance bio-realism with simplicity. Considerations for selecting a model include digital footprint, complexity, and desired spiking behavior. We aim to capitalize on a small digital footprint with low complexity by using regular spiking behavior [4]. For this, we chose the Leaky Integrate and Fire (LIF) neuron model. The LIF neuron model capitalizes on the fact that the behavior of the action potential is roughly the same across any given neuron [5]. Therefore, information is transmitted through spike outputs, rather than the shape of the action potential. This simplifies the model to be described by (1).

$$\tau_m \frac{du}{dt} = -[u(t) - u_{rest}] + RI(t) \quad (1)$$

Where the membrane potential $u(t)$ returns to u_{rest} when it reaches or exceeds the threshold voltage θ. In the scope of electrical engineering, this is a leaky RC-circuit evolves over time, governed by the time constant τ_m, where R and C are in parallel.



(a) CPG Network Architecture



(b) Rythmic Timing Diagram for Gait Pattern

Fig. 2. (a) CPG Network Architecture and (b) Rythmic Timing Diagram. Here the naming convention is as follows: **FLT** is Front Left Top, **FLB** is Front Left Bottom, **FRT** is Front Right Top, **FRB** is Front Right Bottom, **BLT** is Back Left Top, **BLB** is Back Left Bottom, **BRT** is Back Right Top, **BRB** is Back Right Bottom. In (a), the suffixes **B** and **F** represent backward and forward respectively. Solid lines represent excitatory connections and dotted lines for inhibitory. In (b), blue events represent forward spikes and orange events backward.

B. Bipedal Locomotion using CPGs

Central pattern generators (CPGs) are self-contained networks of neurons responsible for various unconscious tasks in organisms. Neuroscientists have observed their role in biological processes such as breathing, chewing, and, most relevant to this work, locomotion. In one exploration, scientists fabricated a silicon chip to control spinal cords in lampreys, proving the fundamental reasoning that CPGs can be modeled using electrical systems [6]. In another study, researchers explore complex bipedal locomotion through a hierarchical model, with a top-level controller for synchronization and modulation and a low-level for joint control [7]. This provided the modality for precise movement, while maintaining composure in the whole system. For this work's implementation of bipedal locomotion, we model the system using 16 neurons: each leg is represented by four neurons, corresponding to the top and bottom joints. Each joint has two neurons, one for forward motion and one for backward motion. Additionally, we incorporate two control neurons – start and stop – to activate and inhibit the network.

As shown in Fig. 2(a), each leg consists of a cluster of four neurons. The spiking activity propagates in a specific sequence: top forward, bottom forward, top backward, and bottom backward. Moreover, the forward and backward motions of the opposing legs are coupled so that the front left and back right legs move in sync, while the front right and back left legs move in sync but out of phase with the first pair. This configuration produces a trotting gait pattern with four distinct states. Additionally, the forward-bottom neurons in each leg excite the forward-top neurons in the same-side-leg cluster. So, the FLB neuron excites the BL-cluster and the BLB neuron excites the FL-cluster, and similarly for the FR and BR clusters. To initialize movement, the start neuron first excites the clusters controlling the front-right and back-left legs. To stop, the stop neuron inhibits the four forward-acting neurons.

This behavior is further observed in Fig. 2(b) where the network begins with the FRT and BLT neurons exciting proceeding neurons which then in turn excite other clusters in the network.

C. FEAGI Interface

FEAGI [8], [9] (Framework for Evolutionary Artificial General Intelligence) is an open-source neuromorphic software platform that facilitates the rapid design and development of brain-inspired control systems. Built on the foundational principles of spiking neural networks, FEAGI employs event-based computation alongside a user-friendly, no-code interface. This combination enables intuitive design, development, and integration of adaptable learning systems capable of controlling both physical and simulated embodiments.

At the core of FEAGI's architecture is the concept of the *artificial genome*, a dense data structure that encodes the spiking neural network's structure, mappings, parameters, and configurations through indirect encoding. This genome serves as a blueprint for the neural network, enabling intricate spatial mappings between layers of neurons, much like how biological neurons project axons to target neurons. These mappings allow for flexible and complex neural connectivity, which can be fine-tuned for different applications. The platform is designed in a modular fashion, where microcircuits – bundled as distinct *brain regions* – act as building blocks that can be assembled into larger, more sophisticated neural circuits. The comprehensive data structure that details individual neurons, synapses, and their run-time state parameters is referred to as the *connectome*.

In this project, we used the web-based version of FEAGI, known as Neurorobotics Studio (NRS), to design, develop, and tune a CPG network. NRS provided an intuitive 3D brain visualizer, which we utilized to create independent motor neurons packaged as isolated nodes, referred to as *cortical areas*. Our chosen embodiment, the Petoi robot dog, features four legs, each controlled by two servomotors. To achieve effective motor control, we allocate two motor neurons to each servo: for moving the servo forward and backward. Once all motor neurons were defined, we use circuit builder in FEAGI to establish the synaptic mappings necessary to form the CPG network. As depicted in Fig. 3, excitatory connections, in green, were used to link motor neurons, forming the core of the CPG network. To enable the ability to stop the network, we added inhibitory connections, in red, to select motor neurons. These were activated by a neuron responsible for issuing the stop command, ensuring precise control over motor activity.

III. Implementation

A. Modified LIF Neuron Model

As discussed, we have chosen the LIF neuron for our implementation of a CPG network. As it stands, we do not need to compute the decay using an exponential function, instead considering the next-state value of the potential as a factor of the previous state in addition to the input current. So, the solution for the next state value of the membrane potential $u(t+1)$ can be further simplified into the form of (2).

$$u(t+1) = I(t) + \beta u(t) \tag{2}$$

The membrane potential $u(t)$ decays at a rate of β without input. With an input current $I(t)$ being a weighted sum of each synaptic current with its weight. When the threshold voltage θ is reached, $u(t)$ is reset to 0 and a spike event is assigned to the output. The following high-level circuit diagram outlines the behavior of this neuron.

As seen in Fig. 4, the neuron can be reduced to a multiplexer, comparator, and flip-flop. The MUX is controlled by the presence of a spike at the current time step. The comparator will output high when $u(t)$ is greater than θ, otherwise it will remain low.

B. CPG Architecture on FPGA

For this work, we selected the Opal Kelly XEM7310 development board, which integrates the Artix-7 FPGA along with various peripherals and external hardware interfaces. The Opal

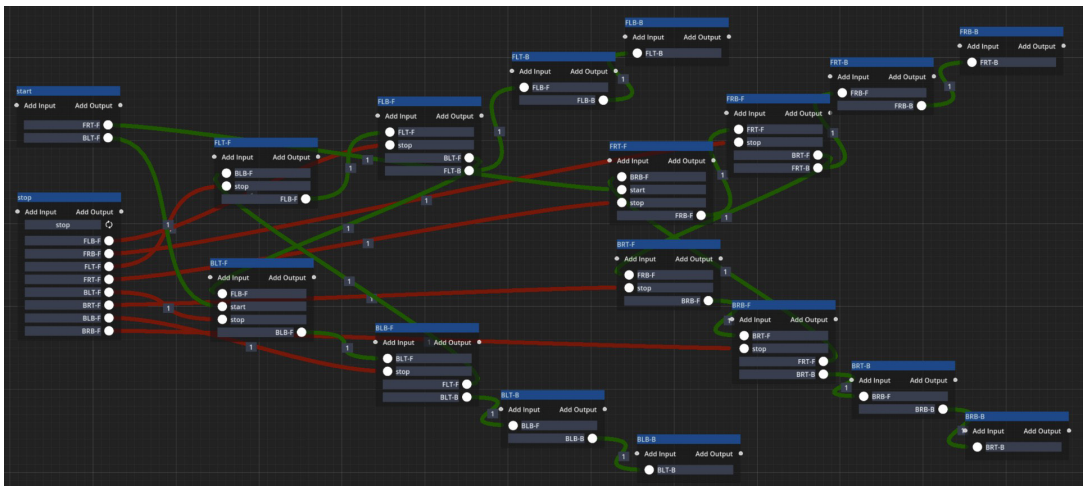


Fig. 3. CPG circuit representation in FEAGI environment showing the neuron inputs and outputs.

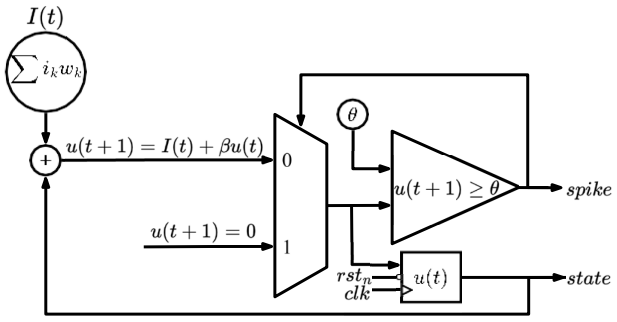


Fig. 4. Modified Leaky Integrate and Fire Circuit Diagram.

Kelly platform offers notable flexibility, particularly through its Python API and Verilog modules, which streamline serial communication, deployment, and testing of our CPG model. The network is uniquely fitted for an FPGA. As mentioned, the modified LIF neuron has been tailored to reduce computational resource demands.

Moreover, our architecture leverages a modular design methodology that allows for minimal additional hardware to operate well. Essentially, the top-level model encapsulates all 18 neurons with input, output, and internal connections. This intention allows the network to be rewired if a different gait pattern is desired. Finally, this design framework paves the way for the potential development of a script that automates the generation of key network topologies, thereby reducing the need for direct implementation in hardware description languages such as Verilog or VHDL.

For deployment on an FPGA, our design includes a counter module that steps down the internal clock on the development board by a tunable scale. This counter is adjustable in real-time with a 32-bit threshold value that when reached releases a single clock pulse to the CPG. The XEM7310 board houses a 200MHz clock oscillator. So, with the counter, this network has an operating range from 5ns to roughly 21s. However, for a typical gait, the time-scale runs in the range of tens to hundreds of milliseconds.

C. Communication with the Petoi Bittle X

The Petoi Bittle X is a quadruped robotic pet built on the open-source OpenCat software framework [10]. We selected this platform for its accessibility and its active user base. The Bittle X is powered by an ESP32 microcontroller, which supports SPI protocol, USB-C, and Bluetooth connectivity. Furthermore, the OpenCat documentation provides a comprehensive list of serial commands and predefined functions for various gait patterns and actions.

To facilitate communication between the FPGA and the Petoi robot, we use a PC as an intermediary. The PC runs a Python script that translates the spike outputs from the FPGA into commands for the ESP32. Our network topology defines six states—four for gait patterns and two for starting and stopping movement. The PC periodically polls the FPGA for changes in the bitstream; if a change is detected, it encodes the new state into angles for the robot's eight servomotors (two per leg). These commands are then sent to the ESP32, which adjusts the servomotor angles accordingly. This setup is demonstrated in Fig. 5 where the FPGA and Bittle X are connected to the PC via USB-C SPI. Here, the FPGA is controlled over SPI by setting and resetting the reset pins, start and stop neurons. The output spikes are read using a Python script as a stream of 16 bit binary values and decoded into the spike-out values from each motor neuron.

IV. Results

A. Comparison of Platforms

To better understand the leverage FPGAs provide over traditional CPU based motor controllers, we must quantify this by implementing a similar SNN model implementing with a variable size. In doing so, we have designed a comparable model in software using Python with which we can compare the latency and utilization of SNNs on hardware and software.

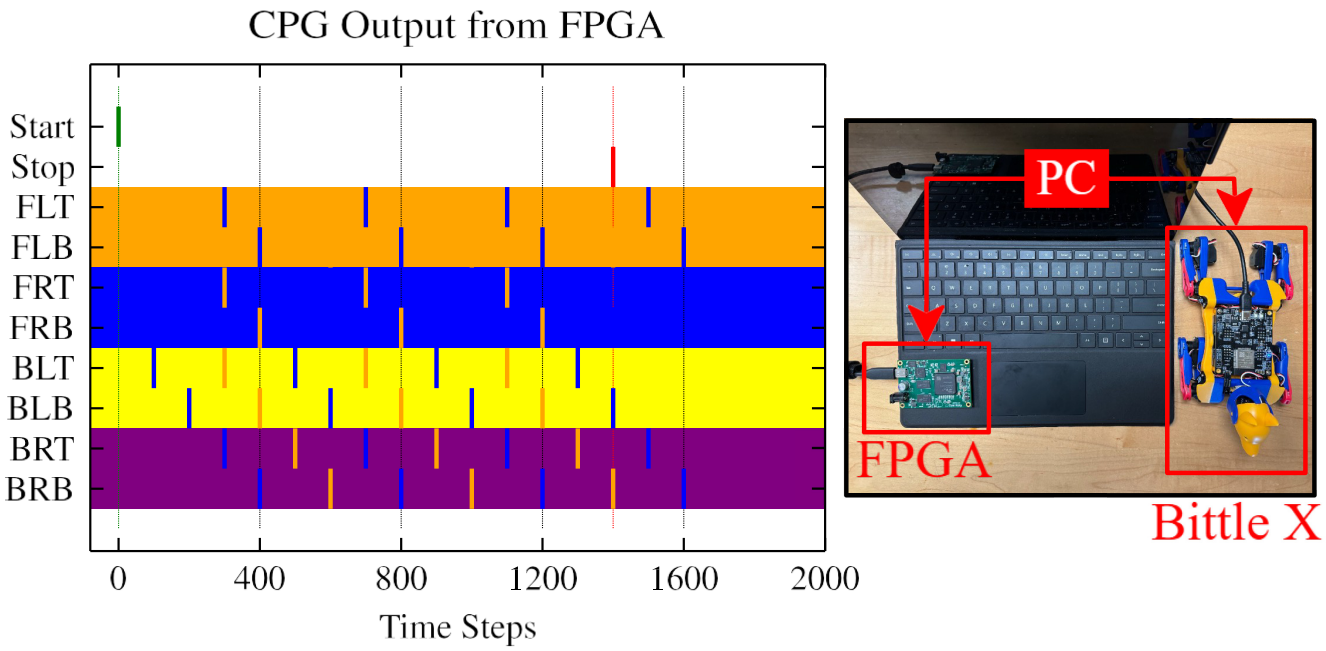


Fig. 5. Event plot from FPGA data stream and deployment setup. The naming and coloring conventions remain the same as in Fig. 2(b). The start neuron excites the network, initializing motion in the robot. This continues for multiple iterations of the gait denoted by dark verticle lines every 400 time stepsuntil the stop neuron inhibits the forward acting neurons. The residual spikes propagate through the network and motion stops.

TABLE I
COMPARISON OF CPG IMPLEMENTATIONS ON FPGA

Paper	Neuron Model	Neurons	Synapses	LUTs	FFs	Utilization	Deployed
Rostro-Gonzalez [2]	gIF	12	20	796	449	0.88%	✓
Ambroise [11]	IZH	8	14	1037	1093	1.50%	*X*
Chen [12]	Modified KK	2	2	27,872	533	20.0%	*X*
This Work	Modified LIF	18	30	813	989	1.30%	✓

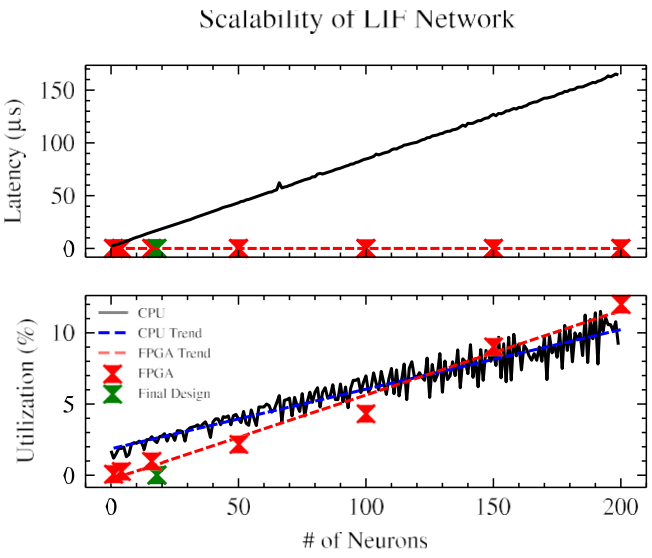


Fig. 6. Latency and utilization metrics on CPU and FPGA implementations

For valid readings, we employ the following test bench. We first sweep across a range of various model sizes. Then, we let the network run for 6000 time steps and measure the latency and CPU utilization for each step. To measure latency, we take the difference in time for all neurons to compute a step. For CPU utilization, we take the raw output of the Python function `psutil.cpu_percent()` and divide it by the number of CPU cores using `os.cpu_count()`. Finally, we get a mean over the entire runtime of the test. We do this for each model size to get a sweep between 1 to 200 neurons. In this exploration, we used a Surface Pro 9 with a *12th Gen Intel(R) Core(TM) i7-1255U*, with a 2600 Mhz clock, 10 cores, and 12 logical processors.

To gather the metrics from the FPGA implementation, we designed a scalable dummy SNN that generates a variable amount of LIF neuron modules in Verilog. Then, we run synthesis and implementation in Vivado without optimizations so as to allow the hardware and software models to be relatively equal in complexity. For increasing model size, we

select a subset of neuron counts between 1 and 200. To measure latency, we obverse the time in simulation to compute a single step for the network – noting the number of cycles – then, using the physical clock frequency, we have the latency of one clock cycle, with the latency being the clock period multiplied by the number of periods per time step. To measure hardware utilization, we take the sum of utilized look-up tables (LUTs) and flip-flops (FFs), often reported as registers, and divide it by the sum of each total. For the XEM7310-A75 development board, there are 47200 LUTs and 94400 FFs respectively.

The results of this can be visualized in Fig. 6. Here, the solid black lines denote the software metrics with the red X's and dotted lines for hardware measurements and trend lines, respectively. In both plots, the green x marker indicates the final post-optimization CPG design of 18 neurons. In the utilization plot, the blue dotted line shows the software trend.

As demonstrated in this test bench, we observe a clear trend that is forming. Most evidently, in terms of run-time, the FPGA implementation of the network starkly eclipses the software model at all timescales. This is expected as we can deduce from the logic for calculating the next-state values of each neuron. Within each module, the computations are entirely combinational, therefore occurring in a single clock cycle. Moreover, all neuron computations occur in parallel, resulting in the run-time of the model being $O(1)$ – solely dependent on the frequency of the physical clock on the FPGA.

In terms of resource utilization, we observe equally interesting behavior. Initially, hardware utilization outperforms software marginally. However, as the model increases in size, the software implementation tends to slightly outpace that of the hardware – roughly occurring at 125 neurons. This is critical to note that while both implementations' utilization scale linearly in terms of model size, they do not scale proportional to each other. We must also consider that the CPU benchmark is done on an exceptional personal computer, so these results may vary slightly in terms of empirical data whether done on a dedicated system or not; however, the linear trend would remain the same across all CPU implementations. Moreover, this does not account for the optimizations from synthesis and implementation that are disabled in this benchmark. For example, the final 18 neuron model slightly outperforms the 16 neuron model without optimizations, so one could surmise that similar reductions would propagate as the model increased in size.

B. Comparison of Works

We show a comparison among works in Table. I. Here, we calculate utilization in terms of the total logical components in the FPGA in this work. In [2], researchers implement a generalized Integrate-and-Fire model(gIF) in a 12 neuron, 20 synapse CPG network for Hexapod robot and report a usage of 796 look-up tables and 449 flip-flops. While this work's model is noticeably larger, at 18 neurons and 30 synapses, we manage to perform equivalently in terms of resources. Other works aim to implement more bio-accurate neuron models. In [11], [12] researchers implement an Izhikevich (IZH) and modified Komendantov–Kononenko (KK) model respectively. Each provides more realistic biomechanical behavior – aiming to implement bursting neuron models. However, they use significantly more resources for the FPGA, limiting their scalability to larger models and deployment to hardware.

In addition to computational resources, we examined the timing restraints of the network when communicating over a serial connection. To do this, we modified the CPG clock speed in real time by modifying the count threshold. We swept from a latency of 168ms (6Hz) down to 5.25ms (200Hz) without noticing any delay in command execution. This benchmark was not noted in any of the mentioned works, but it is crucial as the obvious bottleneck of the network would be the limitations of communication between the FPGA and servomotors. A video demonstration of this test is available at https://youtu.be/gxWJllk5rwg.

V. Conclusion

In this work, we have presented a process to design, implement, and deploy a Central Pattern Generator network for bipedal locomotion using an accessible connectome model and open-source tools. By leveraging the FEAGI interface and the Petoi Bittle X platform—both widely available and accessible to general consumers—we aim to lower the entry barrier for software engineers interested in exploring the field of robotics and neuromorphic systems.

Our model is designed to be easily scalable, accommodating larger networks and allowing for adjustable gaits and speeds, which provides a flexible foundation for more complex locomotion tasks. Additionally, we utilize a modified Leaky Integrate-and-Fire neuron model with a reduced digital footprint, optimized specifically for FPGA hardware deployment. This approach achieves a higher level of performance and adaptability than is typically feasible with a consumer-grade microcontroller, positioning this system as an ideal platform for developers looking to explore robotics at a low level.

Through rigorous technical benchmarking, we have proven that this model consistently outperforms or keeps pace with current solutions on both hardware and software. This is a result of the unique characteristic of trading an increased utilization for higher throughput via parallelization native to FPGA architecture.

Beyond technical achievements, our work emphasizes the potential of modular neuron-level control, empowering developers to experiment with custom connectome designs. The integration of open-source tools like FEAGI and the Petoi robot offers unprecedented opportunities to open access to advanced robotics concepts and to inspire innovation in neuromorphic engineering, where networks can be configured and adapted for varied applications in both research and industry.

Overall, this work not only showcases the feasibility of deploying complex neural models on accessible hardware but also highlights the potential for collaborative, open-source approaches in the advancement of robotics and neuroscience.

REFERENCES

[1] Taryn Klarner and E Paul Zehr. Sherlock holmes and the curious case of the human locomotor central pattern generator, Jul 2018.

[2] H. Rostro-Gonzalez, P.A. Cerna-Garcia, G. Trejo-Caballero, C.H. Garcia-Capulin, M.A. Ibarra-Manzano, J.G. Avina-Cervantes, and C. Torres-Huitzil. A cpg system based on spiking neurons for hexapod robot locomotion. *Neurocomputing*, 170:47–54, 2015. Advances on Biological Rhythmic Pattern Generation: Experiments, Algorithms and Applications Selected Papers from the 2013 International Conference on Intelligence Science and Big Data Engineering (IScIDE 2013) Computational Energy Management in Smart Grids.

[3] Vijay Shankaran Vivekanand, Samarth Chopra, Shahin Hashemkhani, and Rajkumar Chinnakonda Kubendran. Robot locomotion through tunable bursting rhythms using efficient bio-mimetic neural networks on loihi and arduino platforms. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, ICONS '23, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Sijia Lu and Feng Xu. Linear leaky-integrate-and-fire neuron model based spiking neural networks and its mapping relationship to deep neural networks. *Frontiers in Neuroscience*, August 2022.

[5] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics from single neurons to networks and models of cognition Wulfram Gerstner; Werner M. Kistler; Richard Naud; Liam Paninski*. Cambridge University Press, 2016.

[6] R. Jacob Vogelstein, Francesco Tenore, Ralph Etienne-Cummings, M. Anthony Lewis, and Avis H. Cohen. Dynamic control of the central pattern generator for locomotion. *Biological Cybernetics*, 95(6):555–566, Nov 2006.

[7] Sayantan Auddy, Sven Magg, and Stefan Wermter. Hierarchical control for bipedal locomotion using central pattern generators and neural networks. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 13–18, 2019.

[8] Mohammad Nadji-Tehrani and Ali Eslami. A brain-inspired framework for evolutionary artificial general intelligence. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–15, 03 2020.

[9] Mohammad Nadji, Kakcalu13, Eric Miller, Amir-Rasteg, Sabra, NeuravilleDeveloper, dependabot[bot], Chloe Chen, Dante Smith, Kat, github actions[bot], jenz0512, and pradeeppathak9. *feagi/feagi*. 10 2024.

[10] Rongzhong Li, Jason.W, Psycho, LucasYFL, Zehui ZHAO, RongzhongLi, cc, Friende, Kai Zhen Mai, atzlinux, Hoani Bryson, sidpitt89, and tym999. *PetoiCamp/OpenCat*. 9 2024.

[11] Matthieu Ambroise, Timothe´e Levi, Se´bastien Joucla, Blaise Yvert, and Sylvain Sa¨ıghi. Real-time biomimetic central pattern generators in an fpga for hybrid experiments. *Frontiers in Neuroscience*, 7, 2013.

[12] Qi Chen, Jiang Wang, Shuangming Yang, Yingmei Qin, Bin Deng, and Xile Wei. A real-time fpga implementation of a biologically inspired central pattern generator network. *Neurocomputing*, 244:63–80, 2017.