POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

FPGA Implementation of Izhikevich Neuron Model for Signal-to-Spike Encoding

Relatori
Gianvito Urgese
Michelangelo Barocci

Candidato
Shakil Mahmud Boby

October 2025

Abstract

The growing popularity of neuromorphic computing in edge and robotic applications leads to new necessities in terms of hardware solutions that can encode real measurements into spike trains compatible with the execution of biologically plausible Spiking Neural Networks (SNNs).

This thesis presents a complete Field-Programmable Gate Array (FPGA) implementation of the Izhikevich neuron model, which is specifically designed to be used for signal-to-spike encoding purposes. Unlike existing resource-optimized implementations that sacrifice biological fidelity through mathematical approximations, this work demonstrates a complete, flexible, modular Izhikevich model that can be implemented on an FPGA through a novel pipeline architecture design.

The hardware implementation of the Izhikevich model on PYNQ-Z2 reconfigurable board has a 4-stage pipelined architecture with circular buffer management. It uses Q5.11 fixed-point arithmetic with a biologically-accurate time step of 0.25 ms. The single-core neuron module consumes moderate resources while maintaining suitable computational efficiency. The FPGA as a whole links the neuron core with the ZYNQ processing system architecture through AXI4-Lite interfaces that facilitate the real-time control capabilities of the parameters through Jupyter notebooks. Dual BRAM controllers make data retrieval and logging easier. The implementation demonstrates consistent behavioral accuracy through comprehensive verification across Python simulation, Vivado behavioral simulation, and hardware execution levels. The adopted model is capable of replicating all 20 original Izhikevich spiking patterns, such as tonic spiking, bursting behavior, and more complex dynamics like bistability and inhibition-induced spiking that warrant the biological validation of the model. The modular architecture supports both parallel multi-neuron implementations and time-multiplexed techniques when resources are limited.

To demonstrate the practicality, a subset of real-world inertial measurement unit (IMU) sensor data from the WISDM dataset is used as input for both single and multi-neuron spiking encoders. These encoders are tasked with encoding six-channel sensor streams into activity-specific spike patterns. This validation relates laboratory neuron models to real-world applications, showing that sensor data can be processed in real time by the implementation in neuromorphic IoT systems.

The performance analysis indicates that the implementation easily achieves a 4kHz processing rate per neuron for accurate biological neuron modeling. Also, frequency scaling analysis establishes the maximum processing speed that various neuron configurations can achieve. This is essential for understanding the performance limits of the design and optimizing multi-neuron implementations for various application requirements. The resource consumption analysis demonstrates that the resource has the proper scaling, i.e., 10 parallel neurons can be effectively integrated with an FPGA resource utilization of around 50%.

This study develops a complete signal-to-spike encoding system with biologically accurate Izhikevich neurons on an FPGA to mainly emphasize research-grade systems capable of practical deployment. The transition from biological validation to hardware implementation and then to real-world signal encoding is a significant step in the research of neuromorphic computing, which creates a foundation for the high-level applications of spiking neural networks in diverse signal processing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid increase in the number of Internet of Things (IoT) devices and edge computing applications has resulted in a critical need to develop energy-efficient and real-time signal processing systems that can operate on a limited amount of power and provide high computational accuracy at the same time [1]. Current computing systems are mainly based on the von Neumann architecture, where processing and memory components are physically separated. This physical isolation causes the so-called "von Neumann bottleneck", in which data must be continuously transferred between memory and processor, resulting in limited computational performance with massive energy consumption [2]. On the contrary, biological neural systems have demonstrated impressive computational efficiency through distributed processing in which memory and computation co-exist in the same physical medium. The human brain carries out complicated cognitive functions using just 20 watts of power, whereas similar artificial intelligence systems use megawatts. The resulting sharp efficiency disparity has inspired the design of neuromorphic computing – an engineering framework that aspires to the architecture and operational principles of biological nervous systems [3][4].

Neuromorphic computing systems use Spiking Neural Networks (SNNs) as their basic model of computation [5]. In contrast to traditional artificial neural networks that receive continuous-valued activations, SNNs receive discrete time events known as spikes, which resemble the transmission of information via action potentials by biological neurons [6]. This paradigm of computation as an event-driven process has several benefits. First, spike communication is sparse, thus lowering power consumption because inactive neurons use little power [7]. Second, rich information is represented by temporal encoding, where spike timing carries meaning beyond simple firing rates [8]. Third, SNNs are inherently connected to neuromorphic sensors, including dynamic vision sensors and silicon cochleas, and directly provide event-based data [9].

Nevertheless, deploying SNNs for real-world applications requires converting analog sensor signals into spike trains. The signal-to-spike encoding transfers

continuous-valued time-series sensor data (e.g., accelerometers, gyroscopes, and temperature sensors) into temporal spike patterns that can be processed by SNNs [10]. There are two major encoding strategies. First, the rate-based encoding algorithms encode signal amplitude into firing rate, while ignoring the temporal structure, which is relatively simple to implement [11]. Second, the temporal encoding schemes maintain accurate timing information, as the features of signals are represented by spike intervals, delays, or burst patterns [12]. However, the neuron model-based encoding is a potential intermediate candidate, in which biologically motivated neuron dynamics naturally convert signals into spike trains while preserving temporal structure [13].

From the basic integrate-and-fire model to the advanced Hodgkin-Huxley model, the Izhikevich neuron model offers the optimal balance for hardware implementation [14]. This model, developed by Eugene Izhikevich in 2003, utilizes only two coupled differential equations to capture the diverse firing patterns observed in biological cortical neurons [15]. This model is capable of producing 20 different spiking behaviors, including tonic spiking, bursting, adaptation, and resonance, with only simple parameter adjustments [15], [16]. This simplicity in computation, coupled with its biological richness, makes it the best choice for implementation in hardware, where the complexity of a circuit directly influences resource consumption and power efficiency.

There are key architectural choices that must be made when implementing neuron models in physical hardware. These options encompass analog, digital, and mixed-signal approaches [17]. Analog implementations use CMOS circuits that can achieve exceptional power efficiency by directly mapping the equations of neurons to circuit dynamics. However, they face problems with parameter variability due to variations in the manufacturing process, limited configurability as parameters are hardwired into circuit designs, and scalability issues since analog neurons occupy large areas on the die [18]. Digital implementations provide accurate control of parameters, high scalability due to the sharing of resources, and are immune to analog noise; however, they conventionally demand more power, as they must run continuously on a clock [19]. The mixed-signal methods strive to have the benefits of both paradigms, but add complexity to analog-digital interfacing and calibration [20].

In digital implementation, there are three fundamental choices, namely custom neuromorphic chips, Application-Specific Integrated Circuits (ASICs), and Field-Programmable Gate Arrays (FPGAs). Neuromorphic chips, such as IBM TrueNorth and Intel Loihi, offer optimized spiking neural network architectures with in-built learning features [21], [22]. They are, however, restricted to predefined neuron models and are costly for small-scale research [23]. ASIC implementations outperform FPGAs in terms of maximum performance and energy efficiency when used in high-volume production; however, they are expensive to set up and offer no reconfigurability after fabrication. FPGAs are a compelling option, offering

a reconfigurable fabric that enables refinement of the design through an iterative process, moderate development costs, accessible development tools, and adequate performance to run real-time neural simulations [24].

Several studies in the literature have implemented different neuron models on FPGA hardware. The Morris-Lecar, Izhikevich, and Hodgkin-Huxley models have been realized on FPGAs in work [25]. Grassia et al. [26] implemented a silicon neuron based on the Quartic model using an FPGA. Moreover, several studies [27], [28], [29], [30], [31], [32], [33], [34], [35], [36] have implemented the Izhikevich neuron model on various FPGA platforms. In many of these works, the original Izhikevich model was modified to make the model simpler to implement on hardware, typically by eliminating the quadratic term. Some studies employed power-of-two methods, while others used the CORDIC algorithm, piecewise linear approximations, or stochastic logic. The primary goal of these modifications was to lower the hardware cost. However, such simplifications often compromise the dynamic behavior of the original model, limiting the ability to reproduce all 20 spiking patterns in hardware. Although some studies report a maximum number of spiking patterns, their results are mainly based on simulation. None of the hardware implementations has demonstrated all 20 spiking patterns with a perfect match with the simulation results. Furthermore, some modified models also exhibit reduced accuracy in some instances.

In this research, we implement the Izhikevich neuron model, maintaining its originality, using the PYNQ Z2 development board with a Zynq SoC. The Xilinx Zynq SoC family combines ARM processors with FPGA fabric, allowing for hardware-software co-design. This enables the implementation of complex control and data logging on processors, while computationally expensive neural dynamics are implemented in parallel on the hardware [37]. The fundamental objectives of this thesis are as follows:

- The first objective is to design, implement, and validate an FPGA realization of the original Izhikevich neuron model. This involves developing a Verilog hardware description that maintains a balance between computational accuracy and resource consumption. Specific goals include: selecting an appropriate fixed-point representation that minimizes bit width while maintaining sub-millivolt accuracy in membrane potential computation; implementing numerical integration using forward Euler discretization with appropriate timestep selection to ensure stability and accuracy; validating implementation correctness through three-tier comparison: Python floating-point software simulation, Vivado behavioral simulation, and physical FPGA hardware deployment; characterizing resource utilization (LUTs, flip-flops, DSP blocks, BRAM), timing performance (maximum operating frequency), and power consumption (static, dynamic, total on-chip power).

- The second objective extends the single-neuron design to scalable multi-neuron systems through two complementary approaches. Firstly, a time-multiplexing approach enables the implementation of large virtual neuron populations by sequentially reusing a single hardware neuron module across multiple timesteps. This approach maximizes neuron count within fixed resources but introduces latency, as not all neurons can update simultaneously. Second approach, parallel implementation that instantiates multiple independent neuron cores operating simultaneously, enabling true real-time processing where all neurons evolve concurrently. This approach sacrifices maximum neuron count for reduced latency and simplified control.

- The third objective demonstrates practical applicability by encoding realistic Inertial Measurement Unit (IMU) sensor data into spike trains using neuron model-based encoding. Using the WISDM (Wireless Sensor Data Mining) dataset [38], [39] containing 6-axis IMU data (3-axis accelerometer, 3-axis gyroscope) recorded during human activities, to implement: multi-channel encoding utilizing parallel neuron cores for simultaneous processing of all six sensor axes; temporal interpolation to bridge the mismatch between sensor sampling rates (20 Hz typical) and neural timesteps (4 kHz implementation frequency).

- The overall objective establishes a complete development and validation framework integrating FPGA hardware acceleration with software control and interfacing. Leveraging the Xilinx Zynq SoC architecture, combining ARM processors with FPGA fabric, to implement: AXI communication protocols for bidirectional data transfer between Processing System (PS) and Programmable Logic (PL); Python-based control software using the PYNQ framework for high-level experiment management, parameter configuration, and results visualization; create pathways to enable sensor signal dataset input and spike train output.

This thesis is divided into five chapters that outline the entire research process, from foundational concepts to implementation validation. The rest of the parts are organised as follows:

Chapter 2 provides the necessary theoretical backgrounds of both biological neuroscience and hardware implementation platforms. The basics of neurons (structure and the formation of action potentials) are presented in Section 2.1. Section 2.2 systematically compares neuron models (such as Hodgkin-Huxley, integrate-and-fire variants, FitzHugh-Nagumo, Morris-Lecar, Hindmarsh-Rose, Adaptive exponential integrate-and-fire, and Izhikevich) and provides mathematical formulations as well as analysis of hardware implementation complexity, which justifies the use of Izhikevich models. The biological significance of 20 most important neurocomputational features of spiking neuron is discussed in Section 2.3. Section 2.4 presents FPGA

technology, Xilinx Zynq SoC architecture and the PYNQ Z2 development board that was used in this work.

Chapter 3 explains the experimental platform and implementation methodology. It involves the discretization of the Izhikevich neuron model (with the forward Euler method of time stepping, a time step of 0.25 ms), the choice of a fixed-point arithmetic representation (Q5.11 format), the architecture of a four-stage pipeline hardware implementation, and the block diagram and co-design architecture of the overall design.

Chapter 4 delivers detailed implementation and experimental findings. Section 4.1 justifies the single-neuron implementation through an analysis of resource utilization, timing aspects, power consumption, fixed-point accuracy, and systematic reproduction of all 20 spiking patterns. Section 4.2 describes multi-neuron scalability using the time-multiplexed virtual neuron architecture, which illustrates the scalability of resources by implementing multiple virtual neurons on a single core of physical hardware. Section 4.3 discusses parallel multi-neuron implementations designed for high throughput. Section 4.4 presents a comparison of resource consumption across all implemented architectures. Section 4.5 of the paper presents a frequency-dependent performance analysis of single-neuron and 4-neuron cores operating at frequencies ranging from 5 MHz to 60 MHz. Section 4.6 illustrates the encoding of IMU signals with the WISDM dataset and demonstrates that the temporal patterns of acceleration and gyroscopes spike trains are maintained across different physical activities.

Chapter 5 summarizes the research contributions, discusses the limitations of the current implementation, and suggests future research directions.

Chapter 2

Background

2.1 Neuron Fundamentals

The nervous system processes information through neuron networks. These are specialized cells that receive input signals, change their internal electrical state, and send out the output signals to other neurons through action potentials (spikes). Although the number of various types of neurons is very large, the majority of them have a similar structure: the cell body (soma) houses the nucleus; dendrites receive input signals; and an axon transmits electrical impulses to other neurons. Synapses act as a connector between neurons and transform incoming spikes into electrical currents that alter the voltage of the receiving neuron [16], [40], [41], [42]. This simple neuronal topology is depicted in Fig. 2.1, where two interconnected pyramidal neurons are represented with their typical dendritic trees, cell bodies, and axonal projections.

When a neuron is at rest, the cell membrane maintains an electrical charge difference between inside and outside of the cell, producing a membrane potential that typically ranges from $-80\,\mathrm{mV}$ to $-60\,\mathrm{mV}$, as illustrated by the resting state ($-70\,\mathrm{mV}$) in Fig. 2.2. The membrane acts as a leaky capacitor: the capacitance is provided by the cell membrane, whereas the ion channels serve as a voltage-gated switch that can permit current inflow and outflow. When a neuron is stimulated by injected current, either from internal synaptic inputs (as shown in Fig. 2.1) or externally, the voltage across the membrane varies. A positive current increases the voltage (depolarization), whereas a negative current decreases it (repolarization, hyperpolarization) [14], [41], [42].

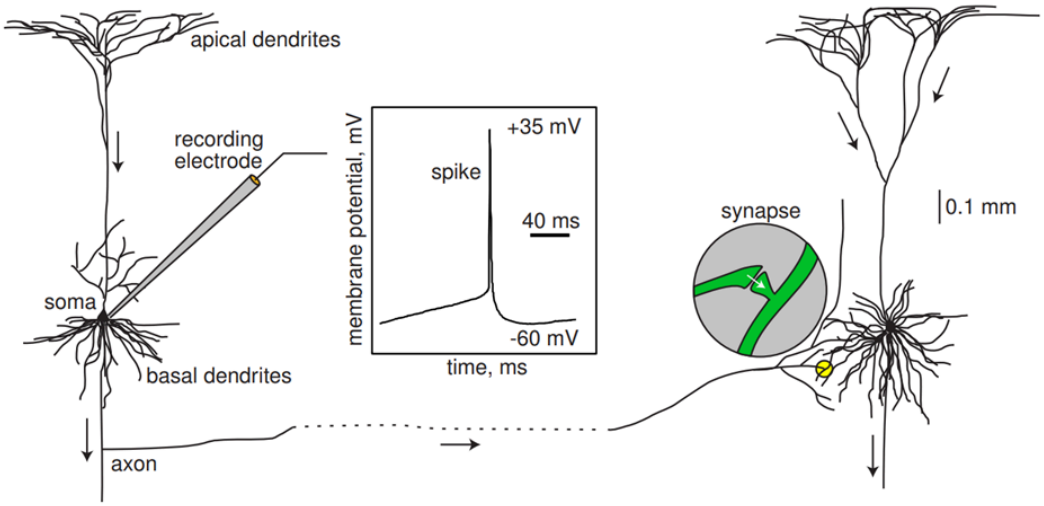


Figure 2.1: Two interconnected cortical pyramidal neurons showing basic neuronal anatomy, including soma, dendrites, axon, and synapses. The inset shows a typical recorded action potential with characteristic spike shape reaching +35 mV [16].

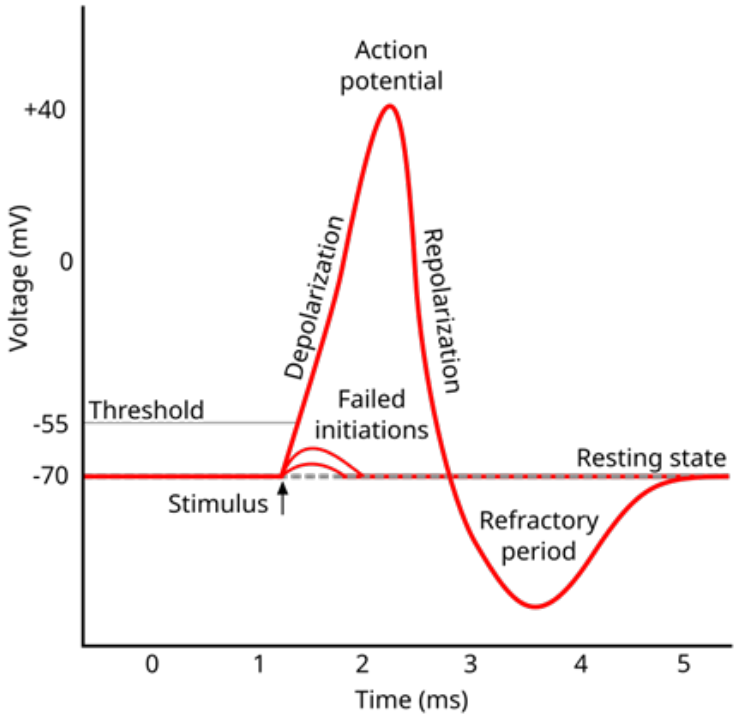


Figure 2.2: Action potential waveform showing the complete spike cycle, including stimulus, threshold crossing, depolarization phase, repolarization, refractory period, and return to resting state. Failed sub-threshold responses are also shown [43].

As the voltage becomes large enough to open the voltage-gated sodium channels, the neuron discharges an action potential (spike). This process occurs at a highly rapid rate: sodium enters rapidly, and the voltage surges to (+40 mV). The potassium channels are then opened, bringing the voltage back down, usually dropping below the resting level for a short time. This creates the typical spike shape shown in the inset of Fig. 2.1 and detailed in Fig. 2.2. After firing, there is a refractory period during which the neuron cannot fire again immediately, which actually limits the rate at which spikes can occur. The firing threshold does not remain constant and varies according to the recent activity of the neuron and its current condition [15], [41], [44], [45]. These phases of the action potential are clearly shown in Fig. 2.2, including the stimulus that causes the firing, the rapid depolarization phase, repolarization, and the refractory period.

The balance of current governs the basic electrical behavior of a neuron. If C is the membrane capacitance and $V(t)$ is the voltage across the membrane, with currents from ion channels $I_{\text{ion}}(V,t)$, synaptic inputs $I_{\text{syn}}(t)$, and external stimulus $I_{\text{ext}}(t)$, then Kirchhoff's current law gives:

$$C\,\frac{dV}{dt} = -I_{\text{ion}}(V,t) - I_{\text{syn}}(t) + I_{\text{ext}}(t) \tag{2.1}$$

Different neuron models define ionic currents (I_{ion}) and spike-generation rules differently, including how the membrane voltage resets after a spike. This combination of gradual voltage dynamics with sudden spike events forms the basis for many neuron models and is particularly useful for converting continuous signals into spike trains. In signal encoding, two factors are most important: (i) how input signals influence output, in terms of spike rate and timing, and (ii) how precisely spike timing is controlled by membrane properties, adaptation, and refractory mechanisms. These properties determine how accurately a neuron's spikes can represent features of an input signal, such as amplitude, frequency, or envelope [14], [41], [42].

The neurons use short spikes to communicate, but the time of the spikes may contain various pieces of information. In rate coding, the mean rate of spikes per second is used to describe the strength of the input signal. In temporal coding, the timing of spikes (when they occur, gaps between spikes, phase relative to other signals) contains more information. In population coding, groups of neurons work together with coordinated spike patterns to represent complex signals. A combination of these methods is used in real neural circuits [46], [47]. With hardware signal-to-spike encoders, we typically excite the neuron model with the injected input signal as a current and utilize the natural behavior of the neuron to encode signal features into spikes with the desired rate, timing, or phase relationships [6], [41], [42].

2.2 Neuron Models

Various mathematical models have been formulated to explain the workings of neurons, ranging from complex biophysical models to the simplest computational models. In this case, we are interested in models capable of simulating realistic spiking behavior and serving as signal-to-spike encoders, which can be executed on FPGA hardware. Some notable models, along with their mathematical equations and effectiveness as encoders, are described in the following subsections.

2.2.1 Hodgkin-Huxley (HH) model

The Hodgkin-Huxley model is the most detailed descriptive model of action potential generation. It was first developed to describe the electrical behavior of the giant axon in squid. The model explains the membrane current between voltage-gated sodium channels and potassium channels, as well as a leak current [44].

The voltage equation is:

$$C\,\frac{dV}{dt} = -g_{\mathrm{Na}}\,m^3 h\,(V - E_{\mathrm{Na}}) - g_{\mathrm{K}}\,n^4\,(V - E_{\mathrm{K}}) - g_{\mathrm{L}}\,(V - E_{\mathrm{L}}) + I(t) \tag{2.2}$$

where C is the membrane capacitance (typically $1\,\mu\mathrm{F\,cm^{-2}}$), V represents the membrane voltage, g_{Na}, g_{K}, g_{L} are the maximum conductances for sodium, potassium, and leak channels, respectively, E_{Na}, E_{K}, E_{L} are the reversal potentials (typically $50\,\mathrm{mV}$, $-77\,\mathrm{mV}$, and $-54.4\,\mathrm{mV}$, respectively), m, h, n are gating variables (between 0 and 1), and $I(t)$ is the injected current.

The gating variables control how the ion channels open and close:

$$\frac{dm}{dt} = \alpha_m(V)(1 - m) - \beta_m(V)m \tag{2.3}$$

$$\frac{dh}{dt} = \alpha_h(V)(1 - h) - \beta_h(V)h \tag{2.4}$$

$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V)n \tag{2.5}$$

The rate functions α and β depend on voltage in a complex way and are given by the complete set of equations:

$$\alpha_m = 0.1(V + 25)/\left(\exp\frac{V + 25}{10} - 1\right) \tag{2.6}$$

$$\beta_m = 4\exp(V/18) \tag{2.7}$$

$$\alpha_h = 0.07 \exp(V/20) \tag{2.8}$$

$$\beta_h = 1/\left(\exp\frac{V+30}{10} + 1\right) \tag{2.9}$$

$$\alpha_n = 0.01(V+10)/\left(\exp\frac{V+10}{10} - 1\right) \tag{2.10}$$

$$\beta_n = 0.125 \exp(V/80) \tag{2.11}$$

The HH model is capable of simulating extremely realistic spike shapes and many detailed behaviors of actual neurons. Being an encoder, it provides an exact representation of biological behavior and may support complex timing codes. It, however, requires solving four differential equations with complex exponential functions, which are computationally expensive and challenging to execute effectively on FPGA hardware [41], [42], [44], [45].

2.2.2 Leaky integrate-and-fire (LIF) model

The LIF model treats the membrane as a simple RC circuit with a spike threshold and reset mechanism. This is much simpler than HH but still captures the basic integrate-and-fire behavior. The voltage equation is [6], [48]:

$$C\frac{dV}{dt} = -g_L(V - E_L) + I(t) \tag{2.12}$$

with the spike condition:

$$\text{if } V \geq V_{\text{th}} \text{ then spike occurs, } V \leftarrow V_r, \text{ and } t \leftarrow t + t_{\text{ref}} \tag{2.13}$$

The parameters include the membrane capacitance C, the leak conductance g_L, and the leak reversal potential E_L, which acts as the resting potential. The spike threshold V_{th} is typically $-55\,\text{mV}$, while the reset potential V_r is usually $-70\,\text{mV}$. The refractory period t_{ref} normally lasts 1 ms to 2 ms.

The membrane time constant is:

$$\tau_m = \frac{C}{g_L} \tag{2.14}$$

which determines how quickly the voltage responds to input current. For a constant input I_0, the steady-state voltage is:

$$V_\infty = E_L + \frac{I_0}{g_L} \tag{2.15}$$

The firing rate for constant current follows:

$$f = \frac{1}{t_{\text{ref}} + \tau_m \ln\left(\frac{V_{\text{th}} - E_L - I_0 R_m}{V_r - E_L - I_0 R_m}\right)} \tag{2.16}$$

where $R_m = \frac{1}{g_L}$ is the membrane resistance.

LIF has the ability to encode rate and timing information despite its simplicity. It is highly suitable to hardware since it needs only simple arithmetic operations and can be easily implemented using fixed-point numbers. But as an encoder, it has limitations: the frequency-current relationship is too rigid without adaptation, and it cannot generate the range of firing patterns (such as bursting) that occur in real neurons [6], [14], [41], [42], [48].

The quadratic integrate-and-fire (QIF) variant replaces the linear leak with a quadratic term [6]:

$$\tau_m \frac{dV}{dt} = (V - V_{\text{rest}})(V - V_{\text{th}}) + R_m I(t) \tag{2.17}$$

This provides a more realistic spike initiation while remaining computationally efficient.

2.2.3 FitzHugh-Nagumo (FHN) model

The FHN model simplifies the complex HH model into a two-dimensional system that still captures the essential excitable behavior. It uses a fast variable v (like voltage) and a slow recovery variable w [49], [50]:

$$\frac{dv}{dt} = v - \frac{v^3}{3} - w + I(t) \tag{2.18}$$

$$\tau \frac{dw}{dt} = v + a - b\,w \tag{2.19}$$

where v and w are the fast variable (similar to membrane voltage) and the slow recovery variable, τ represents the time constant ratio (typically 10–20), and a, b are parameters that control excitability and oscillations.

The cubic term $v - \frac{v^3}{3}$ creates the characteristic N-shaped nullcline that produces excitable behavior. The parameter a controls the resting state, while b affects the recovery dynamics. For different parameter values, the model can show stable rest (no oscillations), oscillatory behavior (repetitive firing) and bistable behavior (two stable states).

The FHN model can reproduce basic excitable behavior including threshold effects, refractory periods, and repetitive firing. As an encoder, it provides richer dynamics than LIF while being much simpler than HH. However, implementing the cubic nonlinearity $\frac{v^3}{3}$ on FPGA requires careful design, and the parameter values need to be chosen carefully to ensure stable behavior with fixed-point.

2.2.4 Morris-Lecar (ML) model

The Morris–Lecar model is a two-dimensional conductance-based model originally developed for muscle fibers but widely used for neurons. It combines biological realism with computational efficiency [51]:

$$C\,\frac{dV}{dt} = -g_{\mathrm{Ca}}\,M_\infty(V)(V - E_{\mathrm{Ca}}) - g_{\mathrm{K}}\,w(V - E_{\mathrm{K}}) - g_L(V - E_L) + I(t) \tag{2.20}$$

$$\frac{dw}{dt} = \phi\,\frac{W_\infty(V) - w}{\tau_w(V)} \tag{2.21}$$

The steady-state functions are:

$$M_\infty(V) = \frac{1}{2}\left[1 + \tanh\left(\frac{V - V_1}{V_2}\right)\right] \tag{2.22}$$

$$W_\infty(V) = \frac{1}{2}\left[1 + \tanh\left(\frac{V - V_3}{V_4}\right)\right] \tag{2.23}$$

$$\tau_w(V) = \frac{1}{\cosh\left(\frac{V - V_3}{2V_4}\right)} \tag{2.24}$$

The parameters include the maximum conductances g_{Ca}, g_{K}, and g_{L} for calcium, potassium, and leak channels, respectively, along with their corresponding reversal potentials E_{Ca}, E_{K}, and E_{L}. The calcium channel properties are controlled by V_1 and V_2, which represent the half-activation voltage and slope, respectively. Similarly, the potassium channel is characterized by V_3 and V_4 for half-activation voltage and slope. The time constant parameter ϕ affects the overall dynamics, while w represents the potassium channel activation variable.

The ML model can exhibit different types of excitability. In Type I excitability, oscillations start smoothly, whereas in Type II excitability, oscillations occur abruptly and with a fixed frequency. This is regulated by the values of the parameters, especially V_3 and V_4. Resonance behavior can also be observed in the model, where it responds most strongly to inputs at specific frequencies.

Being an encoder, the Morris-Lecar (ML) model is capable of supporting phase coding and frequency locking, which is why it can be applied in the process of encoding oscillatory signals. Nonlinear functions, however, M_∞, W_∞, and τ_w, contain hyperbolic functions that are difficult to execute efficiently on an FPGA, which may require either look-up tables or polynomial approximation [25].

2.2.5 Hindmarsh–Rose (HR) model

The Hindmarsh–Rose model extends the dimensionality to three variables to capture bursting behavior, which is common in many real neurons [52]:

$$\frac{dx}{dt} = y - ax^3 + bx^2 + I(t) - z \tag{2.25}$$

$$\frac{dy}{dt} = c - dx^2 - y \tag{2.26}$$

$$\frac{dz}{dt} = r\left(s(x - x_R) - z\right) \tag{2.27}$$

The parameters include x as the fast variable representing membrane potential, y as the fast recovery variable, and z as the slow adaptation current. The fast subsystem parameters a, b, c, and d typically take values of 1, 3, 1, and 5, respectively. The slow-fast time scale ratio r usually ranges from 0.001 to 0.01, while s represents the coupling strength between fast and slow variables. The reference level for the slow variable is denoted as x_R.

The dynamics operate on two separate time scales. The fast subsystem with the variables x and y generates individual spikes with a period of approximately 1–10 ms. Meanwhile, the slow variable z couples with bursting behavior with very long periods of nearly 100–1000 ms. The model is able to generate various firing patterns with different parameter values and input conditions. Tonic spiking occurs when z is small, characterized by regular spikes at regular intervals. Bursting behavior creates groups of spikes separated by quiet periods. The model can also exhibit chaotic behavior with irregular spike patterns.

As an encoder, HR is excellent for capturing envelope information and rhythmic structure through burst timing. The burst frequency can encode slow signal components while spike timing within bursts can encode faster features. However, the cubic and quadratic terms pose difficulties for fixed-point arithmetic in FPGA implementations, and the model is also parameter-sensitive. The piecewise-linear approximations might assist in decreasing the complexity of the computations and maintaining the necessary bursting behavior [52].

2.2.6 Adaptive exponential integrate-and-fire (AdEx) model

The AdEx model improves upon LIF by adding an exponential spike-generating current and an adaptation variable w [53]:

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\Big(\frac{V - V_T}{\Delta_T}\Big) - w + I(t) \tag{2.28}$$

$$\tau_w \frac{dw}{dt} = a(V - E_L) - w \tag{2.29}$$

with reset condition:

$$V \geq V_{\text{spike}} \Rightarrow \begin{cases} V \leftarrow V_r, \\ w \leftarrow w + b \end{cases} \tag{2.30}$$

The parameters include the membrane capacitance C, leak conductance g_L, and leak reversal potential E_L. The spike threshold V_T acts as a soft threshold, while the spike slope factor Δ_T typically ranges from 2–5 mV. The spike detection threshold V_{spike} is usually set to 0 mV, and V_r represents the reset potential. The adaptation time constant τ_w typically ranges from 100–500 ms, while parameter a controls sub-threshold adaptation coupling and b determines the spike-triggered adaptation increment.

The exponential term $g_L \, \Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right)$ provides realistic spike initiation. Unlike the LIF model's hard threshold, this creates a soft threshold where spikes develop gradually, similar to real neurons.

Adaptation current w introduces several important features that render the model biologically more realistic. Spike-frequency adaptation causes the firing rate to decrease when there is a constant input, a behavior similar to that of real neurons. Post-spike hyperpolarization causes the neuron to become temporarily less excitable. Bursting and irregular firing patterns can also be made by the model, depending on the parameter values. The various combinations of parameters give various firing patterns that can be adapted to particular applications. When b is small and a is positive, regular spiking occurs. Bursting behavior is experienced when a is negative and b is large. The values of intermediate parameters can produce irregular or chaotic firing patterns.

AdEx has a number of notable benefits in the case of encoding applications. The adaptive rate coding automatically adjusts to the signal statistics and is thus robust to different input conditions. Spike-frequency adaptation improves the encoding of non-stationary signals by preventing saturation during sustained inputs. The more realistic spike shapes are more precise in time than the simple ones. Additionally, AdEX offers a relatively easy implementation compared to full conductance-based models, along with rich dynamics. FPGA implementations of the exponential function can be done either via lookup tables or a set of polynomials. Thus, AdEx is more feasible than full HH models, while offering far richer dynamics than simple LIF model.

2.2.7 Izhikevich model

The Izhikevich model achieves an optimal balance between biological realism and computational efficiency. It couples a quadratic voltage equation to a recovery variable u with simple reset condition [14]:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I(t) \tag{2.31}$$

$$\frac{du}{dt} = a(bv - u) \tag{2.32}$$

with the reset condition:

$$\text{if } v \geq 30 \text{ mV, then } \begin{cases} v \leftarrow c, \\ u \leftarrow u + d \end{cases} \tag{2.33}$$

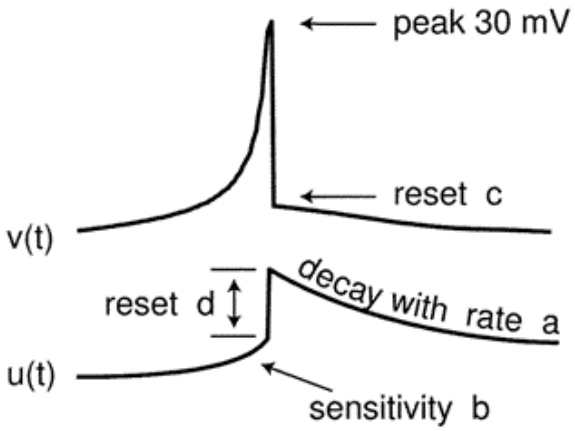


Figure 2.3: The Izhikevich model reset dynamics illustrate the interaction between the membrane potential $v(t)$ and the recovery variable $u(t)$. The figure highlights key model parameters: spike peak at 30 mV, reset value c for voltage, reset increment d for the recovery variable, decay rate controlled by parameter a, and sensitivity parameter b [14].

The parameters have clear biological meanings that make the model intuitive to use. The variable v represents the membrane potential in millivolts, while u is the recovery variable that represents the combined effects of potassium and sodium channel recovery. After the spike reaches its apex (+30 mV), the membrane voltage and the recovery variable are reset according to the equation (2.33). Synaptic currents or injected dc-currents are delivered via the variable I. The parameter a

controls the recovery time constant and typically ranges from 0.01 to 0.1, where smaller values of a lead to slower recovery. Parameter b determines the sensitivity of u to v and usually takes values between 0.2 and 0.25, with larger b values creating stronger coupling between the variables. The reset value c sets the membrane potential after a spike and typically ranges from $-65\,\mathrm{mV}$ to $-50\,\mathrm{mV}$. Finally, parameter d controls the reset increment for u after each spike, typically ranging from 2 to 8, where larger d values produce more substantial adaptation effects. Figure 2.3 illustrates the Izhikevich model's reset dynamics, showing the impact of different parameters.

From a mathematical perspective, the model's behavior can be understood through the analysis of nullclines. The v-nullcline is defined by $u = 0.04\,v^2 + 5\,v + 140 + I(t)$, while the u-nullcline follows the simpler linear relationship $u = b\,v$. The model exhibits a saddle-node bifurcation for spike initiation, similar to Type I neurons in the Morris-Lecar (ML) classification.

The Izhikevich model has a number of important benefits as an encoder. It is computationally inexpensive, with a single quadratic operation and straightforward arithmetic operations. The model is very dynamic and can reproduce more than 20 firing behaviors of a real neuron. Minor changes in parameter values may lead to qualitatively different behavior, providing a finer control over the encoding properties. The u variable offers automatic gain control through adaptation and is used to prevent saturation and dynamic range expansion. Above all, the model is most suitable for a fixed-point implementation on FPGA platforms in terms of hardware implementation.

In comparison to LIF, Izhikevich provides far richer dynamics, including adaptation, bursting, and resonance. Compared to HH/ML/HR, it is computationally lighter while still capturing the necessary neuron behaviors, making it ideal for high-throughput FPGA implementations [14], [15].

2.3 Features of Biological Spiking Neuron

Biological neurons display significant variability in firing patterns. These patterns arise from specific combinations of ion channel types and densities that create distinct dynamical behaviors. It's important to understand the biological significance of this diversity of spiking patterns. Izhikevich [14], [15] systemically categorized the 20 most prominent neurocomputational properties observed in cortical neurons through electrophysiological recordings, which are as follows:

Tonic Spiking: This is the regular and repeated firing of neurons resulting from a persistent input as shown in Fig. 2.4. It implements rate coding in which the strength of the stimulus is reflected in the firing rate. Tonic spiking is found biologically in three types of primary cortical neurons: regular spiking (RS) excitatory pyramidal cells, fast-spiking (FS) inhibitory interneurons, and low-threshold spiking (LTS) interneurons [54], [55]. The continuous firing encodes the magnitude of persistent inputs, which actually play an important role in representing steady stimuli and motor commands. Step current is frequently injected as a stimulus in neurophysiological experiments to investigate neuronal excitability and classify neurons based on their firing rate and adaptation properties [14], [15], [16].

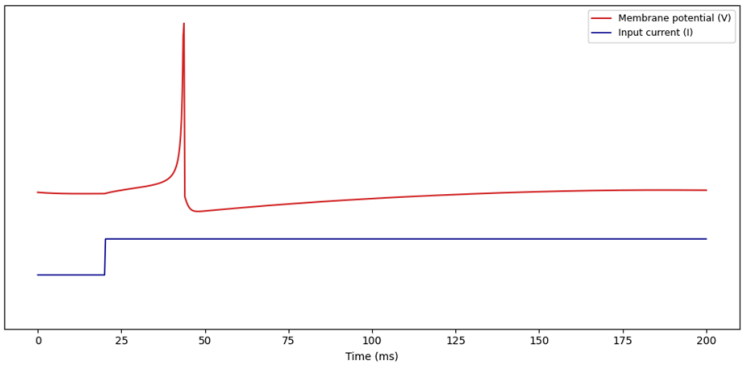


Figure 2.4: Tonic spiking neurocomputational property of biological spiking neurons.

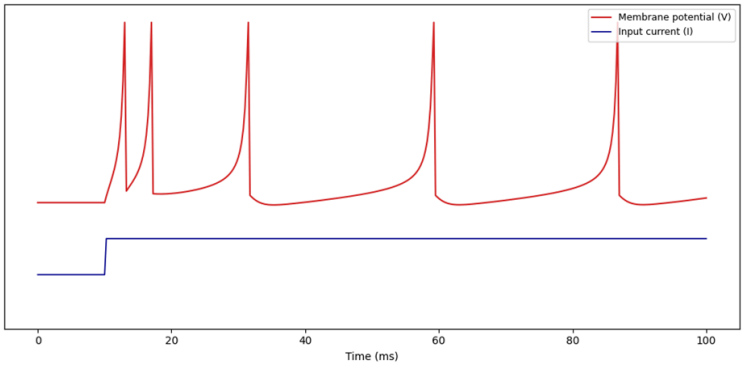


Figure 2.5: Phasic spiking neurocomputational property of biological spiking neurons.

Phasic Spiking: This pattern simply generates a single spike in response to a step stimulus as illustrated in Fig. 2.5. After the first spike, it follows a period of silence. It acts as a temporal edge detector, responding to the beginning of stimulation

rather than its continuous presence. The phasic behavior of a neuron plays a vital role in detecting novelty, identifying changes in sensory input, and segmenting signals over time. The adaptation mechanism involved in this pattern rapidly increases the recovery variable (u), which suppresses further spike generation. This behavior acts like a high-pass filter for a step-persistent input stimulus [14], [15], [16].

Tonic Bursting: This pattern exhibits a rhythmic cluster of rapid spikes that represents important information about the applied stimulus, as shown in Fig. 2.6. It is typically observed in chattering neurons of the cat neocortex [56], which contribute to gamma-frequency oscillations (30–80 Hz) associated with attention, sensory integration, and consciousness. The modified reset potential ($c = -50\,\text{mV}$, compared to the typical $c = -65\,\text{mV}$) enables faster recovery after depolarization. As a result of this, multiple spikes can be observed within a burst before the recovery variable accumulates and terminates the activity. Tonic bursting supports temporal segmentation in neural encoding and helps to maintain network synchronization [14], [15], [16].

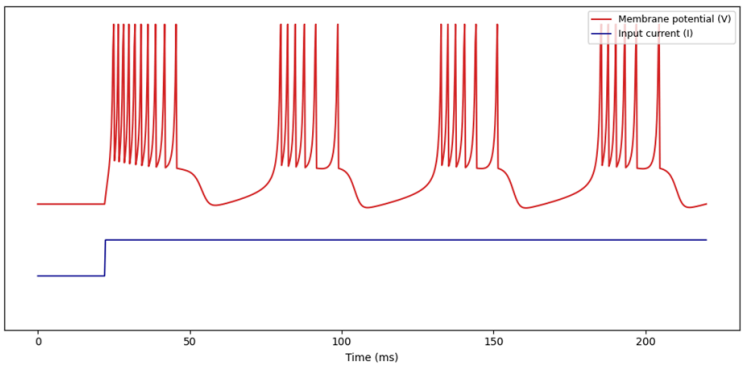


Figure 2.6: Tonic bursting neurocomputational property of biological spiking neurons.

Phasic Bursting: Here, a single burst of rapid spikes is exhibited at the beginning of a stimulus and then follows the same silence period as in the case of phasic behavior shown in Fig. 2.7. This phenomenon offers several computational advantages, such as (1) reliability enhancement: bursts reduce the effects of synaptic transmission failures and neuronal noise that ensure stable communication in unreliable networks [57], (2) salience encoding: bursts have a more substantial influence on postsynaptic neurons than single spikes that effectively highlights important events, and (3) selective communication: the frequency of spikes within a burst enables specific routing of information between neurons that allow multiplexed signaling pathways [58]. For cortical network models to achieve biological realism,

bursting mechanisms are essential for accurate functional representation [14], [15], [16].

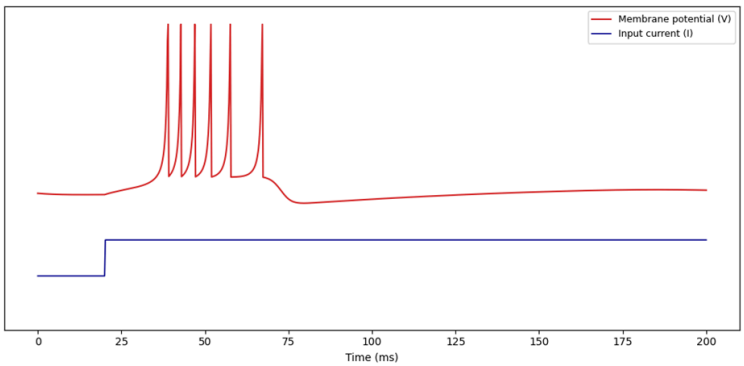


Figure 2.7: Phasic bursting neurocomputational property of biological spiking neurons.

Mixed Mode: This pattern consists of an initial burst of spikes succeeding regular tonic spiking as shown in Fig. 2.8. This mixed behavior enables neurons to encode both temporal and rate information. It is commonly observed in the intrinsically bursting (IB) excitatory neurons of mammalian neocortex [54]. The computational role of this pattern includes: (1) Onset detection – the initial burst signals the onset of the stimulus with high salience, and (2) Magnitude encoding – the subsequent tonic firing rate reflects input strength. This dual-mode operation provides two pieces of information from a single neuron: when an event occurs (through burst timing) and how strong the stimulus is (through firing rate) [14], [15], [16].

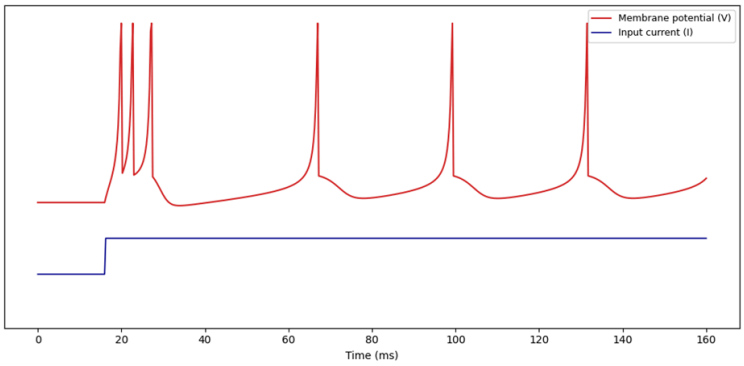


Figure 2.8: Mixed-mode neurocomputational property of biological spiking neurons.

Spike Frequency Adaptation: In this pattern, a gradual decrease of firing rate is observed during sustained input as illustrated in Fig. 2.9. This behavior helps to

normalize neuronal responses. This spiking phenomenon is typically associated with one of the most common cortical neurons, regular spiking (RS) excitatory neurons, and also with low-threshold spiking (LTS) inhibitory interneurons. Functionally, spike frequency adaptation enhances temporal contrast and encodes elapsed time. This pattern allows neurons to act as an intrinsic timer, as the instantaneous inter-spike interval (ISI) carries exact time information. The calcium-activated potassium currents and synaptic depression, as an adaptation mechanism, can provide gain control that enables cortical networks to respond preferentially to novel or changing stimuli rather than static background signals [14], [15], [16].

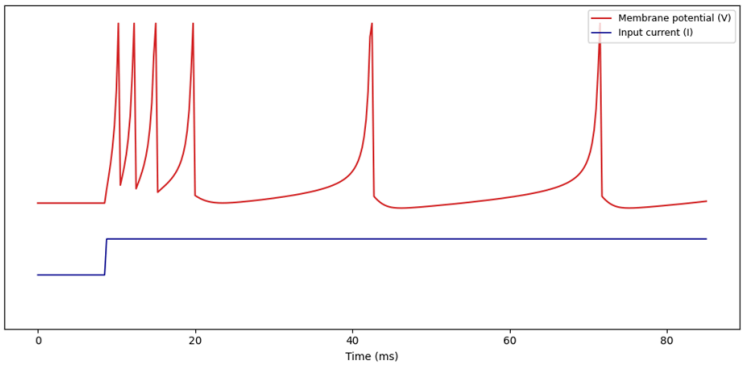


Figure 2.9: Spike frequency adaptation neurocomputational property of biological spiking neurons.

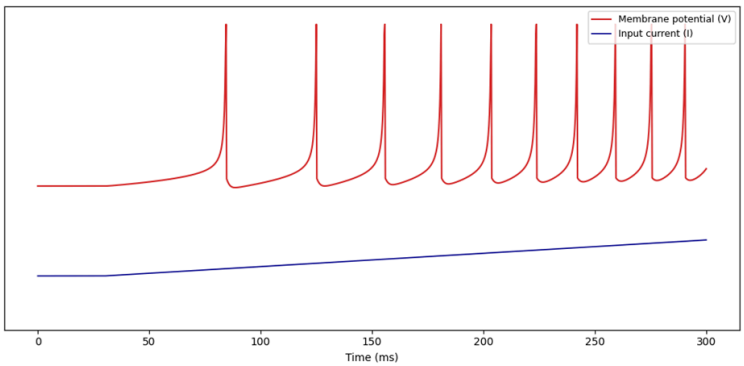


Figure 2.10: Class 1 excitability neurocomputational property of biological spiking neurons.

Class 1 Excitability: Due to a saddle-node bifurcation, the neurons that hold this excitability property can fire at arbitrarily low frequencies near threshold as indicated in Fig. 2.10. It is an important feature of neocortical regular spiking

(RS) excitatory neurons [59], [60]. It allows them to respond to weak suprathreshold inputs. Unlike Class 2 neurons, which exhibit a minimum jump in firing frequency, Class 1 neurons support continuous rate coding that enables encoding input strength across a broad range (2–200+ Hz) without dead zones. This permits graded analog-to-digital conversion and smooth control of firing rates. The excitability arises from saddle-node bifurcation dynamics, where the stable equilibrium and limit cycle merge. Class 1 excitability is important for population coding, where stimulus intensity is represented by distributed firing rates across neural ensembles, maximizing information transfer through linear frequency–input (f-I) relationships [14], [15], [16].

Class 2 Excitability: Neurons with this property fire at a relatively high minimum frequency at onset as shown in Fig. 2.11. They exhibit a binary threshold behavior. They are either quiescent or fire above a set frequency (typically $\geq$ 40 Hz), with no low-frequency states [59], [60]. This behavior arises from Hopf bifurcation dynamics, where oscillatory limit cycles appear at a finite frequency rather than gradually from equilibrium. Class 2 neurons act as threshold detectors or binary switches that signal the presence or absence of a stimulus rather than its magnitude, which is different from the Class 1 case. This makes them well-suited for decision-making circuits, winner-take-all competitions, and bistable memory states, where discrete state transitions are preferred over graded analog coding [14], [15], [16].

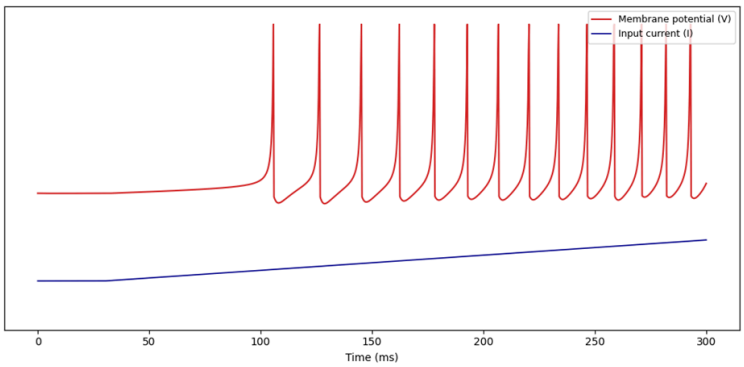


Figure 2.11: Class 2 excitability neurocomputational property of biological spiking neurons.

Spike Latency: This pattern encodes input strength through the delay to the first spike, enabling rapid temporal coding as exhibited in Fig. 2.12. It is a key mechanism in cortical neurons, especially regular spiking (RS) cells, where response latency can vary by tens of milliseconds depending on input intensity. The main computational advantage is time-to-first spike encoding, which characterizes that

stronger inputs elicit faster spikes, while weaker inputs produce longer delays. This allows neurons to represent stimulus magnitude through latency rather than firing rate, supporting rapid single-spike communication without the need for long observation windows. Spike latency also enables temporal multiplexing, where the timing of population responses generates spatiotemporal patterns that encode multidimensional stimuli [14], [15], [16].

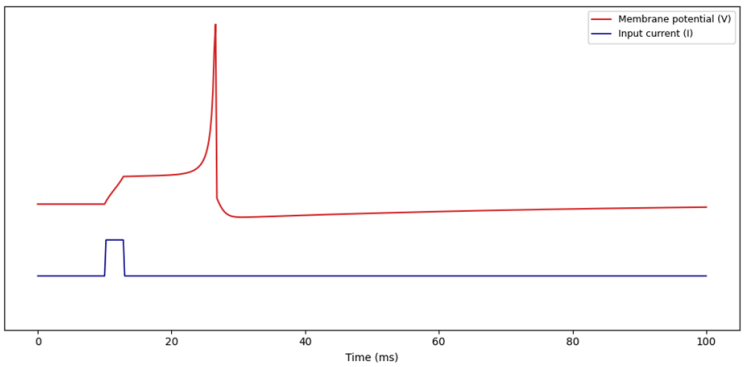


Figure 2.12: Spike latency neurocomputational property of biological spiking neurons.

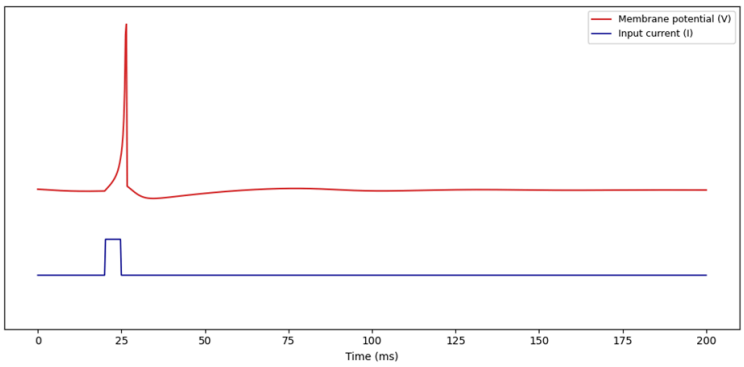


Figure 2.13: Subthreshold oscillations neurocomputational property of biological spiking neurons.

Subthreshold Oscillations: This pattern shows small and damped fluctuations of membrane potential as illustrated in Fig. 2.13. The subthreshold oscillations are common in the brain and have significant roles in neural information processing. They arise from the interaction of voltage-gated conductances, producing intrinsic membrane resonance at characteristic frequencies (e.g., theta: 4–8 Hz, alpha: 8–12 Hz, gamma: 30–100 Hz, depending on neuron type). They typically amplify inputs that match the neuron's resonant frequency and attenuate the inputs that

are off-frequency signals. Due to these behaviors, computationally, these oscillations act as bandpass filters. This allows neurons to perform frequency-selective integration, supporting: (1) temporal feature detection of rhythmic patterns, (2) selective communication within oscillatory networks, and (3) enhanced coincidence detection by amplifying synchronized inputs. Reducing the recovery reset parameter value to $d = 0$ prevents damping and enables sustained oscillatory dynamics, which are critical for network synchronization [14], [15], [16].

Resonator: Resonator neurons only respond to the inputs that match their intrinsic frequency perfectly as displayed in Fig. 2.14. They implement frequency-selective integration by aligning input timing with intrinsic subthreshold oscillations, enabling advanced temporal filtering [58], [61]. Their computational functions include: (1) Frequency-modulation (FM) detection – neurons best react to inputs with a frequency that equals their resonance frequency and rejects out off-frequency inputs, (2) Temporal multiplexing – several channels of information can be represented at the same time, with resonant cells selectively attuning to a channel, and (3) Enhanced coincidence detection – the resonant responses are maximized by constructive interference, and the signal-to-noise ratio is improved. The negative recovery increment ($d = -1$) reverses typical adaptation, creating anti-recovery that sustains oscillations and sharpens frequency selectivity. This process favors communication using coherence in oscillatory neural networks [14], [15], [16].

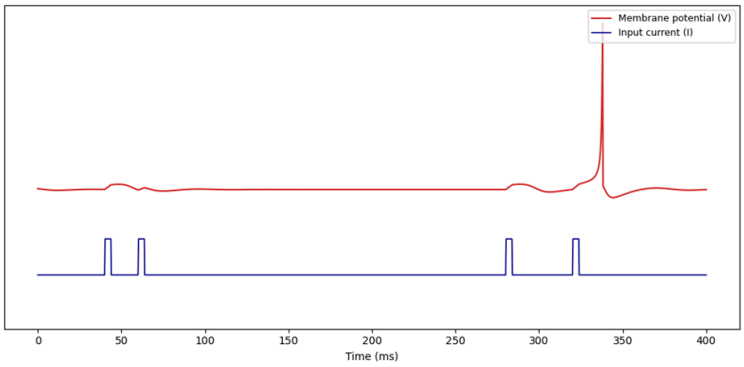


Figure 2.14: Resonator neurocomputational property of biological spiking neurons.

Integrator: Integrator neurons primarily detect coincident inputs and sum incoming signals over time as shown in Fig. 2.15. They are not of an oscillatory nature as resonators, but rather function as temporal integrators, responding preferentially to high-frequency or closely timed inputs. The membrane potential passively integrates synaptic inputs, and therefore, firing likelihood rises with the frequency of synaptic inputs because closely spaced inputs add up before decaying. Integrators play key roles in: (1) spike-timing-dependent detection of correlated

presynaptic activity, (2) implementing AND-gate logic requiring convergent input synchrony, and (3) high-pass temporal filtering, where only rapid input sequences produce spikes. Compared to resonators, which are sensitive to certain frequencies, integrators generally favor high-frequency input and are widely used as coincidence detectors [14], [15], [16].

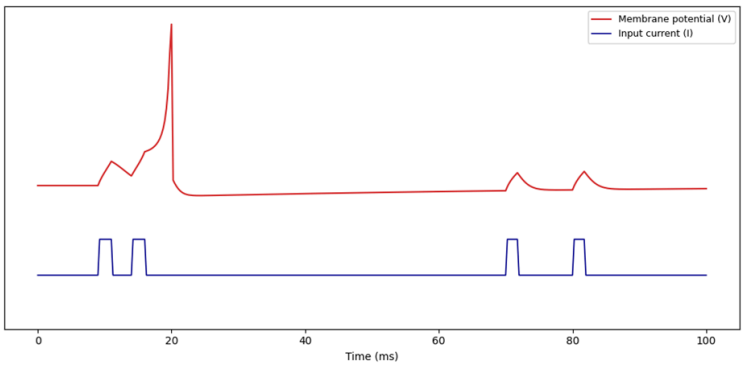


Figure 2.15: Integrator neurocomputational property of biological spiking neurons.

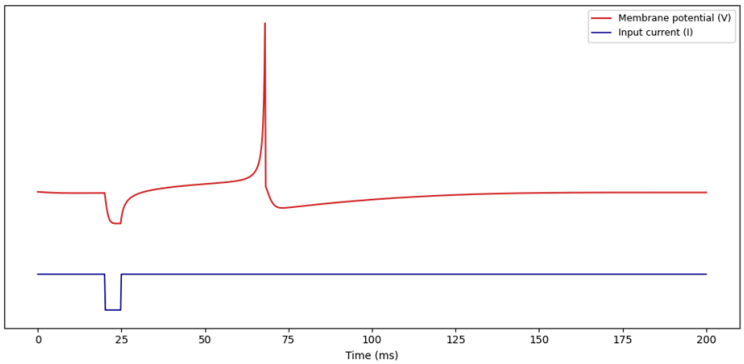


Figure 2.16: Rebound spiking neurocomputational property of biological spiking neurons.

Rebound Spike: It is a pattern in which a neuron releases a spike as a result of anodal break excitation or hyperpolarization as illustrated in Fig. 2.16. The recovery variable (u) decreases during hyperpolarization, which deactivates voltage-gated channels. When inhibition ends and u remains low, the membrane potential rebounds rapidly, which produces a transient depolarization that exceeds threshold. Rebound spikes play several significant roles in computation, such as inhibition paradox: brief inhibitory inputs can generate delayed excitation that enables complex temporal coding, disinhibition amplification: release from inhibition acts as an active excitatory signal, rhythm generation – rebound properties

support oscillatory network dynamics through reciprocal inhibition-rebound cycles, and temporal contrast detection – neurons respond to inhibition offsets that can detect transitions and boundaries. This mechanism shows that not solely excitatory or inhibitory synaptic input is important, but also timing and dynamics play a significant role in neural computation [14], [15], [16].

Rebound Burst: This pattern is usually observed in thalamocortical relay neurons when hyperpolarization triggers a burst of spikes as shown in Fig. 2.17. Immediately after inhibition release, these neurons fire high-frequency bursts that play a key role in sleep oscillations and thalamocortical rhythms. Reciprocal inhibition between the thalamic reticular nucleus and relay cells produces oscillatory cycles of inhibition, then a rebound burst, and then again inhibition, which generates spindle waves (7–14 Hz) during non-REM sleep. The computational functions of rebound bursting include four processes, namely rhythmogenesis: sustaining network oscillations without dedicated pacemaker neurons, state-dependent transmission: bursts occur during low-vigilance states (sleep or drowsiness) and switch to tonic firing during arousal, allowing state-dependent information gating, temporal amplification: brief inhibition produces prolonged high-frequency output, amplifying inhibitory signals, and synchronization: bursts provide strong depolarizing input to downstream neurons, enhancing network coherence [14], [15], [16].

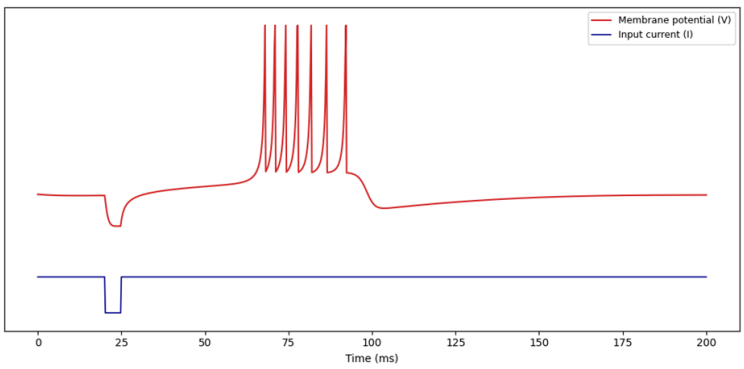


Figure 2.17: Rebound bursting neurocomputational property of biological spiking neurons.

Threshold Variability: This pattern is sensitive to prior activity that reflects history-dependent changes in neuronal excitability as shown in Fig. 2.18. Biological neurons normally adjust their firing threshold through the recovery variable (u), which is not the case in fixed-threshold-based artificial neurons. The recovery variable u decreases during inhibition, lowering the effective threshold, and increases when excitation occurs, raising the threshold. This produces activity-dependent

excitability with several computational roles, such as contextual sensitivity: responses depend on recent input history, supporting short-term memory, contrast enhancement: reduced excitability after excitation sharpens temporal selectivity and prevents redundant firing, disinhibition gain control: inhibition can increase subsequent excitability, amplifying post-inhibitory inputs, and adaptive filtering: threshold modulation acts as automatic gain control, adjusting sensitivity to input statistics. This dynamic threshold enables neurons to encode relative changes rather than absolute values, performing temporal derivative operations, which are essential for sensory adaptation and novelty detection [14], [15], [16].

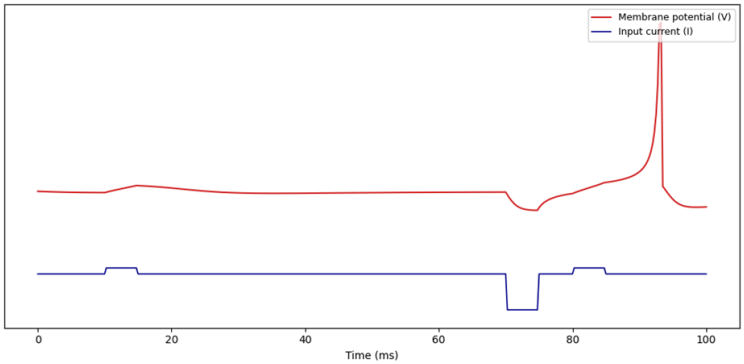


Figure 2.18: Threshold variability neurocomputational property of biological spiking neurons.

Bistability: This is a phenomenon in which the neurons sustain firing despite the short-term termination of the stimulus, which is a way of storing information as demonstrated in Fig. 2.19. This neuron can be in two stable states, that is, resting or tonic spiking/bursting, which allows it to implement biological flip-flop memory. Computationally, bistability has several properties, such as short-term memory: neurons maintain a firing state representing recent inputs without synaptic changes, supporting working memory, binary storage: two discrete states encode 1-bit information in networks, state-dependent processing: network computations depend on the configuration of bistable neurons, enabling context-sensitive circuits, and phase-sensitive control: maintaining or terminating activity requires appropriately timed signals, emphasizing spike-timing precision. Bistability underlies persistent activity in the prefrontal cortex during delay periods, eye position maintenance in oculomotor integrators, and decision-making circuits such as winner-take-all networks [14], [15], [16].

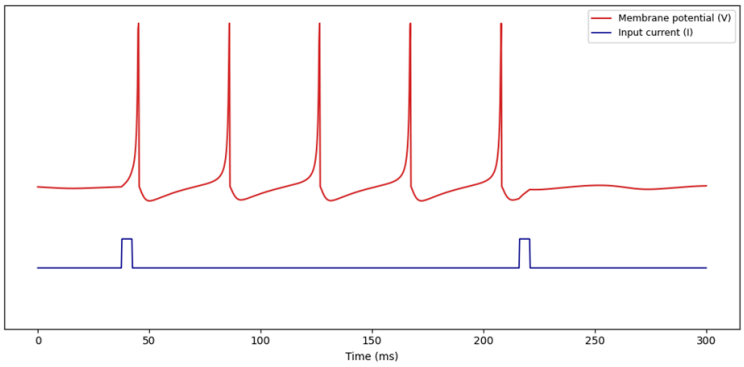


Figure 2.19: Bistability neurocomputational property of biological spiking neurons.

Depolarizing After-potential (DAP): This type of pattern is found when a neuron undergoes a post-spike depolarization, which puts the neuron into a superexcitable state rather than a refractory period as shown in Fig. 2.20. Biological mechanisms include: (1) dendritic calcium influx during spikes activating plateau potentials, (2) high-threshold inward currents (persistent sodium and calcium) that outlast the action potential, and (3) subthreshold voltage-gated current interactions producing regenerative depolarization. Computationally, DAPs provide the ability to shorten refractory periods and therefore increase firing rates and bursting, increase temporal integration window to enable delayed inputs to summate, cause spike doubling or tripling as a consequence of rapid re-firing, and provide gain state dependence in which neurons become more responsive to later inputs. DAPs facilitate bursting in intrinsically bursting neurons, improve dendritic integration in the pyramidal cells, and facilitate quick frequency modulation in motor neurons to cause rapid changes in fast output [14], [15], [16].

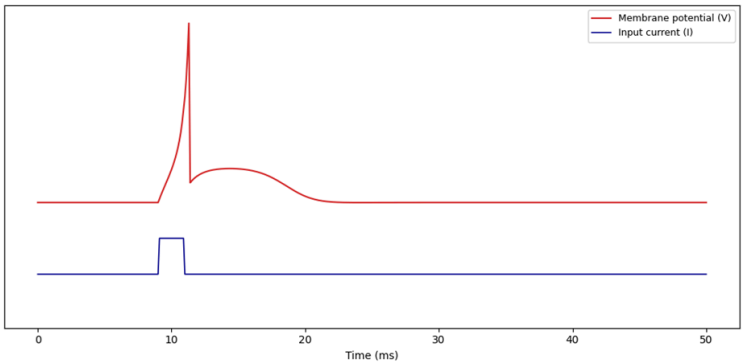


Figure 2.20: Depolarizing after-potential neurocomputational property of biological spiking neurons.

Accommodation: This is a pattern that arises when neurons are unable to produce spikes to slow depolarizing currents, despite the magnitude of the input being large enough to trigger spiking, which is considered to be high-pass temporal filtering as illustrated in Fig. 2.21. Inward currents (sodium and calcium) are inactivated and outward currents (potassium) are activated during slow ramps and gradually increase the threshold and decrease excitability. Computationally, accommodation supports: (1) high-pass temporal filtering – neurons are sensitive to transient stimulus and are insensitive to slow inputs, (2) coincidence detection – brief synchronous inputs overcome accommodation while asynchronous inputs are filtered out, (3) contrast enhancement – adaptation preserves sensitivity to transients, and (4) dynamic range preservation – limiting saturation in situations with gradual increases in input. This process enables sensory mechanisms to remain sensitive across wide input ranges; this is why motion and novel stimuli are detected more effectively than static backgrounds [14], [15], [16].

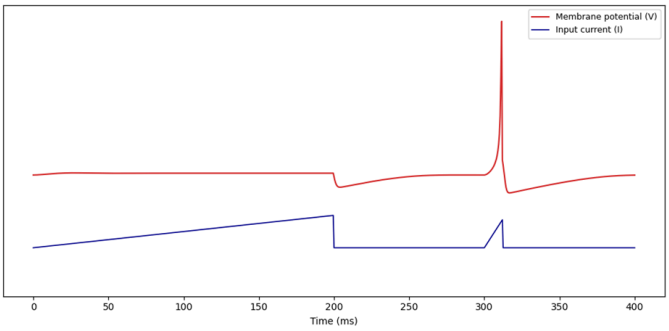


Figure 2.21: Accommodation neurocomputational property of biological spiking neurons.

Inhibition-Induced Spiking: This is a type of pattern where sustained hyperpolarization gives rise to neuronal firing, but in a paradoxical manner as displayed in Fig. 2.22. It is found in thalamo-cortical relay neurons, which are capable of firing during hyperpolarization and are silent during rest. This process is associated with hyperpolarization-activated currents: (1) h-current (Ih) is a slow depolarizing drive of hyperpolarization, and (2) T-type calcium currents deinactivate, which allows calcium-mediated depolarization and the production of spikes. Inhibition-induced spiking displays several properties: state-dependent gating – neurons switch between relay mode and burst mode, with inhibition-induced firing contributing to rhythmic activity; sleep rhythms – hyperpolarization triggers rebound bursts that participate in sleep spindles and slow-wave oscillations; paradoxical gain control – inhibition generates activity rather than suppressing it; oscillatory entrainment – periodic inhibition drives rhythmic firing, synchronizing thalamo-cortical networks [14], [15], [16].

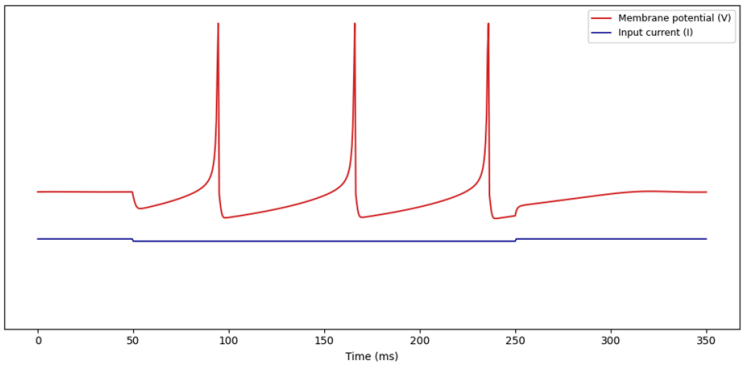


Figure 2.22: Inhibition-induced spiking neurocomputational property of biological spiking neurons.

Inhibition-Induced Bursting: In this pattern, rhythmic bursts of spikes are triggered by sustained hyperpolarization as shown in Fig. 2.23. It plays a major role in the oscillation of the spindle waves (7–14 Hz) in the thalamo-cortical system during non-REM sleep. The mechanism of the underlying activity is analogous to inhibition-induced spiking, except that an increase in the reset potential permits bursts rather than single spikes. Functional roles include: (1) Sleep rhythm generation – rhythmic inhibition from the thalamic reticular nucleus triggers rebound bursts in relay cells, producing spindle oscillations that support memory consolidation and sleep regulation, (2) Thalamo-cortical synchronization – bursts provide strong depolarizing drive, aligning cortical network activity, (3) State-dependent information gating – burst mode during hyperpolarization prevents sensory relay (sleep), while tonic mode during depolarization allows signal transmission (wakefulness), and (4) Network oscillations – reciprocal inhibition-burst cycles between reticular and relay nuclei sustain self-perpetuating rhythms without external pacing [14], [15], [16].

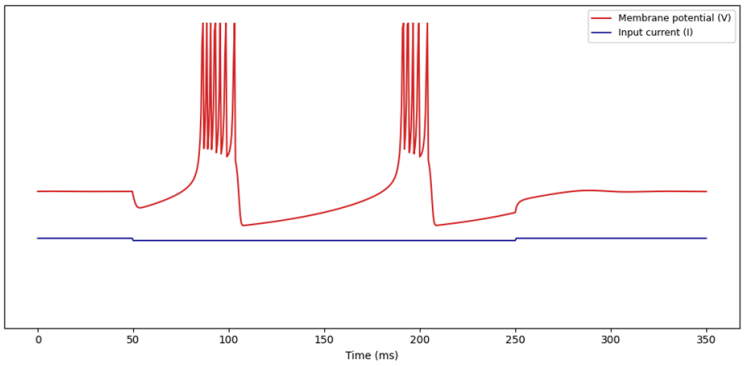


Figure 2.23: Inhibition-induced bursting neurocomputational property of biological spiking neurons.

2.4 FPGA, PYNQ-Z2, and Zynq SoC

2.4.1 Field-Programmable Gate Arrays (FPGAs)

FPGAs are a special type of integrated circuit that combines software flexibility with hardware performance. They enable users to design and reconfigure digital systems after manufacturing, making them very flexible for research and prototyping. FPGAs can be programmed to model any digital circuits, unlike general-purpose microprocessors, which execute instructions sequentially, or application-specific integrated circuits (ASICs), which only perform predefined tasks. This programmability has provided them with a great platform to develop and test new computational models, including biologically inspired neural systems.

An FPGA is built of a large number of configurable logic blocks (CLBs) linked together by a programmable routing network. The CLBs consist of multiplexers, flip-flops, and lookup tables (LUTs), which are the building blocks of digital logic. LUTs are small memory units that store truth tables and are capable of executing any Boolean function. Binary data is stored in flip-flops, and they can be used to implement sequential logic by aligning the logic with a clock signal. Multiplexers control data flow within the circuit by selecting between multiple inputs. Recent FPGAs also have a number of dedicated hardware units to enhance performance. Digital signal processing (DSP) slices have inbuilt multipliers, adders and accumulators, which are optimized to perform rapid arithmetic operations. They are particularly useful in the field of signal processing, control systems and the computation of neuron models. Block RAM (BRAM) provides large internal memory blocks, typically in 18 Kb or 36 Kb blocks, which can be used to store parameters, intermediate results, or neural states. Input/output blocks (IOBs) deal with communication with external devices and provide multiple voltage levels and communication standards. These elements altogether enable an FPGA to form a fully customizable computing platform [62], [63], [64].

In the case of neuron model implementation, FPGAs have a number of significant benefits. Their huge parallelism allows thousands of simulated neurons to interact simultaneously, and each of them changes its state with each clock cycle. This enables real-time neural network simulation, which is not easy to do on traditional processors. The predictable timing of FPGA hardware guarantees that the computations take place within the limits of predictable and fixed time, which is important in time-sensitive and real-time tasks in neural processing. Custom numeric precision is also supported by FPGAs, where the designers are free to select the appropriate fixed-point or reduced-precision format to achieve the right balance between accuracy, resource utilization, and power consumption. Low-latency response of FPGA systems is another great advantage. Since processing occurs directly within hardware, data may be read, processed, and output nearly immediately, allowing the system itself to respond to input signal changes. This is critical

to neural encoding applications in which timing accuracy and rapid feedback are needed. Also, with the self-reconfigurability of FPGAs, neuroscience researchers can modify models of neurons, connection patterns, and learning rules without re-designing the whole hardware, greatly accelerating the development of experiments.

2.4.2 Zynq-7000 SoC and PYNQ-Z2 Development Platform

The Xilinx Zynq-7000 family is a breakthrough in the embedded system design concept by combining an entire processing system with programmable logic on a single silicon die. This is a heterogeneous architecture that integrates the flexibility of software processing and the performance benefits of custom hardware acceleration, and is well-suited to the application of complex neuron models and signal processing applications.

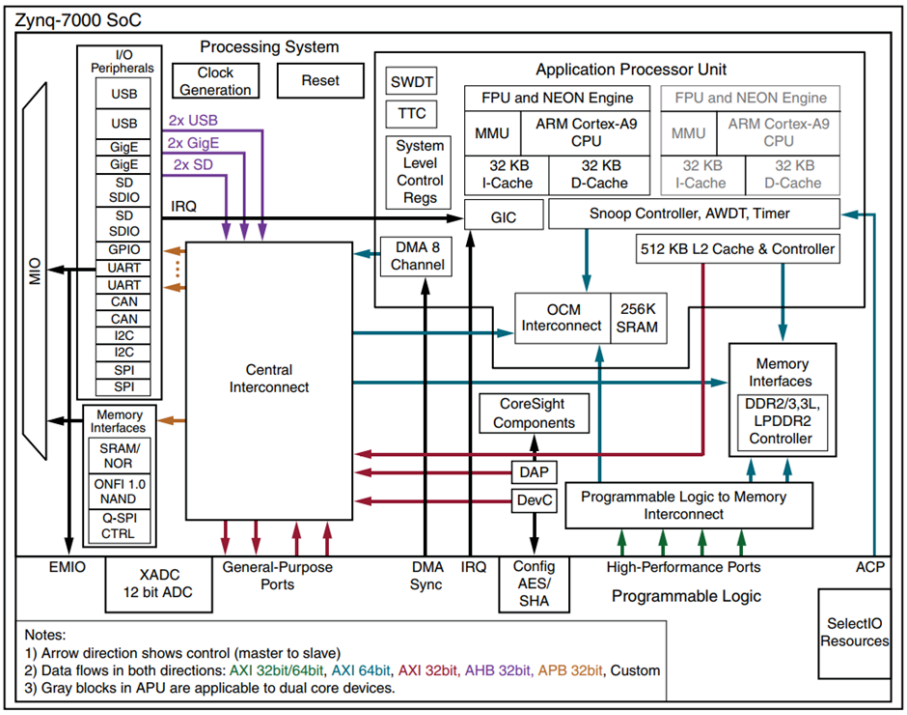


Figure 2.24: Xilinx Zynq-7000 SoC architecture showing the Processing System (PS) with dual ARM Cortex-A9 cores, cache hierarchy, peripheral interfaces, and the Programmable Logic (PL) section with configurable logic blocks, DSP slices, and block RAM connected via multiple AXI interface ports [65].

The Xilinx Zynq-7000 SoC detailed architecture is depicted in Fig. 2.24. The Processing System (PS) has two ARM Cortex-A9 processors operating at a maximum 1 GHz, with 32 KB instruction and data caches, floating-point units, and

NEON engines capable of performing vector processing. The PS has large peripheral interfaces, memory controllers, and system-level functions that are normally found in application processors. This enables the PS to execute complete operating systems like Linux and perform complex software tasks such as user interfaces, network communications, and high-level control algorithms.The Programmable Logic (PL) portion includes the FPGA fabric, which comprises configurable logic blocks, DSP48E1 slices, and block RAM. The XC7Z020 chip installed on the PYNQ-Z2 board has 53,200 logic cells, 220 DSP slices, and 630 KB block RAM. The DSP48E1 slices are of particular significance to neuron models as they can achieve 25×18-bit multiplications at a rate of up to 450 MHz, which is suitable for the implementation of quadratic terms of the Izhikevich model. The communication between PS and PL is over several AXI4 interfaces, which offer various forms of connectivity depending on data flow patterns. There are four AXI General Purpose (GP) ports that can be used as master and slave connections to control and transfer low-bandwidth data, which are usually used to configure neuron parameters and read status registers. Four AXI High Performance (HP) ports allow high bandwidth data streaming between PL and PS memory. The AXI Accelerator Coherency Port (AXI ACP) is designed to permit the access of PS cache-coherent memory by PL, so that shared data structures can be efficiently used. Other connections are event notification interrupts, various clock domains, and direct memory access (DMA) channels for autonomous data movement [65], [66], [67].

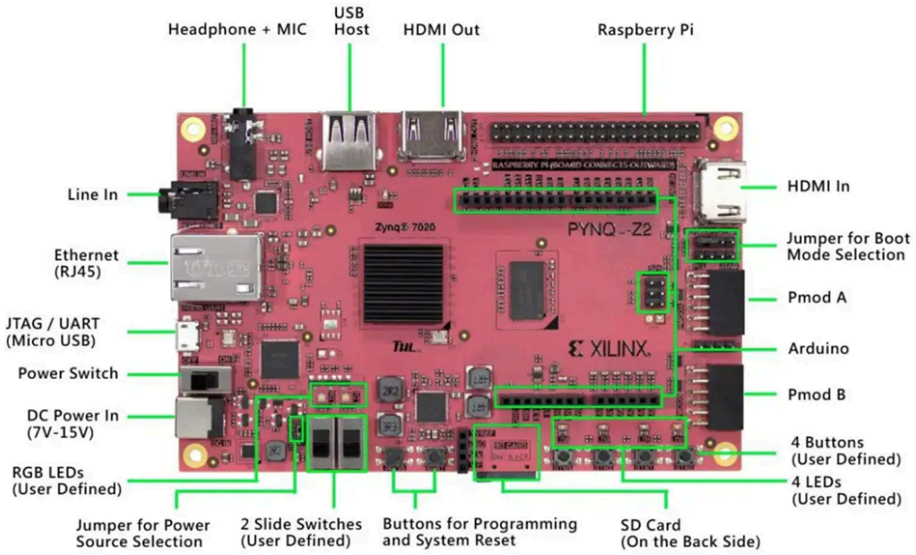


Figure 2.25: PYNQ-Z2 development board layout showing the Zynq-7020 SoC at the center, surrounded by DDR4 memory, peripheral interfaces including Ethernet, USB, HDMI, audio codec, expansion connectors (PMOD A/B, Arduino headers), user interface elements (LEDs, buttons, switches), and power management components [67].

Figure 2.25 illustrates the PYNQ-Z2 development board that offers a complete platform to develop and test neural processing systems on the FPGA platform. The board is designed around the XC7Z020-1CLG400C Zynq and has extensive peripheral interfaces and expansion capabilities that are intended to be used in research and prototyping. The PYNQ (Python Productivity for Zynq) framework, supported by the board, enables the use of high-level Python interfaces for programming the FPGA instead of traditional hardware description languages, thereby saving a lot of development time and complexity [67], [68].

The memory subsystem has 512 KB of DDR4-2133 SDRAM, which is linked to the PS memory controller and offers high-speed access to both PS and PL by the memory interconnect. This shared memory facilitates effective data transfer between computer software and hardware. It has a microSD card slot that supports boot image and file system access, enabling easy updating and data logging. It also has 16 MB Quad-SPI Flash memory, which is used as a configuration storage, and 256 KB on-chip memory (OCM), which both PS and PL can access with minimum latency. The connectivity options facilitate a wide range of input/output requirements for neural processing applications. The Gigabit Ethernet offers network connectivity for remote control, data transfer, and distributed processing applications. USB 2.0 host and OTG ports have external devices and programming interfaces. There are HDMI input and output ports that allow video processing applications and can be used as a source in visual signal-to-spike encoding tests. An audio jack with stereo line input/output and microphone input (3.5 mm) is used to support audio signal processing applications (i.e., practical in testing auditory neural encoding algorithms). Expansion interfaces give the ability to connect custom sensors and actuators. Two PMOD connectors, each with 8 single-ended or 4 differential signals, can support many digital and analog interface modules. Arduino-compatible headers enable connection to both digital I/O and analog inputs through an onboard 12-bit ADC, allowing direct connection of sensors and signal sources. Such interfaces are especially useful for linking real-world signal sources to neuron encoder implementations and for connecting external measurement equipment. User interface elements facilitate development and debugging processes. There are four user-controllable LEDs used to provide visual feedback of the system states, e.g, neuron activity patterns. There are four push buttons and two slide switches, which can be used as a manual input to test and manipulate the parameters. RGB lights offer multi-color status showing various modes of operation. These features are useful for tracking design states, debugging implementations, and managing user interaction during the development and testing stages. The PYNQ-Z2 features power management that supports both USB and external power sources, accommodating various usage scenarios. The micro-USB connector can be used to power the board during development and light prototyping, or an external 7–15 V DC supply can be used to power the board when operating independently and for high-power applications. Onboard voltage regulators supply the various supply voltages

that the Zynq device and peripheral components need, such as core logic voltages, I/O voltages, and analog supply rails. Power selection jumpers enable the selection of power sources and power domain configuration to meet different operational needs [65], [66], [67].

The PYNQ software platform is a game-changer in the development of FPGAs by offering Python-based access to hardware accelerators via a high-level programming interface. PYNQ overlays are pre-programmed hardware designs that can be loaded onto the PL and accessed using Python APIs, without using low-level hardware description language programming. The framework has detailed libraries on common functionality like DMA transfer, interrupt processing, GPIO control, and memory management. The integration of Jupyter notebooks offers an interactive development experience in which hardware experiments can be recorded alongside code and results, enabling reproducible research and collaborative development. The availability of robust dual-core processing, a wide range of connectivity choices, flexible expansion interfaces, and an easy-to-use software framework makes the PYNQ-Z2 platform especially well-suited for implementing and testing neuron-based signal processing systems. This capability to smoothly integrate software control with parallel hardware processing offers the flexibility required by research applications and the performance needed for real-time neural computation and signal-to-spike encoding applications.

Chapter 3

Materials and Methods

3.1 Mathematical Model Implementation

The digital implementation of spiking neuron models must be computationally tractable and biologically accurate. Specifically, the Izhikevich model, represented by a couple of continuous-time differential equations, requires a series of transformations to be made compatible with FPGA implementation. These changes involve the discretization of the governing equations, the choice of a suitable numerical integration method, and the careful choice of arithmetic precision based on fixed-point representations. The design strategies adopted to guarantee a stable, accurate, and hardware-specific implementation are described in the following subsections.

3.1.1 Izhikevich Neuron Model Discretization

Two interconnected differential equations characterize the original Izhikevich neuron model as [14]:

$$\frac{dv}{dt} = 0.04 * v^2 + 5 * v + 140 - u + I \tag{3.1}$$

$$\frac{du}{dt} = a * (b * v - u) \tag{3.2}$$

with the after-spike reset condition:

$$\text{if } v \geq 30 \text{ mV}, \text{then} \begin{cases} v \rightarrow c, \\ u \rightarrow u + d \end{cases} \tag{3.3}$$

where v is the membrane potential in mV, u is the membrane recovery variable, I is the external input current and a, b, c, d are dimensionless parameters controlling different spiking patterns. The nonlinearity of the quadratic equation (3.1) allows for the rapid depolarization that occurs after the spike initiation, and the linear dynamics of the equation (3.2) model the slower recovery mechanisms.

In order to implement these dynamics on digital hardware, the continuous system must be discretized. Numerical discretization plays an important role in determining the accuracy of the reproduced dynamics and the stability of the simulation. While several higher-order methods (like Runge-Kutta) could be employed for better accuracy, the costs in terms of hardware are significantly higher. In contrast, the forward Euler method is a simple and computationally efficient alternative that is suitable for FPGA implementation [69].

The discretized equations using Euler integration are written as:

$$\Delta v = (0.04 * v[n]^2 + 5 * v[n] + 140 - u[n] + I[n])\,\Delta t \tag{3.4}$$

$$\Delta u = a * (b * v[n] - u[n])\,\Delta t \tag{3.5}$$

$$v[n+1] = v[n] + \Delta v \tag{3.6}$$

$$u[n+1] = u[n] + \Delta u \tag{3.7}$$

$$\text{if } v[n+1] \geq 30 \text{ mV}, \text{then} \begin{cases} v[n+1] & \to c, \\ u[n+1] & \to u[n+1] + d \end{cases} \tag{3.8}$$

where Δt is the integration time step. Considering both biological and numerical perspectives, Δt is chosen as 0.25 ms. It provides an adequate resolution to capture spiking events without distortion and also prevents instabilities that may arise with larger time steps [15].

3.1.2 Fixed-Point Arithmetic Design and Precision Analysis

Floating-point arithmetic is highly flexible, but it is too costly in terms of FPGA system resources. To address these constraints, fixed-point arithmetic is employed, and a subtle balance is required between dynamic range, resolution, and hardware efficiency. The fixed-point architecture must be tailored to the statistical properties of the model variables in a manner that prevents overflow, underflow, or loss of precision in computations. A simulation of the Izhikevich model on its full parameter space reveals that the membrane potential typically ranges between -80 mV and $+40$ mV, with momentary deviations to the reset value as negative as -70 mV. The recovery variable u usually stays within roughly ± 20, although its exact range depends on the membrane potential, the applied input current, and the parameter b. The input currents used in the experiment are in the range [0, 30], though negative values are also possible to represent inhibitory synaptic input. Collectively, these ranges determine the required precision, dynamic range, and scaling of the arithmetic units in hardware implementations to ensure accurate neuron behavior.

To meet these requirements, the Q5.11 fixed-point format with five integer bits, eleven fractional bits, and one sign bit is chosen. This representation gives a dynamic range of ± 32, which is enough to represent all relevant values with a margin

of safety, and resolution of about $2^{-11} \approx 0.0004882$. Such resolution permits sub-millivolt accuracy of membrane potentials, which is important for maintaining the fine temporal structure of spikes. Furthermore, the total word length of 16 bits is compatible with FPGA DSP slices, which enables efficient utilization of on-chip arithmetic resources.

Another point to consider in fixed-point implementation is the scaling of model coefficients. In the Q5.11 representation, constants are pre-scaled into integer values, which can be directly used in multiplication operations. For example, the coefficient 0.04 in equation (3.2) is converted to 82 $(0.04 * 2^{11})$, the linear coefficient 5 becomes 10240, and the constant 140 is converted to 286720. By pre-calculating and storing these scaled constants in special registers, the conversion overhead can be removed at runtime. Multiplication results are stored in extended 64-bit registers in order to prevent overflow and are then converted back to Q5.11 format using arithmetic right shifts (>>). This strategy ensures both numerical accuracy and hardware efficiency.

The other possible resource optimization factor is bit-width allocation [70]. The state variables (v, u), state parameters (a, b, c, d) and input current (I) are all internally represented using 32-bit registers. Such an allocation provides enough headroom beyond the Q5.11 representation to accommodate intermediate values without saturation. The intermediate multiplication values are stored in 64-bit registers, such that high-precision computations, like quadratic terms, are performed with no error propagation. Registers with 32 bits are used to control the data flows between stages of computational pipelines and memory. It enables an effective communication linkage between BRAMs and external communication links.

This discretization, combined with scaling of the coefficients and optimizing the bit-width, results in a digitally realized computationally reasonable efficient version of the neurons that is biologically faithful. Through extended precision representation of Q5.11, the model prevents numeric instability problems, and it provides real-time performance on hardware.

3.2 Four-Stage Pipeline Architecture Design

3.2.1 Pipeline Design Rationale

A pipelined computational structure implementation is critical to the achievement of biological fidelity and real-time computation of neuromorphic systems on FPGA. The Izhikevich neuron model, which integrates quadratic nonlinear dynamics with threshold-based resets presents specific difficulties to digital hardware realization, since both high arithmetic accuracy and strong time coherence between the membrane potential and the recovery variable are required. These requirements can be met efficiently by a pipeline structure, and can also be scaled to large neural

populations.

In FPGA-based design, the selection of pipeline depth requires a trade-off between throughput, latency, resource utilization, and timing closure [71]. Shallow pipelines have low register utilization, resulting in long combinational paths and low clock frequencies, while deeper pipelines are more complex and delayed. By exploring different configurations, a four-stage pipeline was found to balance a trade-off between performance and resource usage. As a result of this multi-stage arrangement, arithmetic-intensive functions can be distributed across stages to improve clock stability, and single-cycle throughput can be achieved once the pipeline is fully filled.

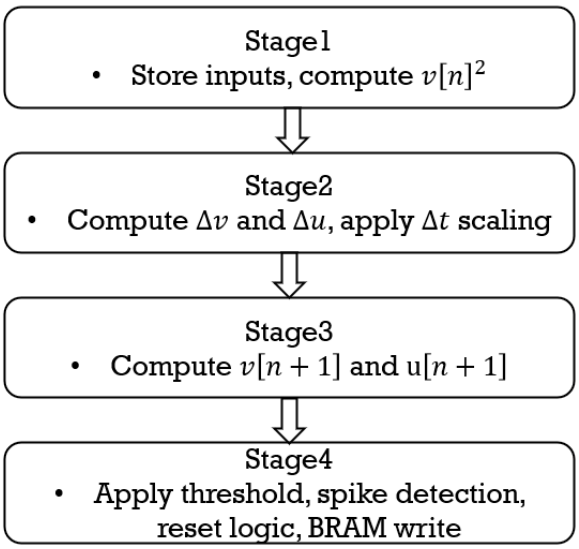


Figure 3.1: Pipeline stage allocation of four-stage pipeline architecture.

The stage partitioning is shown in Fig. 3.1, which follows the intrinsic structure of the model equations. At a high level, the four stages are organized as follows:

- **Stage 1: Input Processing and Quadratic Calculation**
  The first stage gets input current and model parameters, buffers the state variables, and evaluates the quadratic term of the membrane potential. This stage takes the computationally costly multiplication operation and maps it directly onto the DSP resources of the FPGA.

- **Stage 2: Derivative Computation**
  The second stage involves implementing the coupled derivatives of the membrane potential and recovery variable. Scaled coefficients and parameters are used here. This stage represents the mathematical core of the model.

- **Stage 3: State Integration**
  The third stage updates both state variables in parallel using the Euler integration method, so that the temporal evolution of both v and u is synchronized. This ensures biological coupling between the two state variables v and u without any latency mismatch.

- **Stage 4: Threshold and Reset Logic**
  The last stage carries out the spike detection and the reset operation, which results in the specific dynamics of the spiking behavior. When the firing threshold is met, the reset conditions are used, and analog membrane potential and digital spike outputs are generated. The new states are then rewritten to memory for use in the next iteration.

This partitioning facilitates the distribution of computations among each of the pipeline stages, eliminates bottlenecks, and stabilizes timing closure. After filling up the pipeline with an initial four-clock-cycle latency, the architecture can continue with one neuron update per clock cycle.

3.2.2 Detailed Pipeline Implementation

Hardware Scheduling

The hardware scheduling diagram presented in Fig. 3.2 demonstrates how the entire dynamics of the Izhikevich neuron is mapped to the proposed four-stage pipeline. This representation not only describes the computational dependencies of the equations, but also the temporal distribution of arithmetic operations over the pipeline depth. The figure highlights the parallelism of the model and illustrates the breakdown of mathematical expressions into hardware primitives, such as multipliers, adders, comparators, and multiplexers.

The scheduling diagram shows two significant computational paths. The left path corresponds to the membrane potential (v) update, in which the quadratic term $v[n] \times v[n]$ is computed simultaneously with the linear term $5\,v[n]$ and the constant offset (140). These results are carefully synchronised such that they converge at the adder network without causing any timing hazards. This decomposition is important because the quadratic term is the dominant nonlinear contributor to neuronal dynamics and thus must be computed both accurately and with low latency.

The right path corresponds to the recovery variable (u) computation. In this case, the product $b\,v[n]$ is calculated and coupled with the presently existing recovery variable $u[n]$ and parameter a to create the derivative $\frac{du}{dt}$. The diagram also encodes the discretization of time through the left-shift operation $<< 2^{\Delta t}$, which effectively applies the integration step in fixed-point arithmetic. This method does

not need expensive floating-point multipliers and uses the FPGA shift operation to scale effectively.

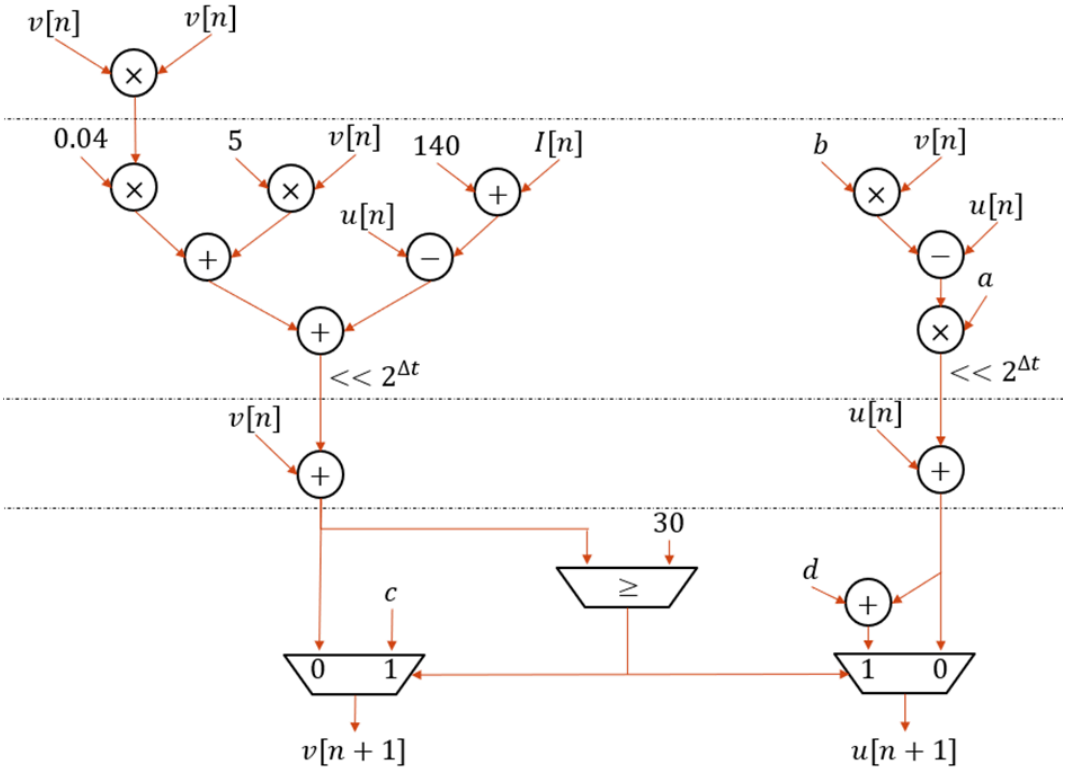


Figure 3.2: Scheduling diagram of hardware implementation of the Izhikevich neuron model.

The threshold detection and reset mechanism is shown at the bottom of Fig. 3.2. The updated membrane potential $v[n+1]$ is compared to the spike threshold of 30 mV. When it crosses the threshold, a control signal is produced and, it triggers the multiplexers to set the membrane potential to c and recovery variable to $u[n+1]+d$. This sudden update reflects the event-driven nature of spiking neurons and is done with minimal hardware to prevent possible pipeline delays.

Timing Relationships and Stage Balancing

The four stages of the pipeline, as presented in Fig. 3.2, are reflected by the horizontal boundaries, which explicitly define the temporal progression of intermediate results. There are three key design insights in this analysis:

- **Computational Balance:** Each stage is designed to maintain approximately a similar amount of arithmetic complexity, where Stage 2 is the most resource-demanding because of the accumulation of quadratic, linear, and

constant terms. The balance in computational weight avoids bottlenecks and guarantees high-frequency operation.

- **Data Dependency Management:** The explicit scheduling ensures that dependent variables (e.g., $v[n]^2$, bv) are generated at the appropriate cycle boundary to be used by the next stage, eliminating read-after-write hazards.
- **Resource Sharing:** Where possible, arithmetic resources are reused without violating throughput requirements. This reduces the usage of DSP slices while maintaining real-time functionality.

Control and Data Flow Graph

While the scheduling diagram emphasizes arithmetic synchronization, the control and data flow graph (CDFG) in Fig. 3.3 provides a holistic view of the system. It captures not only datapath operations but also control logic, memory management, and I/O interactions.

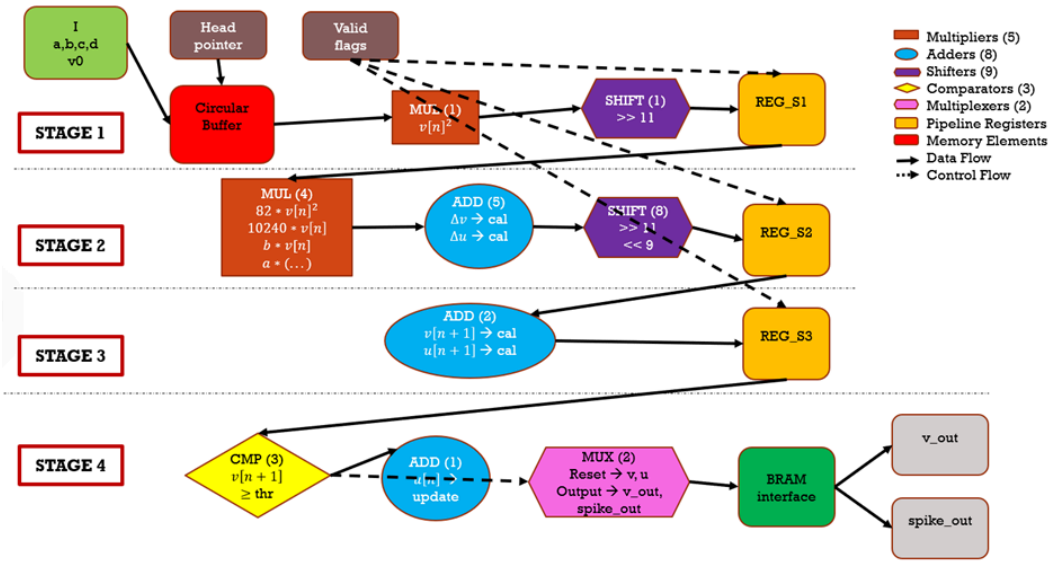


Figure 3.3: Control and data flow graph of hardware implementation of the Izhikevich neuron model.

On the left side of the diagram, the circular buffer subsystem is shown, which manages the neuron state variables and input currents with constant memory usage. The parameters (a, b, c, d) are fed into the pipeline via a special parameter interface. The head pointer logic controls the cyclic address of the buffer to provide deterministic timing, while the valid flags maintain synchronization during initialization and offer a hazard-free shutdown.

The central part of Fig. 3.3 elaborates the four stages of the pipeline, with specific hardware allocation:

- Stage 1 uses a DSP48E1 slice for $v[n]^2$ computation, fixed-point alignment via right shifts, and internal register REG_S1 for temporal isolation.
- Stage 2 holds the most computationally intensive operations, contains four multipliers, five adders, eight shifters, and pre-scaled coefficients (e.g., 82 for $0.04 * 20480$, 10240 for $5 * 20485$). This pre-scaling not only avoids multiplications of floating-point constants at runtime, but also exploits the fixed-point efficiency.
- Stage 3 performs the Euler integration of membrane potential and recovery variable simultaneously with the strict temporal alignment through internal register REG_S3.
- Stage 4 performs a threshold comparison, reset logic, and conditional selection of spike-driven updates, which is implemented by multiplexers. Outputs include both continuous-valued membrane potential (v_out) and binary spike (spike_out).

The right side of Fig. 3.3 illustrates the memory interface and data logging subsystems. Concurrent read/write of membrane potentials and spike events is done using dual-port BRAM. Automatic address allocation eliminates bus conflicts, and careful packing of 32-bit membrane potential values and 1-bit spikes ensures compact storage.

Resource Utilization and Performance Analysis

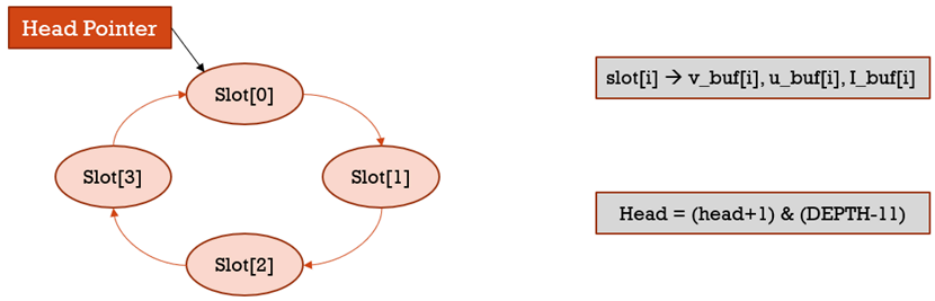
The control-data flow architecture ensures reasonably maximum efficiency with minimum redundancy:

- There are five multipliers that are spread over stages, which are strategically positioned to minimize critical path delays.
- Eight adders offer an adequate parallel accumulation without too much overhead.
- The nine shift operations perform scaling of fixed-point operations, which are used in place of floating-point multiplications.
- It only needs three comparators: a threshold detector and two conditional memory write logic comparators.
- Two multiplexers are used to implement conditional resets to ensure compact conditional logic.

This architecture produces a single neuron update per clock cycle and has an initial latency. The pipeline depth is selected to meet a reasonably minimum response time without timing closure of the FPGA. This ensures not only throughput that is suitable to large-scale neuromorphic systems but also full biological fidelity of the Izhikevich model.

3.2.3 Circular Buffer Memory Management

The circular buffer memory subsystem is an important architectural component that allows the effective synchronization of stages in a pipeline with a low memory overhead. The circular buffer provides a deterministic and constant-footprint memory scheme, unlike traditional memory allocation methods that utilize dynamically managed memory, which can lead to fragmentation or timing uncertainty (and thus are not ideally suitable for real-time simulation of neural systems) [72], [73].




```
Clock 1: head=0 → Stage1: slot[0]; Stage2:    -   , Stage3:    -   , Stage4: -
Clock 2: head=1 → Stage1: slot[1]; Stage2: slot[0], Stage3:    -   , Stage4: -
Clock 3: head=2 → Stage1: slot[2]; Stage2: slot[1], Stage3: slot[0], Stage4: -
Clock 4: head=3 → Stage1: slot[3]; Stage2: slot[2], Stage3: slot[1], Stage4: slot[0]
Clock 5: head=0 → Stage1: slot[0]; Stage2: slot[3], Stage3: slot[2], Stage4: slot[1]
......   REPEATS (circular wrap-around)
```

Figure 3.4: Circular buffer memory management structure (Depth = 4) of hardware implementation of the Izhikevich neuron model.

Buffer Architecture and Organization

The circular buffer has a fixed-depth architecture with DEPTH = 4 slots, each slot has a full set of neuron state variables. This depth is directly proportional to the four-stage pipeline architecture, with each stage of the pipeline acting on a different temporal instance without conflict or data hazard. Every buffer slot holds three data items: the membrane potential state variable $v\_buf[i]$ in Q5.11 fixed-point format, the recovery variable state $u\_buf[i]$ in Q5.11 fixed-point format, and the input current value $I\_buf[i]$ for the corresponding simulation time step. The buffer slots are addressed sequentially between 0 and 3, forming a circular addressing

space that effectively utilizes hardware resources and achieves the required temporal isolation to run the pipeline.

Head Pointer Mechanism

The head pointer system provides the basic timing control for the circular buffer operation. The head pointer implements a modulo-4 counter, which steps forward every clock cycle, producing the circular addressing pattern required for continuous operation. The head pointer progresses as Head = (head + 1) & (DEPTH − 1), where this bitwise AND operation with (DEPTH − 1) = 3 results in an efficient modulo-4 counter that wraps around through addresses $0 \to 1 \to 2 \to 3 \to 0$, thus achieving the desired circular behavior without costly division operations.

Stage-to-Buffer Mapping

The operands of each pipeline stage are stored at an offset in the buffer, thus maintaining temporal alignment:

- Stage 1 reads the stage variables of current state (idxs1 = head).
- Stage 2 accesses idxs2 = (head + 3) & 3, corresponding to the previous time step.
- Stage 3 retrieves idxs3 = (head + 2) & 3.
- Stage 4 completes the updates based on idxs4 = (head + 1) & 3.

This type of indexing is offset-based and thus avoids pipeline hazards since every stage works on temporally consistent data.

The timing diagram of buffer operation is shown in Fig. 3.4. The initial four cycles represents pipeline filling, where later stages remain idle. From cycle 5 onwards, steady-state operation is reached: every cycle generates a neuron update, and all stages remain active and time-synchronized.

3.3 Hardware Architecture and Co-design Framework

The mathematical model of the Izhikevich neuron is implemented as a hardware instance through a carefully structured architecture, where computational, memory, and control subsystems are combined into a single FPGA-based platform. The block diagram of the entire hardware architecture is shown in Fig. 3.5. It illustrates the interconnection between the Zynq-7000 SoC processing system (PS), the programmable logic (PL) that implements the neural computation core, and the memory subsystem. This architecture forms the basis of a real-time neuromorphic computing framework that is capable of providing high-performance neural updates and flexible experimental control.

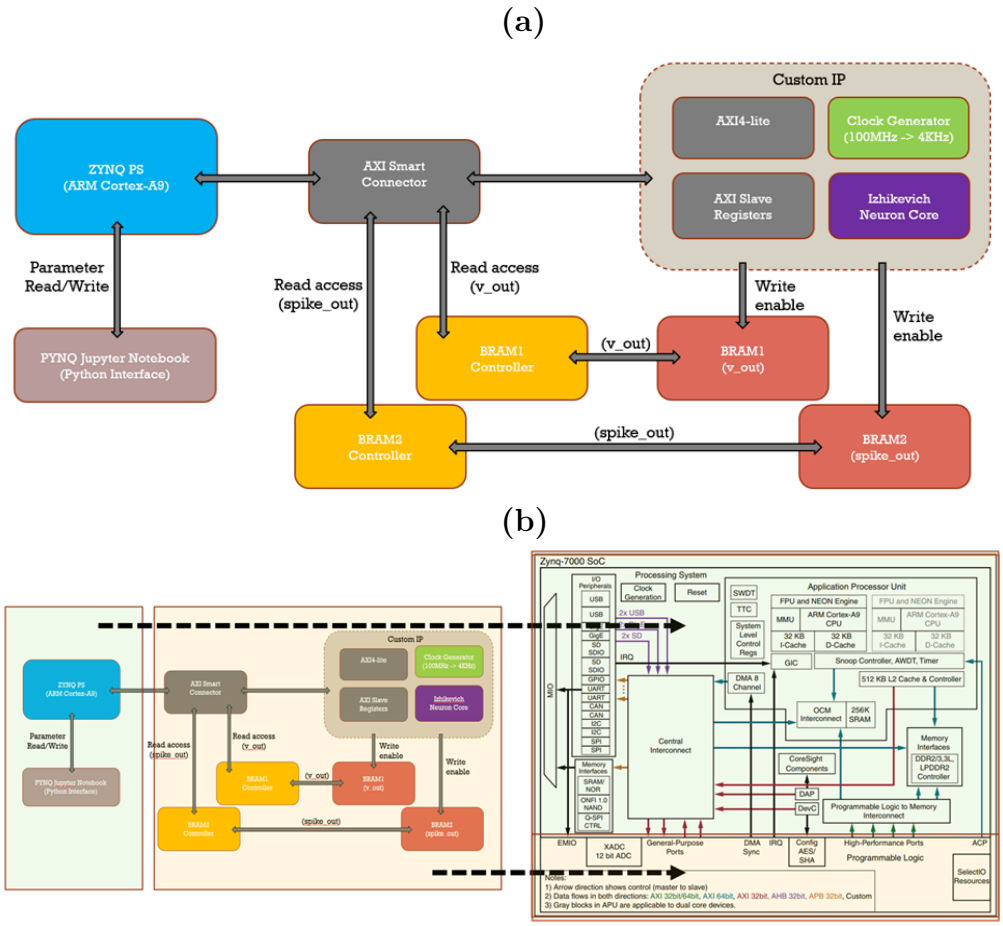


Figure 3.5: Hardware implementation of Izhikevich neuron model: (a) Complete hardware architecture block diagram of a single core, (b) Mapping PS and PL of Zynq-7000 SoC with hardware architecture block diagram.

3.3.1 Hardware Architecture Block Diagram

The proposed system architecture uses a heterogeneous computing approach that fully utilizes the dual-nature of the Zynq-7020 device, consisting of a high-level ARM Cortex-A9 Processing System (PS) and a Programmable Logic (PL) fabric. The system partitioning ensures flexibility and computational efficiency by dividing tasks based on what each part does best. The PS typically handles parameter management, data transfer, and experimental control, while the PL fabric manages the precise neural computations [65].

System Architecture Overview

The entire hardware architecture is a combination of four large and interconnected subsystems. The Zynq Processing System (PS) uses the Linux operating system to provide high-level software control, parameter setup, and experimental data management. The Izhikevich Neural Computation Core is a custom IP block that uses the four-stage pipelined neuron model, specifically designed to run in Q5.11 fixed-point arithmetic and real-time at biological frequencies. AXI4-Lite Communication Interface is an implementation of a standardized memory-mapped structure of registers, which allows an efficient update of parameters and two-way data exchange between the PS and PL space. The Block RAM Subsystem allows storing membrane potential and spike events by using a dual-port memory architecture that supports read and write operations simultaneously.

This modular partitioning strategy enables individual subsystems to be optimized independently while ensuring smooth integration across the entire platform. The architecture is based on best practices in System-on-Chip (SoC)-based hardware-software co-design methodology, which guarantees both performance efficiency and development flexibility. The architecture offers a scalable platform that can be further extended to take on multi-neuron implementations.

Neural Computation Core Architecture

The neural computation core is the main processing unit of the architecture, which includes the four-stage pipelined architecture implementation described in Subsection 3.2. The core is packaged as a reusable IP module with 32-bit interfaces. The module takes all the necessary parameters of neurons, such as the four Izhikevich coefficients (a, b, c, d), initial membrane potential value, and input currents that change with time, and produces both continuous membrane potential outputs and discrete binary spike events. The core also has some critical architectural features, which improve its performance and functionality. The circular buffer memory management ensures the pipeline always runs by maintaining four parallel state instances corresponding to the four stages of the pipeline, removing pipeline stalls, and providing data flow into the pipeline. The Q5.11 fixed-point arithmetic is used

to maintain biological accuracy while using fewer hardware resources than floating-point designs. Introduced BRAM interfaces allow neural activity data to be logged directly without additional computational load on the central processor, supporting high-performance real-time processing without slowing down long-duration experiments.

Its neural computation core runs at a very precisely maintained simulation frequency of 4 kHz, matching the integration timestep of many biological systems, which is 0.25 ms. This frequency selection is also a tradeoff point that balances computation accuracy with hardware efficiency. It ensures that all twenty canonical Izhikevich neuron behaviors are implemented accurately in hardware. The core maintains consistent timing that supports both single-neuron and extended multi-neuron network implementations.

Communication Infrastructure and Protocol Implementation

The neural computation core communicates with the ARM processor by means of an extensive AXI4-Lite slave interface wrapper that offers standardized communication protocols. This wrapper introduces a well-structured collection of eleven 32-bit slave registers, which are mapped directly into the PS address space to allow software applications to conduct easy memory-mapped input/output activities. The register organization offers a systematic interface to all necessary system elements, such as model parameters and initial conditions, control signals to reset operations, and control of simulation duration, dynamic input capabilities of real-time external current updates, and read-only status reports of updated membrane potential values and spike detection flags.

The register-mapped interface architecture allows advanced memory-mapped access to I/O by software applications with strict compliance to the ARM AMBA AXI protocols. The interface design allows for updating parameters individually with fine-grained control and batched operations with high-throughput configuration, providing an optimal balance between system configurability and data throughput. This architecture offers interactive experimental situations with real-time responsiveness and protocol compliance, which is required to integrate with the standard Xilinx development tools and IP ecosystem.

Clock Domain Management and Synchronization

A major architectural issue in the hardware design was bridging the gap between the high-frequency 100 MHz system clock domain and the significantly slower 4 kHz simulation frequency needed to match the dynamics of biological neurons. This bridging in the frequency domain is achieved by a clock division scheme, which is implemented in hardware and directly in the AXI wrapper module. An accurately programmed 16-bit counter produces the neural simulation clock based on

the mathematical expression $f\_neural = f\_system/(2x(12499 + 1))$ which gives a stable 4 kHz neural clock signal with a low jitter in timing and good frequency fidelity.

To ensure fully reliable signal interactions among the disparate clock domains, the architecture uses robust two-stage flip-flop synchronizers on all critical control and data signals that must cross domain boundaries. This synchronization method helps avoid metastability conditions that may jeopardize the system's reliability without affecting signal integrity across all cross-domain communications. The clock domain management system provides a firm basis for deterministic timing behavior, which is critical for the proper simulation of biological neural processes and experimental reproducibility.

Memory Subsystem and Data Logging Architecture

The data logging and data storage are provided by the advanced dual Block RAM (BRAM) system, which offers the complete tracking of continuous and discrete neural signals. BRAM1 is specifically used to store 32-bit traces of membrane potential in the fixed-point format Q5.11, ensuring full temporal resolution of the analog neural dynamics. BRAM2 is efficient in storing small 1-bit spike events with associated high-precision timing data, providing the opportunity to study discrete neural firing patterns in detail.

The two BRAM modules are both in True Dual Port (TDP) mode, that is, they can be written to and read independently by the neural computation core and the processing system, respectively. This two-port design does not cause memory bandwidth contention and ensures continuous logging at the maximum 4 kHz update rate over long experimental periods. The memory subsystem includes automatic address incrementing features, programmed by configurable parameters, to allow fine and coarse-grained memory organization strategies that suit various experimental data acquisition needs.

3.3.2 Co-design Framework

The hardware-software co-design framework is an essential architectural element that fills the gap between high-performance FPGA acceleration and the software-based experimental control. This framework uses the PYNQ platform to provide simple Python interfaces while keeping full access to the hardware's capabilities.

PYNQ Platform Integration and Software Architecture

The implementation of the co-design framework is based on the PYNQ (Python Productivity for Zynq) platform that offers extensive Python APIs and integrated Jupyter notebooks that allow a user to configure experimental parameters, control

neural simulations, and analyze the results without the need to make any low-level hardware programming adjustments or Register Transfer Level (RTL) understanding. The PYNQ integration provides a smooth interface to the complexity of the FPGA programming and maintains full access to hardware performance and functionality.

The software architecture offers advanced parameter control features with Memory-Mapped I/O (MMIO) interfaces, which enable automatic register updates. Data acquisition subsystems use BRAM interfaces to transfer continuous neural data at a full 4 kHz sampling rate, and automatic hardware-to-software conversion of data types between the fixed-point representation, Q5.11, and the IEEE-754 floating-point representation commonly used in standard scientific computing libraries. The integration of the experiment's workflow using Jupyter-based notebooks provides a comprehensive set of tools that facilitate the integration of parameter setting, simulation execution, real-time data recording, and post-processing analysis into a fully reproducible experimental framework.

Vivado Integration and IP Development Methodology

The entire hardware architecture is implemented in Xilinx Vivado IP Integrator that allows easy packaging of custom computation cores and AXI wrapper modules into reusable Intellectual Property (IP) blocks that can be incorporated into more complex system designs. Vivado block design methodology also has several key elements, such as the Zynq PS configuration with DDR3 memory interface, Ethernet connection, USB connection, and numerous AXI master ports to connect the system together. The AXI SmartConnect offers an optimized interconnect fabric that handles data routing and protocol translation between the various system components. The Izhikevich IP Core, with a built-in AXI4-Lite wrapper, serves as the central computing unit. At the same time, the data storage infrastructure is represented by two Block Memory Generators with independent BRAM controllers.

This holistic integration approach enables the reuse of IP blocks in other projects, supports modular testing and verification methods, and facilitates advanced version control techniques that guarantee design reproducibility and collaborative development. The process offers excellent scalability properties, which can be extended to bigger and more intricate neuromorphic computing systems without compromising development efficiency and design quality.

Design Flow and Deployment Methodology

The hardware-software co-design approach is based on the well-designed development and implementation cycle that provides a high quality of system integration and stable performance features. It starts with thorough functional verification on

the Register Transfer Level (RTL) with detailed pipeline verification and extensive memory interface testing to verify that the system is acting correctly under all possible operating conditions. Both the high-frequency 100 MHz system domain and the lower-frequency 4 kHz neural simulation domain are synthesized and placed with timing closure validation, and all timing constraints are satisfied with sufficient safety margins.

Bitstream generation and deployment is done using the mechanism of PYNQ overlay which enables dynamic reconfiguration and easier hardware deployment processes. The overlay discovery process automatically exposes all hardware blocks as Python objects which implement runtime control interfaces, which means that no low-level hardware access programming or driver development is required. The workflow provides a strong system integration and deterministic performance properties as well as allowing a seamless migration of the Vivado design space to real-time experimental execution platforms.

Experimental Control and Analysis Framework

The entire co-design system offers a comprehensive experimental platform that can facilitate the advanced neuromorphic computing research and development process. The capability of real-time monitoring can be used to track membrane dynamics and spike train generation with full temporal resolution and biological accuracy. The framework facilitates adjustable experimentation parameters, such as experiment runtime, arbitrary input current profiles, and systematic exploration of the parameter space with automated sweep functionalities.

Data management and analysis integration automatically formats experimental results into standard NumPy and Pandas data structures. These provide direct compatibility with modern Python scientific computing libraries. High-resolution temporal logging can record continuous membrane potential data and discrete spike event data with accurate timing information at the entire 4 kHz sampling rate. The framework supports both interactive experimental exploration of research and development activities and automated experimental protocols for systematic characterization and validation studies.

Chapter 4

Implementation, Results and Discussion

This chapter presents comprehensive experimental validation of the FPGA-based Izhikevich neuron implementation through systematic analysis of hardware performance, numerical accuracy, and practical applicability. The investigation progresses from fundamental single-neuron characterization to advanced multi-neuron architectures, culminating in real-world signal processing demonstrations.

The experimental methodology begins with single-neuron implementation analysis in Section 4.1, where we establish baseline performance metrics and validate biological accuracy. Subsection 4.1.1 examines FPGA resource utilization, timing characteristics, and power consumption. Subsection 4.1.2 investigates the precision of Q5.11 fixed-point arithmetic through comparative analysis between floating-point and fixed-point Python simulations. Subsection 4.1.3 demonstrates the capability of our implementation to reproduce all 20 characteristic Izhikevich spiking patterns, validating complete biological functionality. Building upon the single-neuron foundation, Section 4.2 explores time-multiplexed virtual neuron architectures, demonstrating resource-efficient scaling by implementing multiple virtual neurons using a single physical hardware core. Section 4.3 investigates parallel multi-neuron implementations designed for high-throughput operation. Section 4.4 presents a comparative analysis of resource utilization across all implemented architectures. Section 4.5 conducts a comprehensive frequency analysis for single-neuron and 4-neuron cores. Finally, Section 4.6 demonstrates the practical applicability of the approach by processing real-world IMU sensor data from the WISDM dataset.

4.1 Single Neuron Implementation

The single-neuron implementation serves as the fundamental building block for all subsequent multi-neuron architectures. Following the design methodology described in Section 3.3, we implemented the Izhikevich neuron core using Vivado's comprehensive hardware-software co-design workflow, which encompasses RTL verification, synthesis, implementation, and bitstream generation. The generated bitstream establishes the hardware-software interface between the FPGA fabric and the PYNQ Jupyter notebook environment.

The experimental framework leverages the AXI4-Lite memory-mapped interface with fixed addressing to control and monitor neural behavior. We configured neural parameters (a, b, c, d) and the input current (I_in) by writing to dedicated slave registers defined in the AXI4-Lite wrapper module. The dual-BRAM architecture enables continuous data acquisition: BRAM1 stores membrane potential values (v_out) at a 4 kHz sampling rate, while BRAM2 captures binary spike events (spike_out) with precise timing information.

Data retrieval and analysis are performed through Python-based Jupyter notebooks, where we read the BRAM contents and conduct real-time visualization and post-processing analysis. Systematic parameter tuning across the four-dimensional parameter space (a, b, c, d) enabled exploration of all 20 characteristic Izhikevich spiking patterns, validating the biological completeness of our hardware implementation.

The following subsections present a detailed analysis of resource utilization (4.1.1), fixed-point arithmetic precision (4.1.2), and comprehensive spiking pattern reproduction (4.1.3), establishing quantitative benchmarks for single-neuron performance.

4.1.1 Resource Utilization, Timing and Power

We characterized the hardware implementation using Vivado's post-implementation reports, analyzing resource consumption, timing constraints, and power dissipation for a single Izhikevich neuron core configured for tonic spiking behavior.

FPGA Resource Utilization

Figure 4.1 presents the overall resource utilization on the Zynq-7020 device. The complete system consumes 13.95% of available LUTs (7,422/53,200), 4.38% of LUTRAM (762/17,400), 7.75% of flip-flops (8,241/106,400), 5.71% of BRAM (8/140), and 7.27% of DSP slices (16/220). These modest utilization percentages indicate significant headroom for multi-neuron scaling while maintaining efficient resource usage.

Resource	Utilization	Available	Utilization %
LUT	7422	53200	13.95
LUTRAM	762	17400	4.38
FF	8241	106400	7.75
BRAM	8	140	5.71
DSP	16	220	7.27

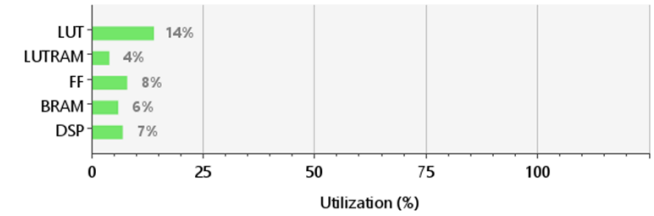


Figure 4.1: Overall FPGA resource utilization for single Izhikevich neuron core implementation on Zynq-7020.

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)
izh_single_wrapper	7422	8241	3066	6660	762	8	16	130	2
izh_single_i (izh_single)	7422	8241	3066	6660	762	8	16	0	2
axi_bram_ctrl_0 (izh_single_axi_bram_ctrl_0	170	184	70	170	0	0	0	0	0
axi_bram_ctrl_1 (izh_single_axi_bram_ctrl_0	170	184	79	170	0	0	0	0	0
axi_smc (izh_single_axi_smc_0)	4636	4924	1629	3879	757	0	0	0	0
blk_mem_gen_0 (izh_single_blk_mem_gen_	7	10	7	5	2	4	0	0	0
blk_mem_gen_1 (izh_single_blk_mem_gen_	7	10	6	5	2	4	0	0	0
izh_axi_0 (izh_single_izh_axi_0_0)	2422	2896	1330	2422	0	0	16	0	1
inst (izh_single_izh_axi_0_0_izh_axi)	2422	2896	1330	2422	0	0	16	0	1
izh_axi_slave_lite_v1_0_S00_AXI_inst	2422	2896	1330	2422	0	0	16	0	1
uut (izh_single_izh_axi_0_0_izh)	1810	2286	1216	1810	0	0	16	0	0
processing_system7_0 (izh_single_processi	0	0	0	0	0	0	0	0	1
rst_ps7_0_100M (izh_single_rst_ps7_0_100M	17	33	12	16	1	0	0	0	0

Figure 4.2: Hierarchical resource utilization breakdown of a single Izhikevich neuron core implementation on Zynq-7020, showing component-level consumption within the complete design.

Figure 4.2 provides a hierarchical resource breakdown, revealing that the core Izhikevich computation module (‘uut’) consumes only 1,810 LUTs, 2,286 FFs, and 16 DSP slices with zero BRAM usage. The dual-BRAM blocks (‘blk_mem_gen_0’) and (‘blk_mem_gen_1’) each utilize 4 BRAM tiles for data logging. Notably, the AXI SmartConnect (‘axi_smc’) infrastructure accounts for the majority of interconnect resources (4,636 LUTs, 4,924 FFs), highlighting the overhead of processor-FPGA communication while demonstrating the lightweight nature of the neural computation core itself.

Timing Analysis

Figure 4.3 shows the static timing analysis results demonstrating successful timing closure. The Worst Negative Slack (WNS) of 1.361 ns and Worst Hold Slack (WHS) of 0.018 ns indicate positive timing margins with zero failing endpoints across all 20,348 timing paths. The Worst Pulse Width Slack (WPWS) of 3.750 ns confirms robust clock signal integrity. These positive slack values confirm that the design operates reliably at the 100 MHz system clock frequency, with sufficient safety margins for process, voltage, and temperature variations.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.361 ns	Worst Hold Slack (WHS):	0.018 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	20348	Total Number of Endpoints:	20348	Total Number of Endpoints:	7130

Figure 4.3: Static timing analysis report of a single Izhikevich neuron core implementation on Zynq-7020.

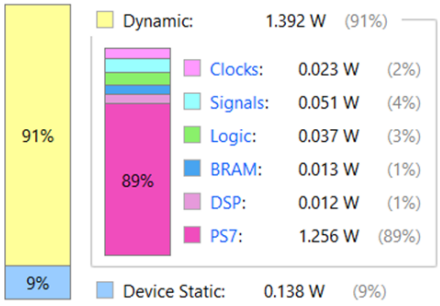


Figure 4.4: On-chip power consumption breakdown of a single Izhikevich neuron core implementation on Zynq-7020.

Power Consumption

Figure 4.4 presents an on-chip power analysis, revealing a total consumption of 1.53 W, partitioned into 1.392 W of dynamic power (91%) and 0.138 W of static power (9%). The Zynq Processing System (PS7) consumes the most power at 89% (1.256 W), primarily due to the ARM Cortex-A9 cores and their associated peripherals. The programmable logic components contribute minimally: DSP blocks (0.012 W, 1%), BRAM (0.013 W, 1%), logic (0.037 W, 3%), signals (0.051 W, 4%), and clocks (0.023 W, 2%). This power breakdown indicates that the neural computation fabric itself is highly power-efficient, with the majority of power budget consumed by the control and interface infrastructure.

4.1.2 Fixed-Point Arithmetic Precision Analysis

The Q5.11 fixed-point representation provides hardware efficiency at the cost of quantization error compared to IEEE 754 floating-point arithmetic. We conducted comprehensive precision analysis by comparing Python-based floating-point and fixed-point simulations of the Izhikevich neuron model, quantifying numerical deviations through multiple error metrics.

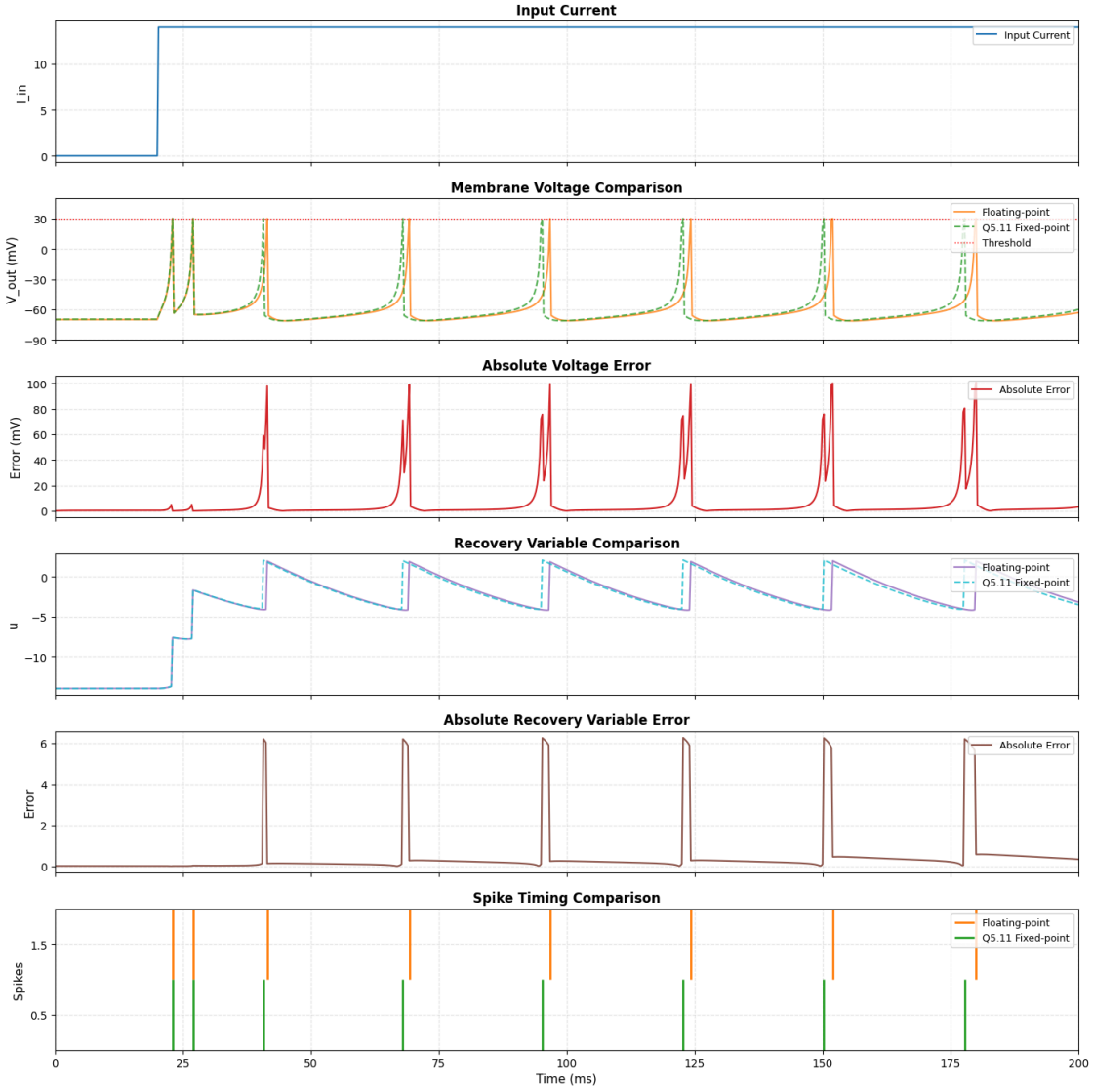


Figure 4.5: Comprehensive time-domain comparison of floating-point versus Q5.11 fixed-point implementations for tonic spiking pattern using Python simulation platform.

Comparative Simulation Analysis

Figure 4.5 presents a time-domain comparison for tonic spiking behavior under constant input current. The six-panel visualization reveals: (1) input current stimulus, (2) membrane potential trajectories for both implementations, (3) absolute voltage error, (4) recovery variable dynamics, (5) absolute recovery variable error, and (6) spike timing comparison. The overlapping traces demonstrate excellent agreement during the initial 30 ms, with deviations gradually accumulating over extended simulation periods due to finite precision effects and error propagation through iterative computations.

Error Quantification Metrics

We employed five standard error metrics to quantify numerical accuracy for both membrane potential (v) and recovery variable (u) over two simulation durations: 30 ms (short-term) and 200 ms (long-term).

Root Mean Square Error (RMSE) measures the standard deviation of prediction errors [29]:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{\text{float}}[i] - x_{\text{fixed}}[i])^2} \tag{4.1}$$

where N is the number of samples, $x_{\text{float}}[i]$ represents floating-point values, and $x_{\text{fixed}}[i]$ denotes fixed-point values.

Normalized Root Mean Square Error (NRMSE) provides scale-independent error assessment [29]:

$$\text{NRMSE}(\%) = \frac{\text{RMSE}}{\max(x_{\text{float}}) - \min(x_{\text{float}})} \times 100 \tag{4.2}$$

This normalization enables comparison across signals with different amplitude ranges.

Mean Absolute Error (MAE) quantifies average absolute deviation [29]:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_{\text{float}}[i] - x_{\text{fixed}}[i]| \tag{4.3}$$

MAE is less sensitive to outliers compared to RMSE, providing complementary error characterization.

Correlation Coefficient (Corr) assesses linear relationship strength [53]:

$$\text{Corr} = \frac{\sum_{i=1}^{N}(x_{\text{float}}[i] - \bar{x}_{\text{float}})(x_{\text{fixed}}[i] - \bar{x}_{\text{fixed}})}{\sqrt{\sum_{i=1}^{N}(x_{\text{float}}[i] - \bar{x}_{\text{float}})^2}\sqrt{\sum_{i=1}^{N}(x_{\text{fixed}}[i] - \bar{x}_{\text{fixed}})^2}} \tag{4.4}$$

where $\bar{x}$ denotes the mean value. Perfect correlation yields Corr = 1.0.

Table 4.1: Quantitative precision analysis comparing Q5.11 fixed-point and IEEE 754 floating-point implementations in Python using RMSE, NRMSE, MAE, and Corr error metrics.

Error Metric	**30 ms Simulation Time**		**200 ms Simulation Time**	
	(v)	(u)	(v)	(u)
RMSE	0.845	0.015	16.278	1.303
NRMSE (%)	0.845	0.122	16.062	8.140
MAE	0.464	0.014	5.474	0.459
Corr (%)	0.001	–	1.125	–

Quantitative Results

Table 4.1 summarizes error metrics for 30 ms and 200 ms simulation durations. For the 30 ms short-term case, membrane potential achieves RMSE = 0.845 mV, NRMSE = 0.845%, MAE = 0.464 mV, and negligible correlation error (0.001%). Recovery variable shows even superior accuracy with RMSE = 0.015, NRMSE = 0.122%, and MAE = 0.014. These results demonstrate that Q5.11 fixed-point arithmetic maintains biological accuracy for short-duration neural simulations.

For the 200 ms extended simulation, accumulated quantization errors become apparent: membrane potential RMSE increases to 16.278 mV (NRMSE = 16.062%, MAE = 5.474 mV, Corr = 1.125%), while recovery variable exhibits RMSE = 1.303 (NRMSE = 8.140%, MAE = 0.459).

Despite the accumulation of errors over time, the Q5.11 representation maintains sufficient accuracy for typical neuromorphic computing applications, including burst detection, rate coding, and short-term temporal patterns, thereby validating our fixed-point design choice for hardware efficiency without compromising functional correctness.

4.1.3 Complete Spiking Pattern Reproduction

The comprehensive validation of our FPGA implementation requires demonstrating biological completeness by reproducing all 20 characteristic Izhikevich spiking patterns. This subsection presents a systematic comparison across three implementation platforms: floating-point Python simulation, fixed-point Verilog simulation, and physical FPGA hardware implementation, establishing functional equivalence while quantifying temporal deviations introduced by fixed-point arithmetic and pipeline delays.

Pattern 1: Tonic Spiking

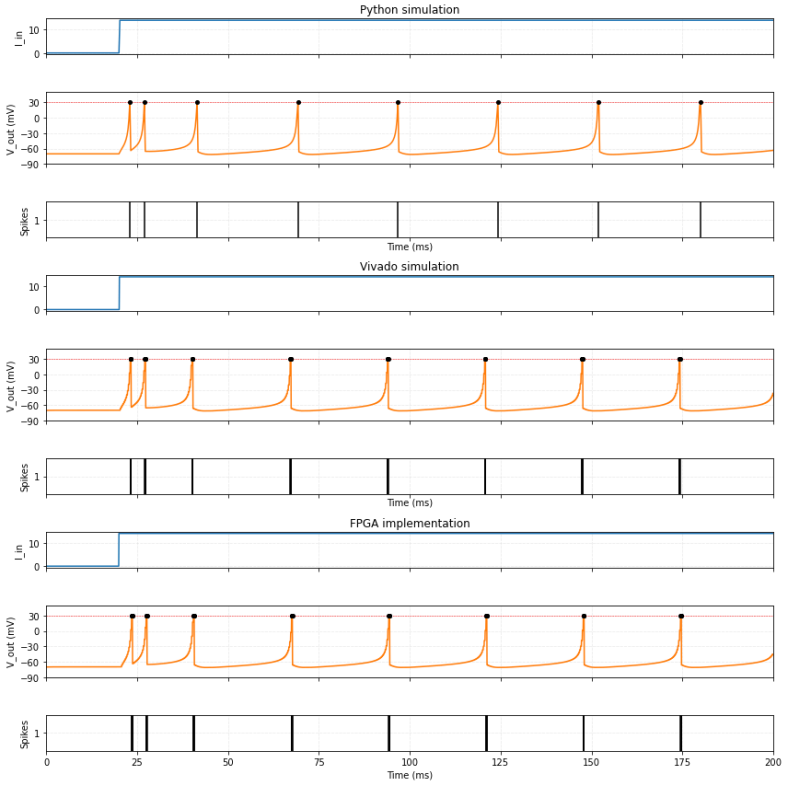


Figure 4.6: Tonic spiking pattern comparison across three platforms. A three-panel visualization shows (from top to bottom): Python floating-point simulation, Vivado fixed-point simulation, and FPGA hardware implementation.

Parameters: $a = 0.02$, $b = 0.2$, $c = -65$, $d = 6$ | Stimulus: Step DC current

Tonic spiking represents the most fundamental excitable behavior in cortical neurons, characterized by sustained periodic firing in response to constant suprathreshold stimulation [15], [16]. Figure 4.6 demonstrates successful reproduction across all three platforms (Python, Vivado, FPGA), with membrane potential trajectories exhibiting consistent spike amplitude (∼30 mV peak), afterhyperpolarization to −65 mV, and regular inter-spike intervals of approximately 25 ms (40 Hz firing rate). The Python and Vivado simulations show near-perfect temporal alignment during the initial 50 ms, with spike events occurring at identical time points. The FPGA hardware implementation maintains precise pattern fidelity with observable time shifts of 1–2 ms emerging after 100 ms due to accumulated quantization errors. Despite this temporal drift, spike frequency remains constant at ∼40 Hz across all implementations, confirming rate-coding accuracy essential for neuromorphic applications.

Pattern 2: Phasic Spiking

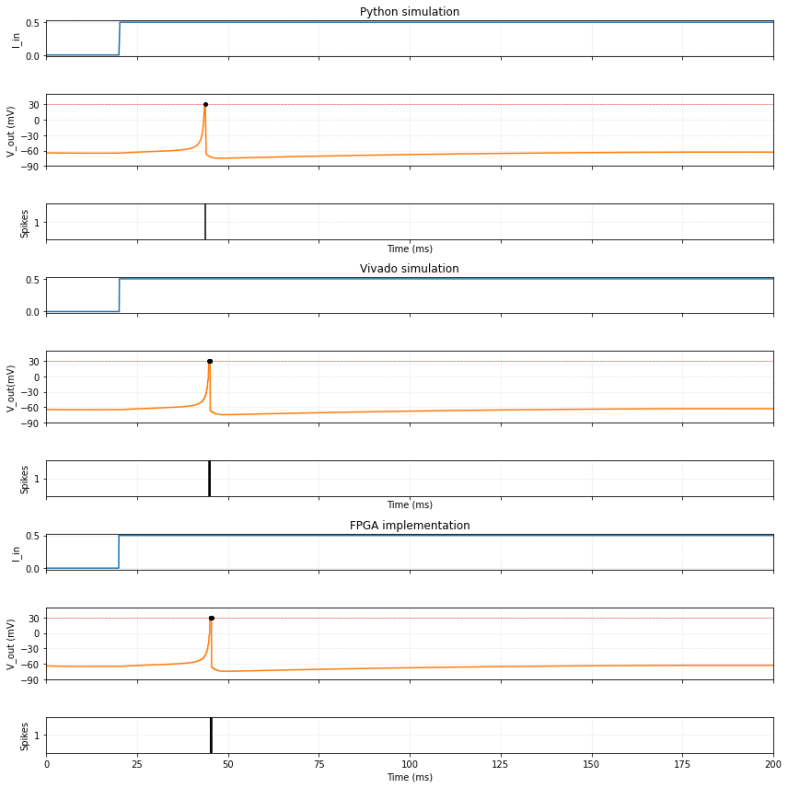


Figure 4.7: Phasic spiking pattern showing a single spike at stimulus onset across three platforms: Python, Vivado, and FPGA implementations.

Parameters: $a = 0.02$, $b = 0.247$, $c = -65$, $d = 6$ | Stimulus: Step DC current

Phasic spiking demonstrates transient excitability where the neuron fires a single spike at stimulus onset and returns to quiescence despite sustained input [15], [16]. Figure 4.7 shows precise reproduction across all three platforms, with spike generation occurring at $\sim$45 ms post-stimulus and subsequent membrane potential stabilization near -60 mV without further spiking activity throughout the 200 ms observation period. All three implementations exhibit identical spike timing at 45 ms with matching spike amplitude and post-spike hyperpolarization dynamics. The FPGA hardware maintains perfect fidelity to the quiescent state after the initial spike, demonstrating accurate threshold detection and sustained subthreshold stability. No temporal drift is observable in this pattern due to the single-spike nature, eliminating accumulated quantization errors from iterative spiking.

Pattern 3: Tonic Bursting

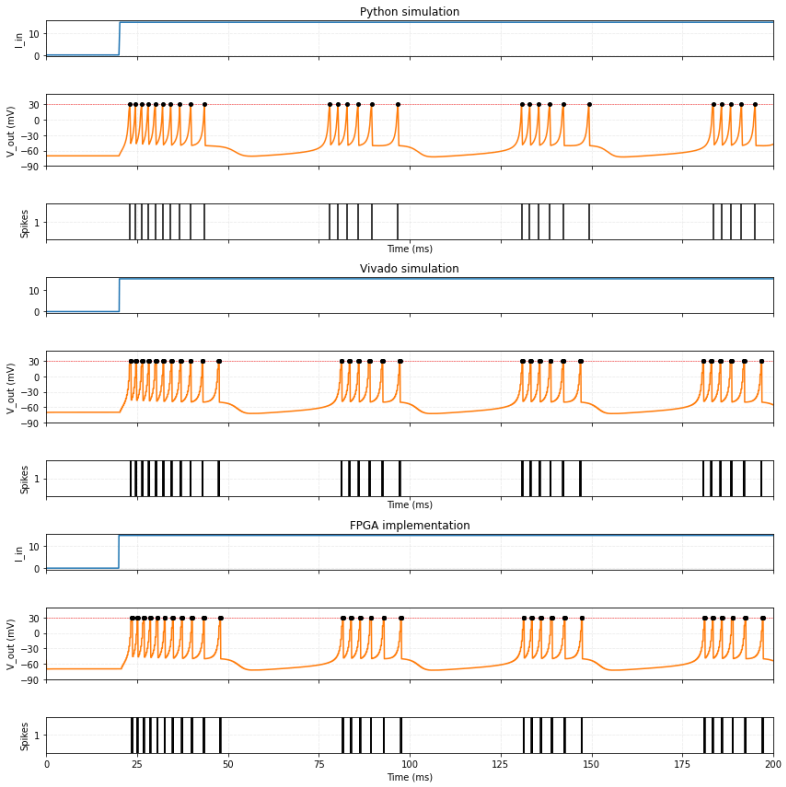


Figure 4.8: Tonic bursting pattern showing periodic spike clusters across Python, Vivado, and FPGA implementations.

Parameters: $a = 0.02$, $b = 0.2$, $c = -50$, $d = 2$ | Stimulus: Step DC current

Tonic bursting exhibits sustained rhythmic clusters of rapid spikes separated by quiescent interburst intervals, representing a fundamental oscillatory firing mode observed in cortical networks [15], [16]. Figure 4.8 demonstrates four complete burst cycles over 200 ms, each burst containing 6–11 spikes at ∼100 Hz intraburst frequency, followed by hyperpolarized silent periods. The interburst frequency of ∼25 Hz matches typical gamma-band oscillation generators. All three platforms reproduce burst structure with high fidelity: 8–10 spikes per burst, consistent intraburst frequency (∼100 Hz), and regular burst periodicity (∼40 ms interburst interval). The FPGA implementation shows minor temporal shifts accumulating to 2–3 ms by 200 ms, with slight variations in spike count at the first burst (10 vs. 11 spikes) due to threshold crossing sensitivity under fixed-point quantization. Critically, burst frequency and overall pattern structure are preserved, validating the reproduction of oscillatory dynamics.

Pattern 4: Phasic Bursting

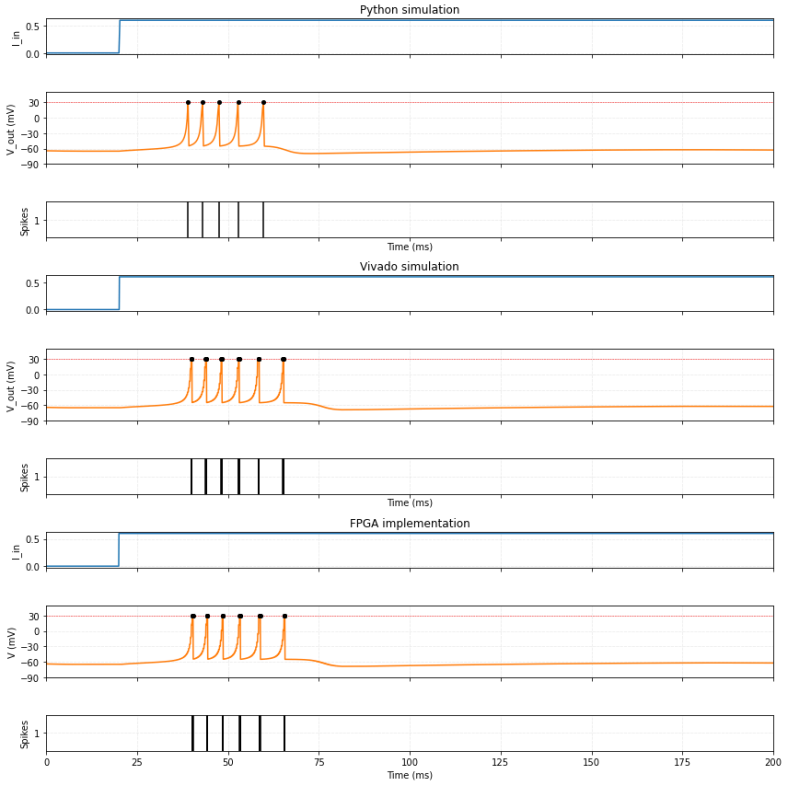


Figure 4.9: Phasic bursting pattern showing single transient spike cluster at stimulus onset across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.02$, $b = 0.247$, $c = -55$, $d = 0.05$ | Stimulus: Step DC current

Phasic bursting combines transient response characteristics with burst-mode signaling, producing a single cluster of 5–6 rapid spikes at stimulus onset followed by sustained quiescence [15], [16]. Figure 4.9 demonstrates the burst occurring at ∼35–60 ms with intraburst spike frequency of ∼100 Hz, after which the membrane potential stabilizes near −65 mV without further activity despite continued stimulation. All three platforms reproduce the single-burst transient response with matching spike count (5–6 spikes), consistent burst duration (∼25 ms), and identical quiescent behavior post-burst. The FPGA implementation shows perfect temporal alignment with Vivado simulation, with <1 ms deviation from Python results. The small recovery parameter ($d = 0.05$, significantly reduced from typical values) prevents burst repetition by minimizing recovery variable reset, enabling the phasic characteristic. Spike timing within the burst maintains high precision across platforms, validating event-based communication fidelity.

Pattern 5: Mixed Mode

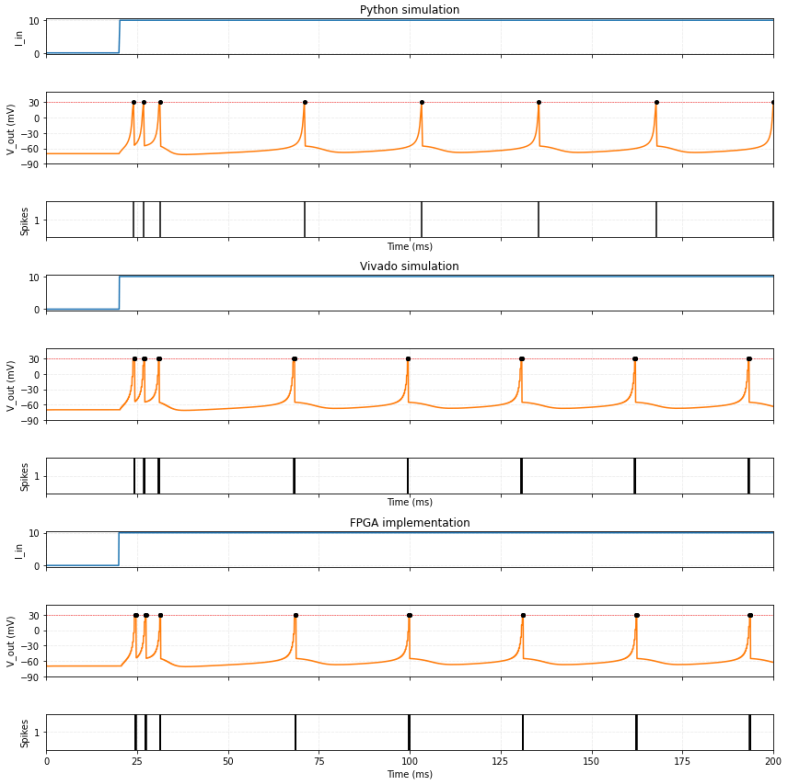


Figure 4.10: Mixed-mode pattern combining phasic burst onset with sustained tonic spiking across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.02$, $b = 0.2$, $c = -55$, $d = 4$ | Stimulus: Step DC current

Mixed mode demonstrates hybrid temporal coding where an initial phasic burst ($\sim$3 spikes at 25–35 ms) signals stimulus onset, followed by transition to tonic spiking ($\sim$30 Hz) that encodes sustained input magnitude [15], [16]. Figure 4.10 shows the characteristic biphasic response: high-frequency burst onset detection followed by rate-coded sustained activity, combining both transient edge detection and steady-state magnitude representation in a single neuron. All three platforms accurately reproduce the mode transition: initial 3-spike burst at 25–35 ms followed by regular spiking at $\sim$30–35 ms intervals. The FPGA implementation maintains burst structure fidelity and correctly transitions to periodic firing with consistent interspike intervals. Minor temporal drift of 1–2 ms accumulates during the tonic phase (100–200 ms), while the critical burst-to-spike transition timing shows $<$0.5 ms deviation, validating accurate parameter-dependent mode switching dynamics.

Pattern 6: Spike Frequency Adaptation

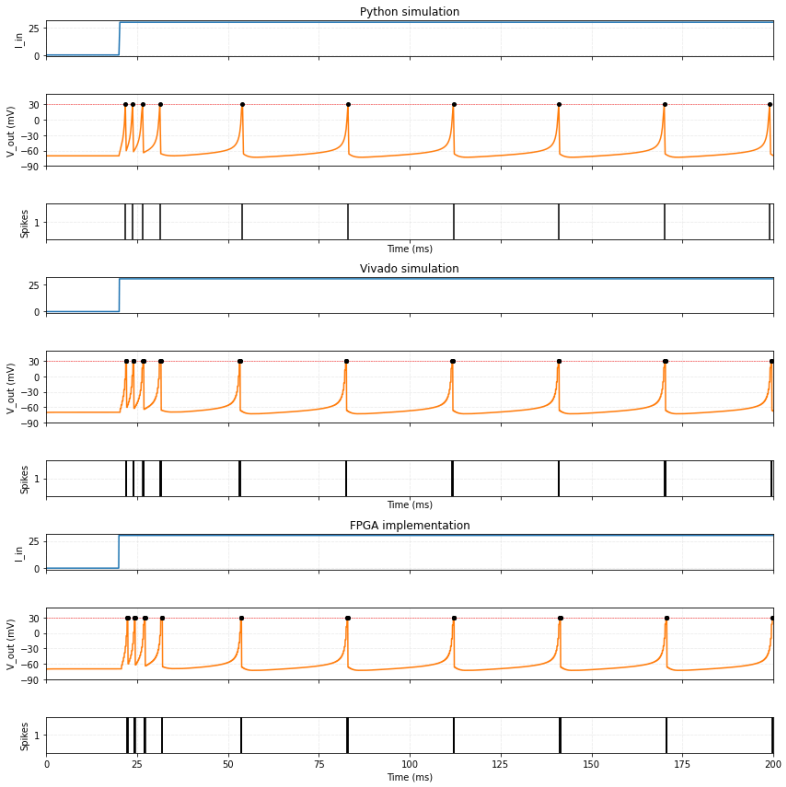


Figure 4.11: Spike frequency adaptation pattern showing progressive ISI lengthening across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.01$, $b = 0.2$, $c = -65$, $d = 8$ | Stimulus: Step DC current

Spike frequency adaptation exhibits progressive interspike interval (ISI) lengthening over sustained stimulation, transitioning from an initial high-frequency burst ($\sim$4 spikes at 15–30 ms) to gradually slowing periodic firing (ISI increasing from $\sim$20 ms to $\sim$30 ms by 200 ms) [15], [16]. Figure 4.11 illustrates the characteristic power-law-like frequency decay, where the firing rate adaptively decreases despite a constant input, encoding temporal history through activity-dependent spike timing modulation. All three platforms accurately reproduce the adaptation dynamics with progressive ISI lengthening from $\sim$20 ms (early phase) to $\sim$30 ms (late phase). The FPGA implementation maintains faithful burst structure (4 spikes at 15-30 ms) and correctly implements the activity-dependent frequency decay throughout the 200 ms simulation. Temporal deviations remain $<$1 ms for the initial burst, with cumulative drift reaching 2-3 ms in late adaptation phase due to compounded ISI approximations.

Pattern 7: Class 1 Excitability

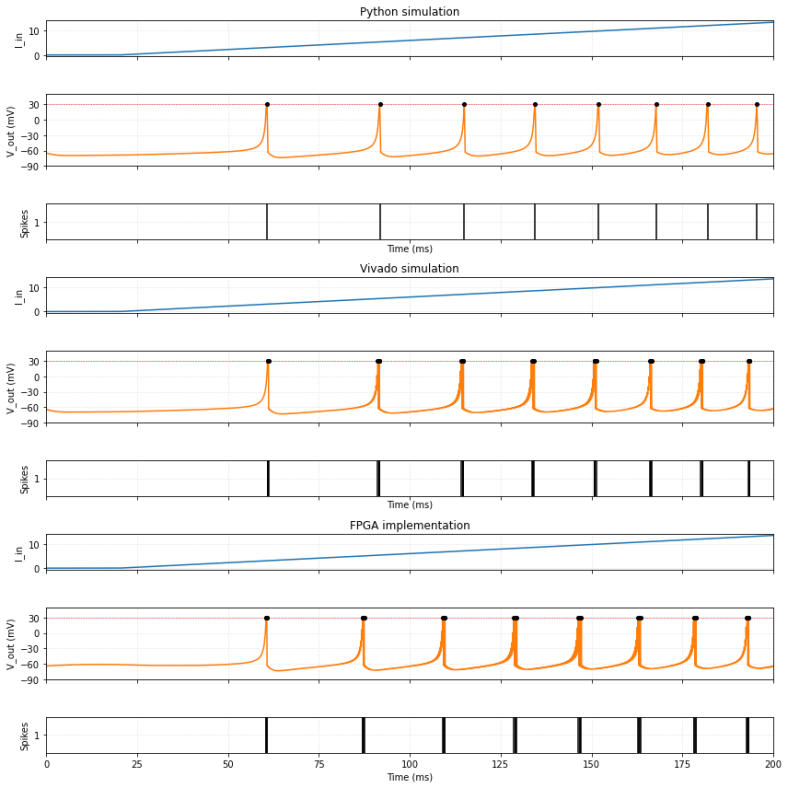


Figure 4.12: Class 1 excitability pattern showing continuous frequency modulation with linearly increasing ramp current across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.05$, $b = 0.22$, $c = -62$, $d = 6$ | FPGA: $a = 0.09$, $b = 0.257$, $c = -62$, $d = 6$ | Stimulus: Ramp current

Class 1 excitability demonstrates continuous frequency modulation from arbitrarily low frequencies (∼8 Hz at threshold onset, ∼60 ms) to progressively higher rates (∼60 Hz by ∼200 ms) as ramp current increases linearly [15], [16]. Figure 4.12 illustrates a firing rate that is proportional to input strength, exhibiting a smooth frequency-current (f-I) relationship characteristic of integrator neurons. Platform-specific parameter adjustments were required: the FPGA parameters ($a = 0.09$, $b = 0.257$) differ from the Python reference ($a = 0.05$, $b = 0.22$) to compensate for quantization effects on bifurcation boundaries. Despite parameter variation, both implementations exhibit the defining Class 1 characteristic: firing onset at low frequency (∼8 Hz) followed by smooth frequency increase to ∼60 Hz. The FPGA maintains accurate spike timing with <1 ms deviation during initial firing (50-80 ms) and 1-2 ms drift in late ramp phase (150-200 ms), demonstrating preserved integrator neuron dynamics.

Pattern 8: Class 2 Excitability

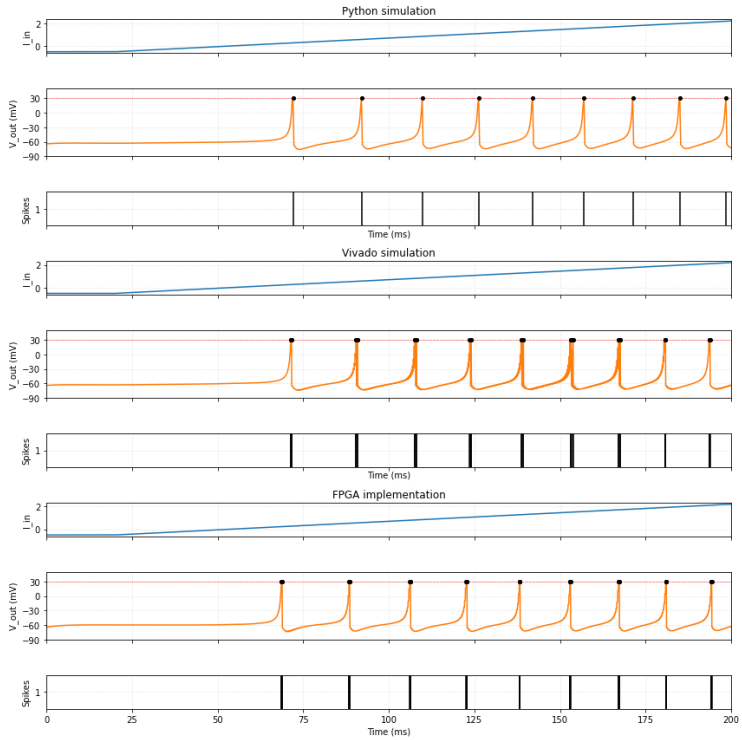


Figure 4.13: Class 2 excitability pattern showing discontinuous threshold with abrupt transition to constant-frequency firing under ramp current across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.03$, $b = 0.267$, $c = -62$, $d = 6$ | FPGA: $a = 0.05$, $b = 0.273$, $c = -62$, $d = 6$ | Stimulus: Ramp current

Class 2 excitability exhibits discontinuous firing threshold with abrupt transition from quiescence to sustained ∼35 Hz spiking at ∼70 ms, maintaining relatively constant frequency (∼15-20 ms ISI) despite continued current increase [15], [16]. Figure 4.13 illustrates the characteristic minimum frequency constraint: neurons cannot produce low-frequency spike trains; instead, they exhibit binary behavior (silent or ∼50 Hz) with a firing rate weakly coupled to input strength. Platform-specific parameter tuning was necessary: the FPGA implementation uses $a = 0.05$, $b = 0.273$, whereas Python's $a = 0.03$, $b = 0.267$ are used to maintain the Class 2 bifurcation structure under fixed-point quantization. Both platforms successfully reproduce the defining characteristic: an abrupt firing onset at approximately 70 ms, followed by the immediate establishment of a ∼50 Hz rhythm. The FPGA maintains remarkably stable ISI (∼15–20 ms) with <1 ms spike timing deviation throughout the 70-200 ms active period, demonstrating preserved frequency constancy.

Pattern 9: Spike Latency

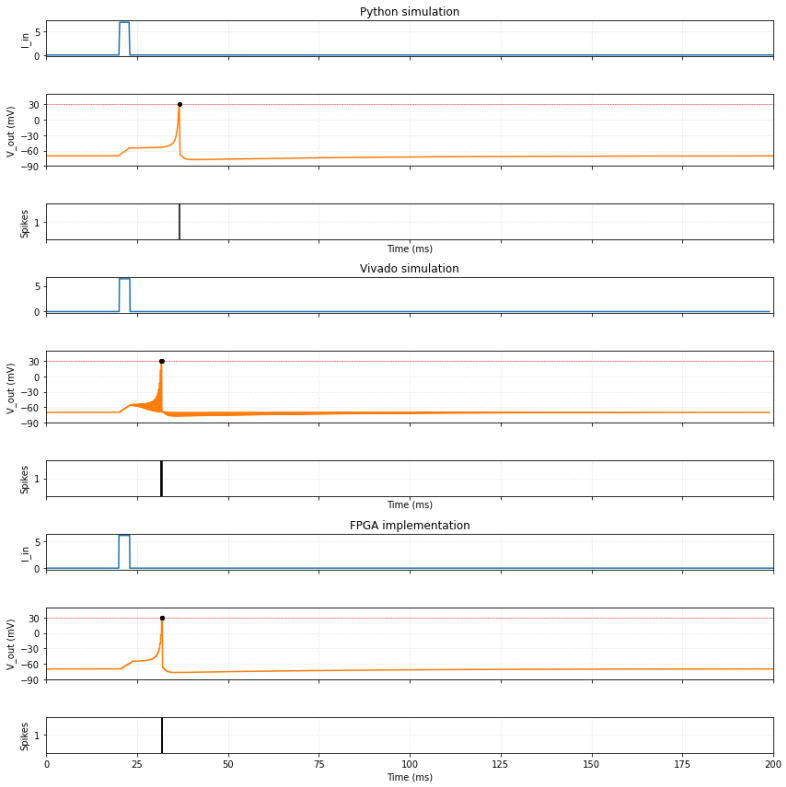


Figure 4.14: Spike latency pattern showing temporal encoding through delayed single-spike response across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.02$, $b = 0.2$, $c = -65$, $d = 6$ | Stimulus: DC impulse current

Spike latency demonstrates an inverse relationship between input strength and response delay, producing a single spike at ∼30 ms following the termination of a brief 10 ms current pulse [15], [16]. Figure 4.14 illustrates the characteristic delayed response, where weak suprathreshold inputs result in long latencies, enabling temporal coding in which spike timing relative to stimulus onset encodes input magnitude. All three platforms accurately reproduce the latency-current relationship with single spike generation at ∼30 ms, approximately 20 ms after the 10 ms impulse termination. The FPGA implementation demonstrates <0.5 ms spike timing deviation from the Python reference, validating precise temporal encoding capability. The brief impulse creates subthreshold depolarization visible in the membrane potential trajectory (elevated baseline after pulse), followed by delayed threshold crossing as recovery dynamics slowly evolve. The absence of subsequent spikes (single-spike response) validates correct reset dynamics and demonstrates faithful implementation of phasic response characteristics under transient stimulation.

Pattern 10: Subthreshold Oscillation

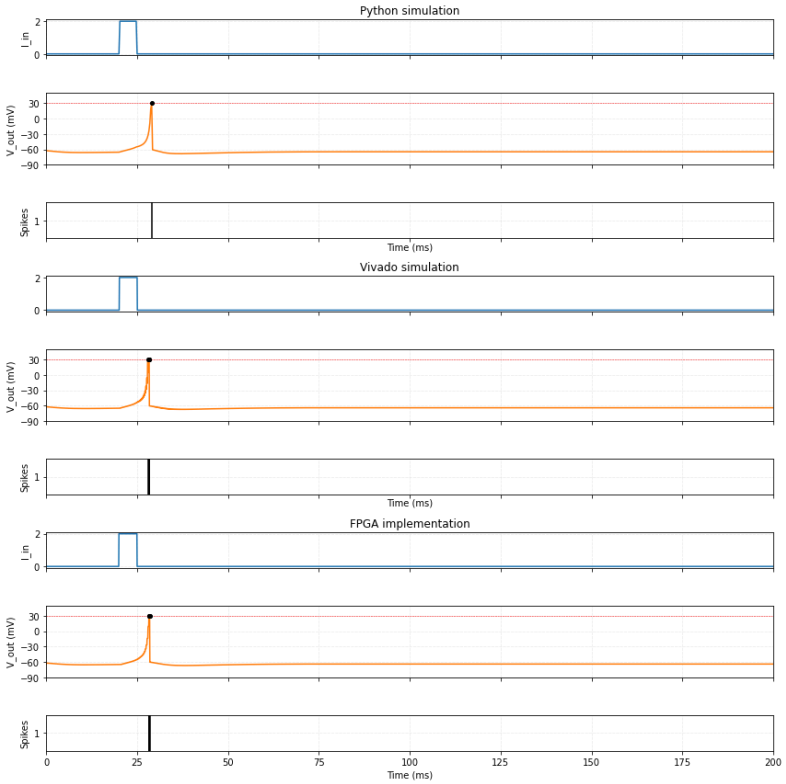


Figure 4.15: Subthreshold oscillation pattern showing damped membrane potential fluctuations following impulse-evoked spike across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.05$, $b = 0.25$, $c = -60$, $d = 0$ | Stimulus: DC impulse current

Subthreshold oscillations exhibit damped membrane potential fluctuations following a single spike at ∼28 ms, with visible oscillatory decay (∼10–15 mV amplitude) continuing for ∼50–100 ms post-spike before stabilizing to resting potential [15], [16]. Figure 4.15 demonstrates intrinsic resonance properties where membrane dynamics naturally oscillate at preferred frequencies, enabling neurons to function as bandpass filters that selectively respond to temporally structured inputs. All three platforms accurately reproduce damped subthreshold oscillations following the impulse-evoked spike at ∼28 ms. The FPGA implementation preserves the oscillatory waveform morphology, with a peak-to-peak amplitude of ∼10–15 mV visible for 50–100 ms, demonstrating a faithful representation of complex subthreshold dynamics despite fixed-point quantization. Spike timing shows <0.5 ms deviation, while oscillation phase and frequency match closely across platforms, validating preservation of resonant properties. Minor amplitude quantization (±1–2 mV) appears in FPGA oscillation peaks due to Q5.11 resolution limits.

Pattern 11: Resonator

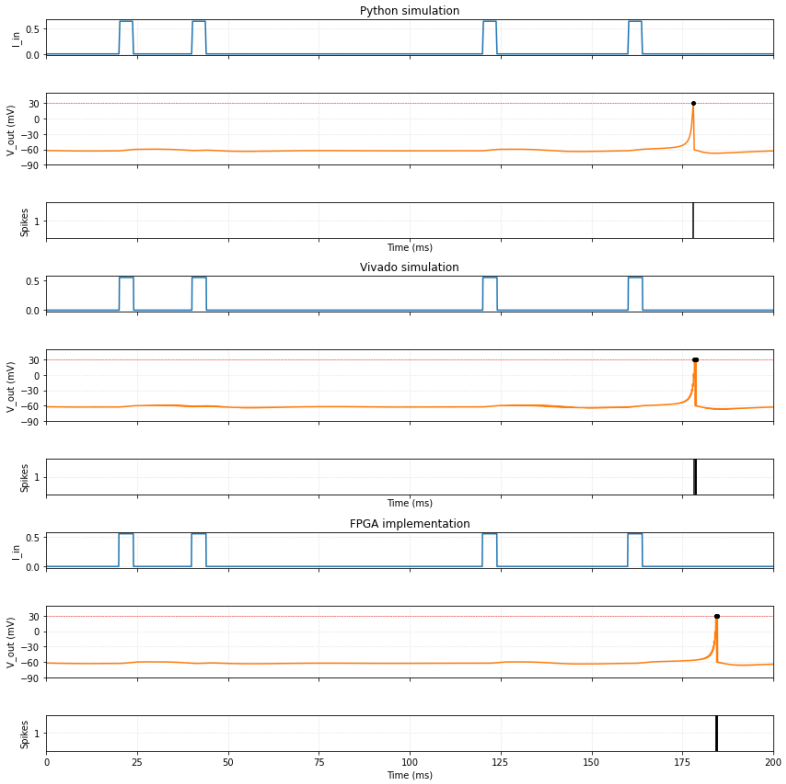


Figure 4.16: Resonator pattern demonstrating frequency-selective spiking across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.1$, $b = 0.26$, $c = -60$, $d = -1$ | FPGA: $a = 0.01$, $b = 0.2571$, $c = -60$, $d = -1$ | Stimulus: Four DC impulse doublets

Resonator dynamics show frequency-selective spiking, where the neuron responds only to input doublets at ∼170–180 ms, whose interspike interval matches the intrinsic subthreshold oscillation frequency, while ignoring non-resonant doublets at ∼15–25 ms and 115–125 ms [15], [16]. Figure 4.16 shows selective amplification of temporally matched inputs through constructive interference between stimulus timing and membrane resonance, enabling biological bandpass filtering. Significant parameter adaptation was required for FPGA implementation: $a = 0.01$ versus Python's $a = 0.1$ (10× difference) while maintaining $b = 0.2571$ close to Python's $b = 0.26$ and critical $d = -1$ constant. Despite parameter differences, both platforms successfully reproduce the defining resonator property: selective response to the ∼170-180 ms doublet while remaining quiescent for earlier non-resonant doublets. The FPGA generates a single spike at ∼185 ms with <2–3 ms timing deviation from the Python reference, validating preservation of frequency-selective dynamics.

Pattern 12: Integrator

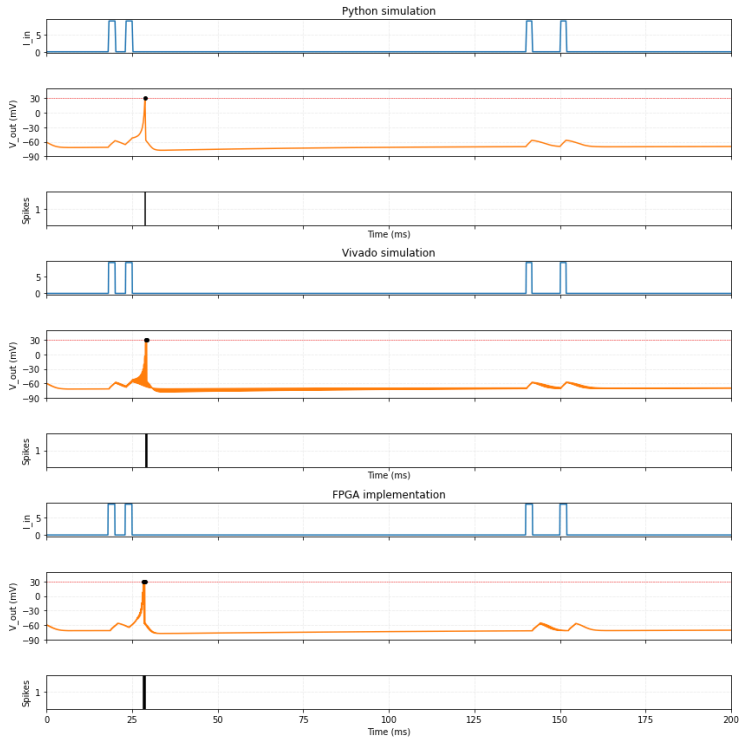


Figure 4.17: Integrator pattern demonstrating temporal summation and coincidence detection across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.02$, $b = 0.21$, $c = -55$, $d = 6$ | FPGA: $a = 0.01$, $b = 0.21$, $c = -55$, $d = 6$ | Stimulus: Two impulse doublets

Integrator dynamics show temporal summation, where the neuron fires a single spike at ∼30 ms only in response to a high-frequency doublet (with an interval of ∼15–20 ms), while remaining subthreshold for identical but temporally separated inputs at ∼130–150 ms [15], [16]. Figure 4.17 shows the absence of subthreshold oscillations (smooth membrane trajectory), enabling pure integration where closely-timed inputs summate to reach threshold, implementing coincidence detection through temporal integration. Minor parameter adjustment between platforms: Python uses $a = 0.02$ while FPGA uses $a = 0.01$ (halved) to maintain integration dynamics. Both platforms successfully demonstrate integrator behavior: a single spike at ∼30 ms for the high-frequency doublet with visible membrane summation. In comparison, the low-frequency doublet at ∼130–150 ms produces only subthreshold depolarization (∼5–10 mV transients) that decays before reaching threshold. The FPGA shows <0.5 ms spike timing deviation and accurately reproduces the smooth, non-oscillatory membrane trajectory characteristic of integrators.

Pattern 13: Rebound Spike

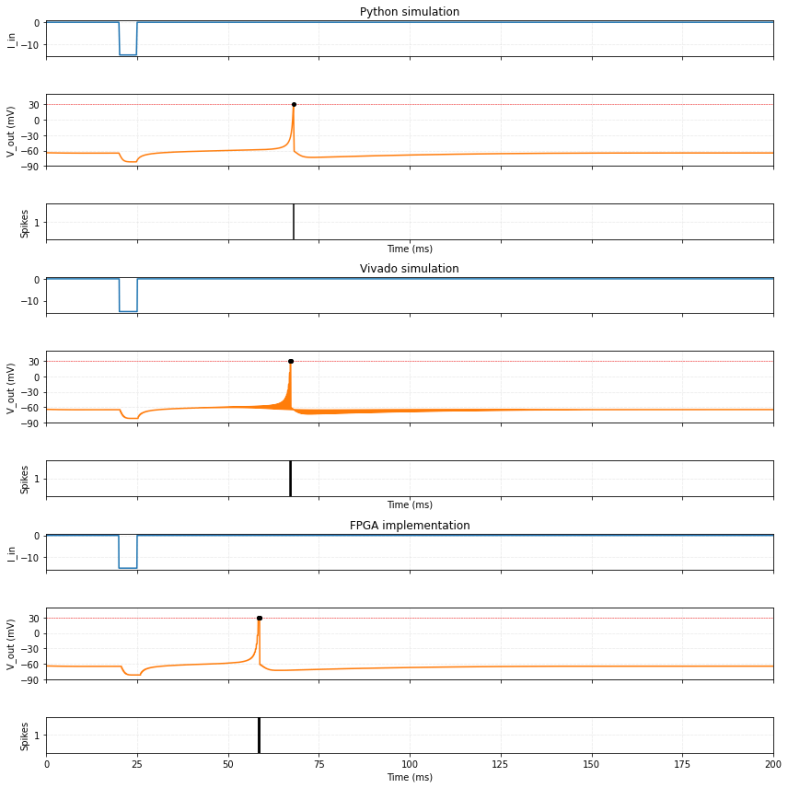


Figure 4.18: Rebound spike pattern demonstrating post-inhibitory excitation across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.03$, $b = 0.25$, $c = -60$, $d = 4$ | FPGA: $a = 0.03$, $b = 0.247$, $c = -60$, $d = 4$ | Stimulus: Negative impulse current

The rebound spike demonstrates post-inhibitory excitation, where a hyperpolarizing input at 20–25 ms paradoxically triggers a spike at approximately 60 ms upon inhibition release, accompanied by visible membrane hyperpolarization (-65 mV) followed by rebound depolarization that exceeds the threshold [15], [16]. Figure 4.18 illustrates counterintuitive dynamics where brief inhibition can produce excitation through anodal break excitation mechanisms. Minimal parameter adjustment required between platforms: only b differs slightly (Python: $b = 0.25$, FPGA: $b = 0.247$) while a, c, d remain identical, indicating robust rebound dynamics. All three platforms accurately reproduce the complete rebound sequence, including initial hyperpolarization to ~ -60 mV during negative current (20–25 ms), a smooth rebound trajectory post-inhibition, and spike generation at 65 ms, with a timing deviation of $\sim$1–5 ms across platforms. The FPGA implementation faithfully preserves the critical nonlinear dynamics required for anodal break excitation.

Pattern 14: Rebound Bursting

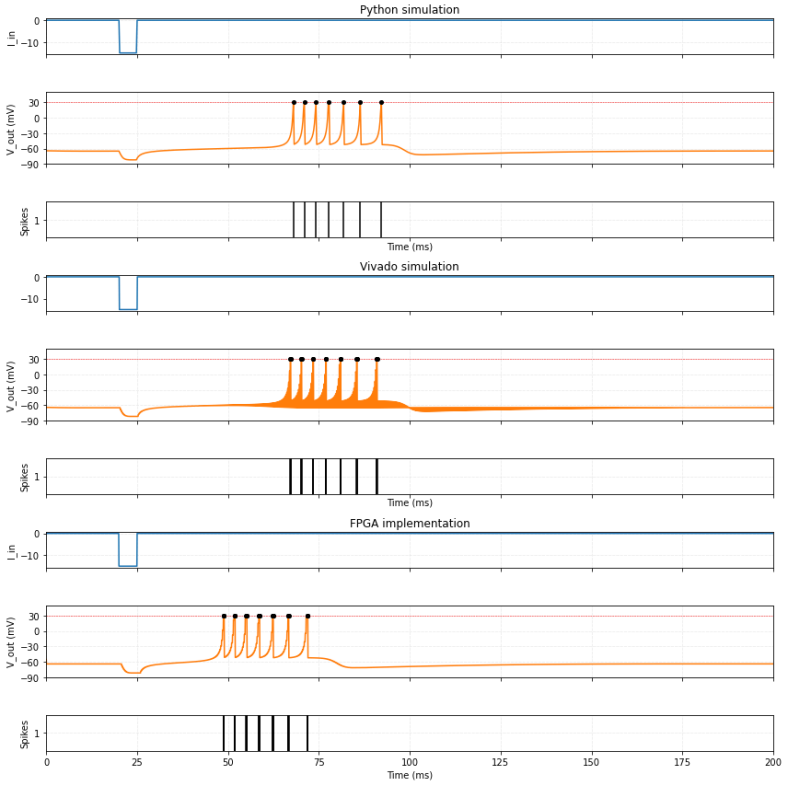


Figure 4.19: Rebound bursting pattern demonstrating post-inhibitory burst generation across Python, Vivado, and FPGA platforms.

Parameters: Python: $a = 0.03$, $b = 0.25$, $c = -52$, $d = 0$ | FPGA: $a = 0.03$, $b = 0.247$, $c = -52$, $d = 0.2$ | Stimulus: Negative impulse current

Rebound bursting demonstrates post-inhibitory burst generation where hyperpolarizing input (at 20–25 ms) triggers high-frequency spike bursts (∼7–8 spikes at 50–95 ms, ∼25 Hz intraburst frequency) upon inhibition release [15], [16]. Figure 4.19 shows amplified rebound response compared to single rebound spike (Pattern 13), combining anodal break excitation with intrinsic burst dynamics through elevated reset potential ($c = -52$ mV) and minimal recovery ($d \approx 0$). Platform parameters show minimal variation: $a = 0.03$ and $b \approx 0.25$ identical, $c = -52$ mV constant (elevated reset enabling bursts), while d differs slightly (Python: $d = 0$, FPGA: $d = 0.2$). Despite a small recovery increment difference, all platforms reproduce the complete rebound burst: hyperpolarization to ∼−65 mV during inhibition (20–25 ms), followed by a 7–8 spike burst at 50–95 ms. The FPGA maintains a burst structure with ∼2–5 ms deviation for individual spike timings and accurate interspike intervals throughout the burst.

Pattern 15: Threshold Variability

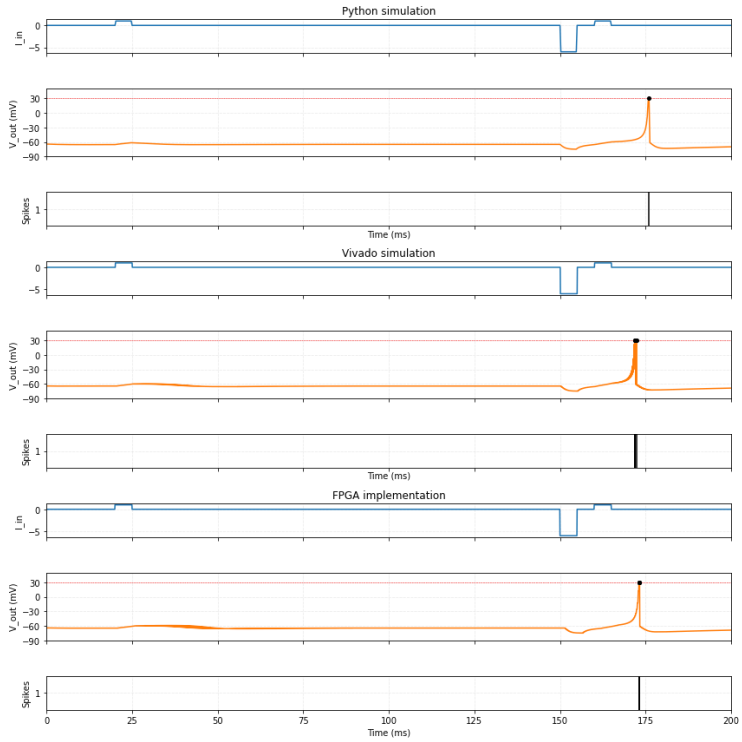


Figure 4.20: Threshold variability pattern demonstrating activity-dependent excitability across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.03$, $b = 0.248$, $c = -60$, $d = 4$ | Stimulus: Positive and negative impulses

Threshold variability demonstrates activity-dependent excitability where identical 10 mV depolarizing inputs (∼20–25 ms and ∼16–165 ms) produce opposite outcomes: the first pulse remains subthreshold (no spike), while the second identical pulse triggers spike at ∼175 ms because preceding inhibition (at 150–155 ms) lowered the effective threshold [15], [16]. Figure 4.20 illustrates that biological neurons exhibit dynamic thresholds modulated by the recovery variable u: inhibition decreases u (lowering the threshold), while excitation increases u (raising the threshold). Identical parameters across platforms ($a = 0.03$, $b = 0.248$, $c = -60$, $d = 4$) demonstrate robust threshold variability dynamics. All three platforms accurately reproduce the defining characteristic: first positive pulse (20–25 ms) produces ∼10 mV depolarization without spiking, while the second identical pulse (160–165 ms) triggers a spike at ∼175 ms with <0.5 ms timing deviation. The FPGA faithfully captures the subtle u dynamics governing dynamic threshold modulation.

Pattern 16: Bistability

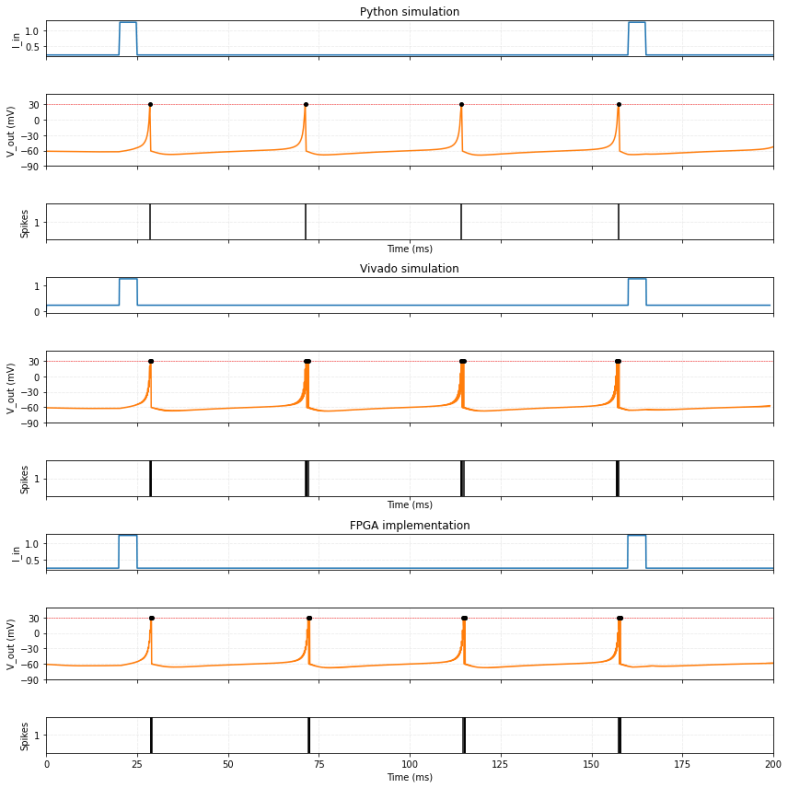


Figure 4.21: Bistability pattern demonstrating stable state switching between resting and tonic spiking modes across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.1$, $b = 0.257$, $c = -60$, $d = 0$ | Stimulus: Two positive impulses

Bistability demonstrates stable state switching where brief excitatory pulses toggle between two persistent modes: first impulse (20–25 ms) transitions the neuron from resting to sustained tonic spiking (∼25 Hz, 30–155 ms), while the second impulse (160–165 ms) terminates spiking and restores resting state [15], [16]. Figure 4.21 shows phase-dependent switching, where termination requires an appropriately timed input during the oscillation cycle, thereby creating binary memory states encoded in ongoing activity patterns without the need for sustained input. Identical parameters across platforms ($a = 0.1$, $b = 0.257$, $c = -60$, $d = 0$) with an elevated recovery rate and zero recovery increment, creating a bistable regime. All three platforms accurately reproduce the complete bistable sequence: initial resting state (0–20 ms), transition to tonic spiking after first impulse (sustained firing 30–155 ms at 25 Hz, 4 spikes), and return to resting after second impulse (160+ ms). The FPGA maintains precise spike timing with <1 ms deviation throughout the sustained firing period.

Pattern 17: Depolarizing after-potential (DAP)

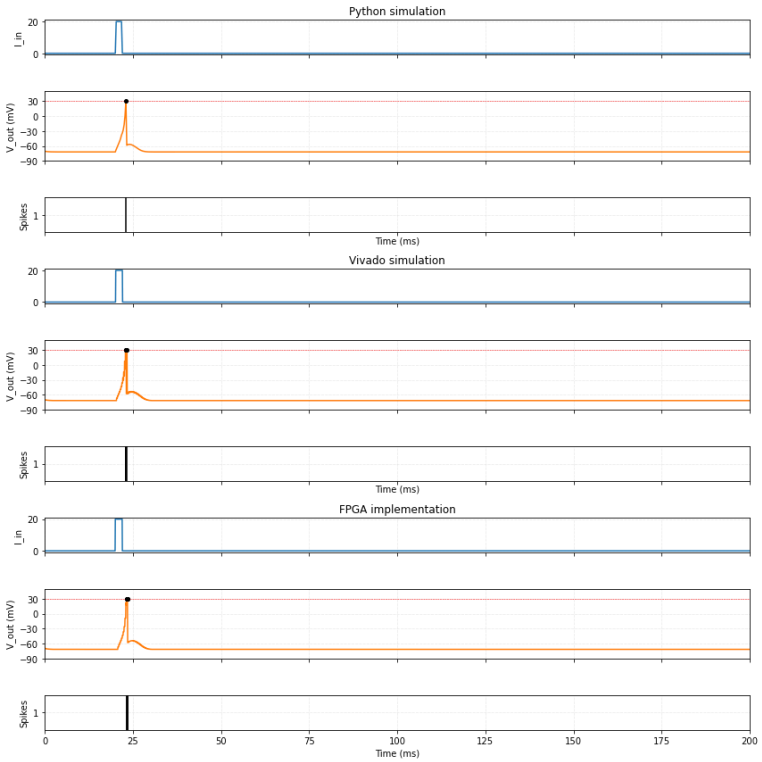


Figure 4.22: Depolarizing after-potential (DAP) pattern showing post-spike superexcitability across Python, Vivado, and FPGA platforms.

Parameters: $a = 1$, $b = 0.18$, $c = -60$, $d = -21$ | Stimulus: Positive impulse current

Depolarizing after-potential (DAP) demonstrates post-spike superexcitability where membrane potential exhibits prolonged depolarization ($\sim$−45 mV to −50 mV plateau for $\sim$10 ms) following spike at 25 ms, rather than typical after-hyperpolarization (AHP) [15], [16]. Figure 4.22 shows an elevated baseline creating a shortened refractory period and an increased excitability window. The extreme negative recovery increment ($d = -21$) produces anti-recovery dynamics, where u decreases after the spike, sustaining depolarization. Identical extreme parameters across platforms: $a = 1$ (speedy recovery rate) and $d = -21$ (massive negative recovery increment, strongest anti-recovery seen). All three platforms accurately reproduce the DAP waveform, which consists of a spike at 25 ms followed by a sustained elevated membrane potential ($\sim$−45 mV to −50 mV) that lasts for more than 10 ms before gradually returning to resting levels. The FPGA exhibits a <1 ms spike timing deviation and faithfully preserves the prolonged depolarized plateau, demonstrating the accurate implementation of the counterintuitive recovery dynamics.

Pattern 18: Accommodation

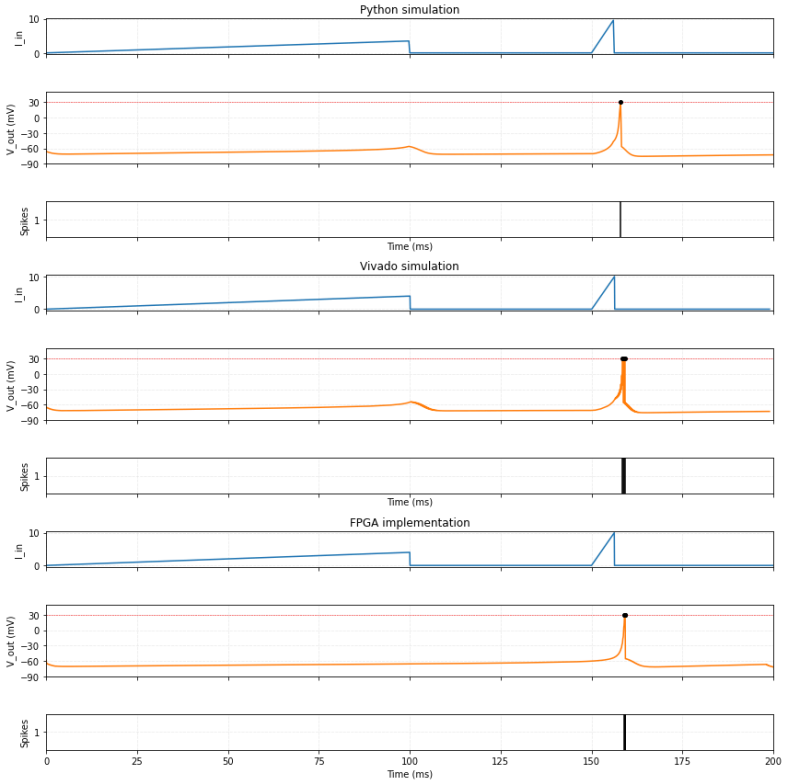


Figure 4.23: Accommodation pattern demonstrating rate-dependent excitability across Python, Vivado, and FPGA platforms.

Parameters: $a = 0.02$, $b = 0.204$, $c = -55$, $d = 4$ | Stimulus: Two ramp currents

Accommodation demonstrates rate-dependent excitability, where a slow current increase (over 100 ms) fails to trigger spikes despite its substantial magnitude, while a smaller but sharper ramp (at 150–160 ms) elicits a spike at 160 ms [15], [16]. Figure 4.23 illustrates counterintuitive dynamics: gradual stimulation enables neuronal adaptation, thereby preventing firing, whereas a rapid onset exceeds the accommodation rate. Visible membrane depolarization during slow ramp ($\sim$−60 mV to −50 mV at 100–110 ms) remains subthreshold as recovery variable u tracks input, elevating threshold. Identical parameters across platforms ($a = 0.02$, $b = 0.204$, $c = -55$, $d = 4$) demonstrate robust accommodation dynamics. All three platforms accurately reproduce the critical accommodation characteristic: slow ramp (0–100 ms) produces visible membrane depolarization ($\sim$10–15 mV elevation) without spiking, and fast ramp (150–160 ms) triggers a spike at 160 ms. The FPGA exhibits a <1 ms spike timing deviation and accurately captures the subthreshold membrane trajectory during the slow accommodation phase.

Pattern 19: Inhibition-Induced Spiking

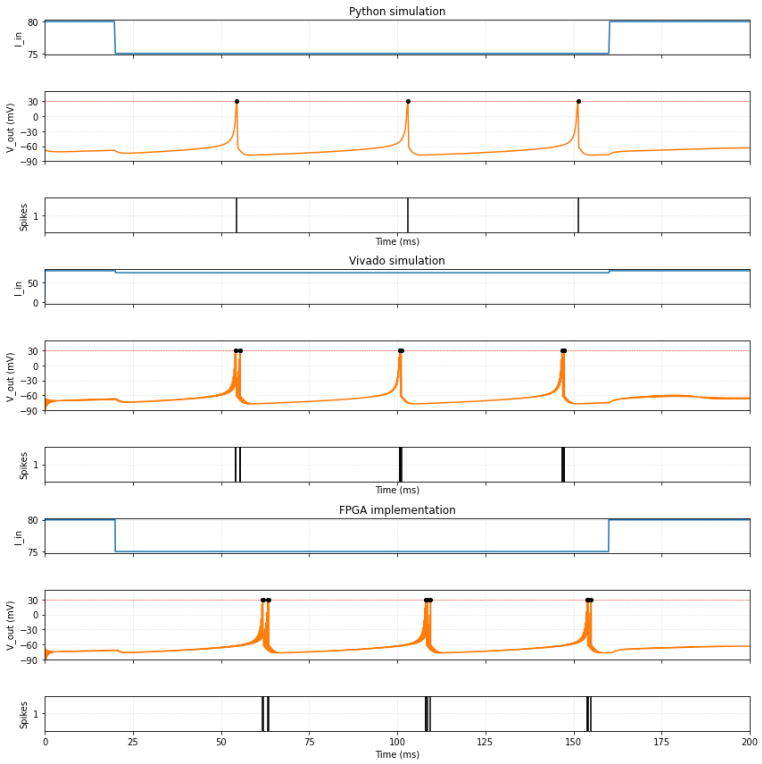


Figure 4.24: Inhibition-induced spiking pattern demonstrating paradoxical excitation by hyperpolarization across Python, Vivado, and FPGA platforms.

Parameters: $a = -0.03$, $b = -1$, $c = -60$, $d = 8$ | Stimulus: Step hyperpolarizing current

Inhibition-induced spiking demonstrates paradoxical excitation by inhibition, where sustained hyperpolarizing current (from 20 ms) triggers periodic spiking ($\sim$60 ms, $\sim$110 ms, $\sim$155 ms intervals) while the absence of input produces quiescence [15], [16]. Figure 4.24 shows counterintuitive dynamics where inhibition activates firing through hyperpolarization-activated mechanisms. The negative recovery parameters ($a = -0.03$, $b = -1$) reverse typical adaptation, implementing inverted excitability. Unique negative recovery parameters distinguish this pattern: $a = -0.03$ (negative recovery rate, anti-recovery) and $b = -1$ (negative coupling, inverted relationship between v and u). All three platforms accurately reproduce inhibition-induced spiking: quiescence during the initial period (0–20 ms), followed by periodic firing at $\sim$60 ms, $\sim$110 ms, $\sim$155 ms ($\sim$50 ms ISI) during sustained hyperpolarization. The FPGA maintains <2–5 ms spike timing deviation across three spikes and faithfully preserves the $\sim$50 ms interspike intervals, demonstrating accurate implementation through negative parameter interactions.

Pattern 20: Inhibition-Induced Bursting

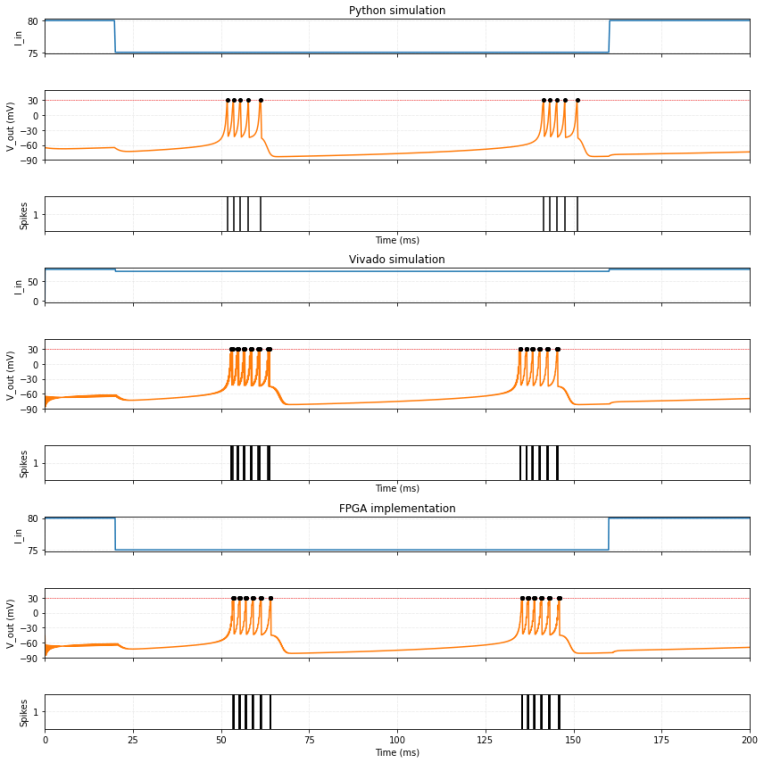


Figure 4.25: Inhibition-induced bursting pattern demonstrating paradoxical burst generation by hyperpolarization across Python, Vivado, and FPGA platforms.

Parameters: $a = -0.03$, $b = -1$, $c = -45$, $d = 0.1$ | Stimulus: Step hyperpolarizing

Inhibition-induced bursting demonstrates paradoxical burst generation by hyperpolarization, where sustained inhibitory current (from 20 ms) triggers rhythmic high-frequency bursts (∼5–6 spikes per burst at 50–70 ms and 140–155 ms, ∼25 Hz intraburst frequency, ∼80 ms interburst interval) while quiescence persists without input [15], [16]. Figure 4.25 shows an amplified version of Pattern 19, where an elevated reset potential ($c = -45$ mV vs. -60 mV) converts inhibition-induced spiking to bursting. Parameters closely match Pattern 19 with a critical difference in reset potential: $c = -45$ mV (elevated by 15 mV) converts spiking to bursting, while $d = 0.1$ (near-zero) enables rapid intraburst firing. Negative recovery parameters ($a = -0.03$, $b = -1$), identical to Pattern 19, maintain inverted dynamics. All three platforms accurately reproduce inhibition-induced bursting, characterized by initial quiescence (0–20 ms), followed by two complete bursts, each containing 5–6 spikes with an intraburst frequency of ∼25 Hz and an interburst interval of ∼90 ms. The FPGA maintains <1 ms spike timing deviation within bursts and accurately preserves burst structure.

4.2 Time-multiplexed Multiple Virtual Neuron Implementation

Having validated single-neuron dynamics across all 20 characteristic spiking patterns, we now demonstrate scalability through the implementation of time-multiplexed virtual neurons. This resource-efficient technique enables a single physical hardware neuron to emulate multiple independent virtual neurons with distinct behavioral configurations. This approach addresses the fundamental challenge of resource-limited FPGA platforms: achieving large-scale neural network emulation without proportional hardware replication.

4.2.1 Time-Multiplexing Architecture

The time-multiplexing mechanism operates by sequentially cycling through different parameter sets within each simulation timestep, where a single neuron core updates multiple virtual neuron states in rapid succession. We utilize the 'config' interface in our Jupyter notebook control framework to dynamically program all 20 validated spiking patterns, allowing for runtime selection of behavioral diversity across virtual neuron populations. Each virtual neuron maintains independent state variables (v_i and u_i), stored in BRAM and accessed sequentially during time-multiplexed updates at a 100 MHz system clock frequency.

4.2.2 Homogeneous Population: Tonic Bursting Validation

Figure 4.26 presents a raster plot of 50 virtual neurons configured identically with tonic bursting parameters (Pattern 3: $a = 0.02$, $b = 0.2$, $c = -50$, $d = 2$), demonstrating synchronized population dynamics under a homogeneous configuration. All neurons exhibit phase-locked bursting at $\sim$40 ms intervals (4 bursts over 200 ms simulation), with each burst containing 8–10 spikes at $\sim$100 Hz intraburst frequency. The perfect synchronization validates: (1) parameter consistency across virtual neuron instances, (2) state variable isolation, preventing cross-neuron interference, and (3) deterministic update sequencing, maintaining identical input conditions.

Figure 4.27 illustrates the detailed dynamics of a representative virtual neuron (Neuron 1) from the population, showing input step current, membrane potential trajectory with characteristic burst envelope, and binary spike output. The voltage trace confirms the faithful reproduction of Pattern 3 dynamics, as validated in Subsection 4.1.3: burst onset at $\sim$5 ms, interburst hyperpolarization to ~ -65 mV, and sustained rhythmic bursting throughout the stimulation period. This single-neuron view provides ground-truth verification that time-multiplexed virtual neurons preserve individual dynamical fidelity established during single-neuron validation.

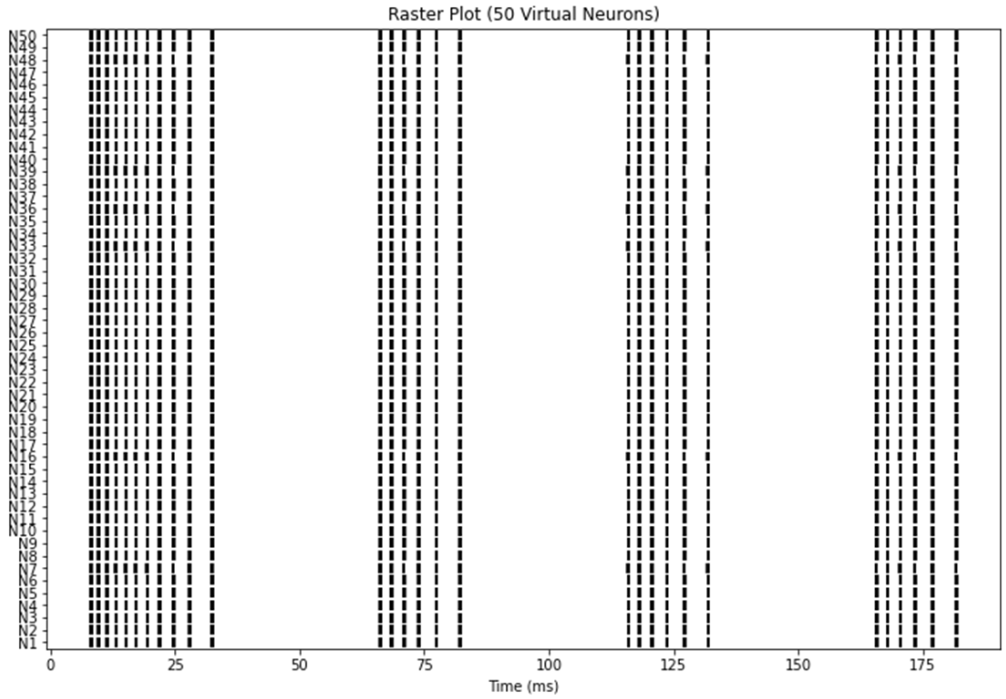


Figure 4.26: Raster plot of 50 time-multiplexed virtual neurons configured identically with tonic bursting parameters (Pattern 3: $a = 0.02$, $b = 0.2$, $c = -50$, $d = 2$).

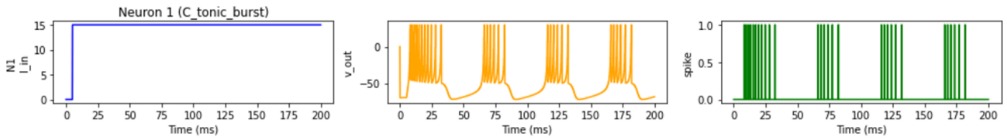


Figure 4.27: Detailed dynamics of representative virtual neuron (Neuron 1) from 50-neuron tonic bursting population. Three panels show: (left) input step current, (center) membrane potential trajectory and (right) binary spike.

4.2.3 Heterogeneous Population: Random Pattern Assignment

To demonstrate the behavioral diversity achievable through time-multiplexing, we implemented heterogeneous populations where each virtual neuron receives a randomly assigned spiking pattern configuration. Figure 4.28 shows a raster plot of 50 virtual neurons with random selection from the first six validated patterns (tonic

spiking, phasic spiking, tonic bursting, phasic bursting, mixed mode, spike frequency adaptation). The resulting spatiotemporal activity exhibits rich dynamics: sparse phasic responses (single spikes or bursts from phasic neurons), sustained tonic activity (regular firing from tonic neurons), and adaptive modulation (frequency-decreasing trains from adaptation neurons). This heterogeneity replicates biological network diversity where cortical populations comprise multiple cell types with distinct intrinsic properties [15], [16].

Figure 4.29 provides detailed examination of the first six virtual neurons from this heterogeneous population, revealing pattern assignments: Neurons 1–2 display tonic spiking (∼40 Hz regular firing), Neuron 3 exhibits phasic bursting (single burst at ∼25–50 ms), Neuron 4 shows tonic bursting (rhythmic burst trains), Neuron 5 demonstrates phasic bursting, and Neuron 6 exhibits mixed mode (initial burst transitioning to tonic spiking). Each neuron's voltage trajectory and spike train precisely match the corresponding pattern characterization from Subsection 4.1.3, confirming that time-multiplexing preserves pattern-specific dynamics without degradation. The input current profiles vary across neurons, demonstrating independent stimulus control for each virtual instance.

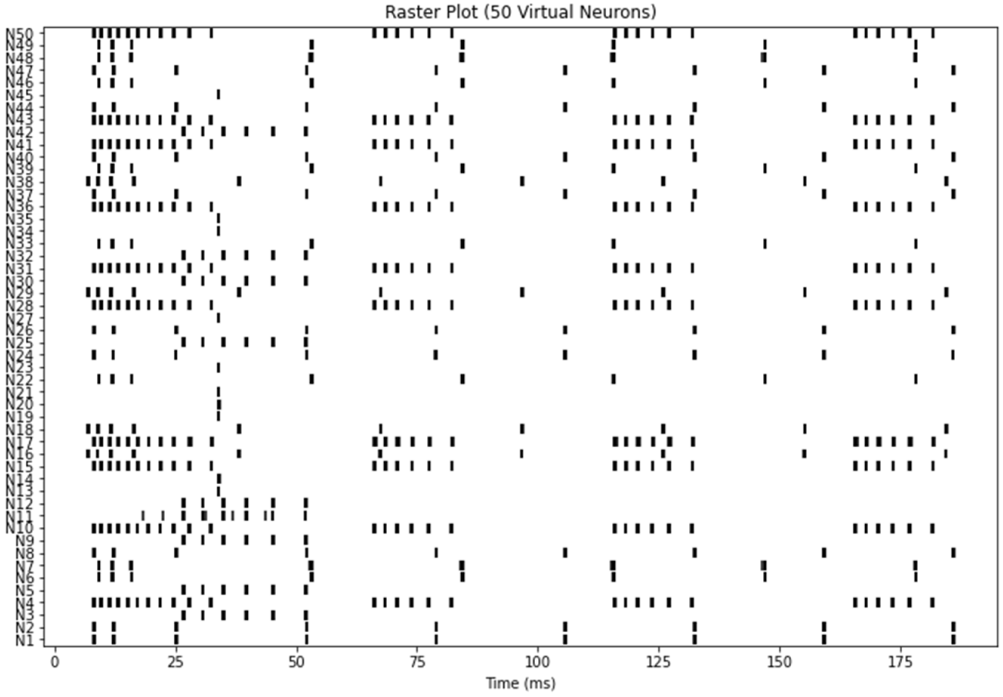


Figure 4.28: Raster plot of 50 time-multiplexed virtual neurons with heterogeneous configuration through random pattern assignment from first six validated patterns (tonic spiking, phasic spiking, tonic bursting, phasic bursting, mixed mode, spike frequency adaptation).

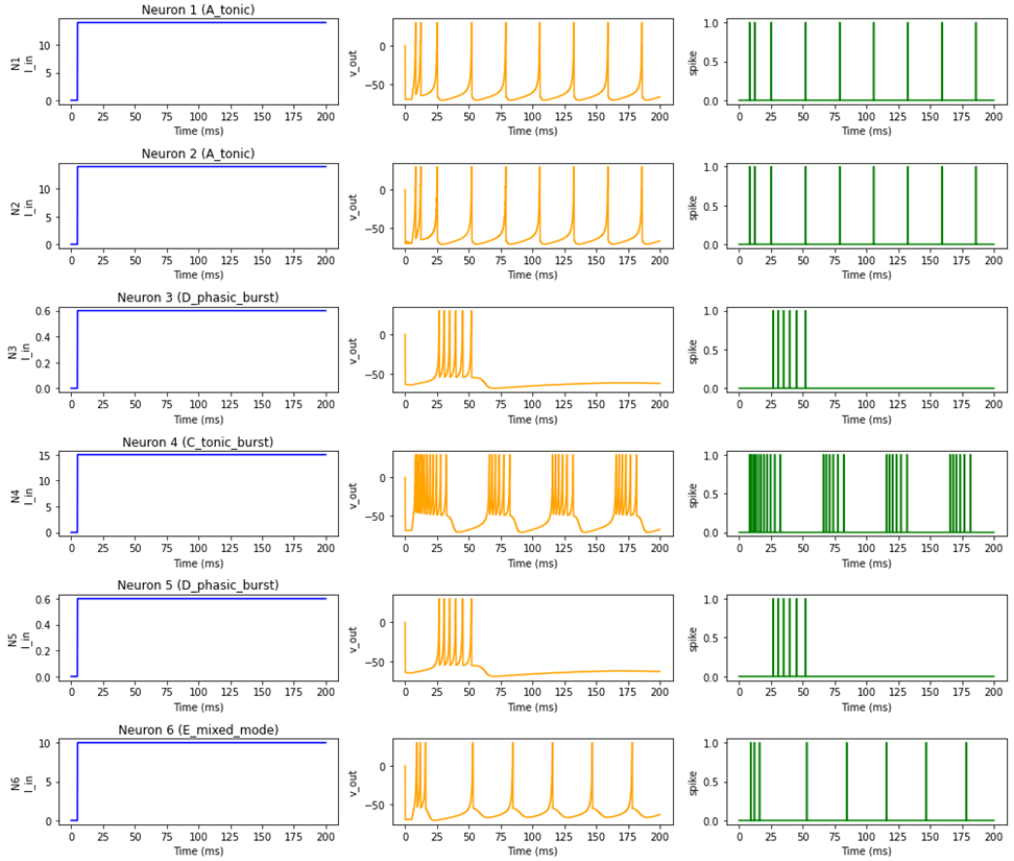


Figure 4.29: Detailed examination of first six virtual neurons from heterogeneous 50-neuron population, revealing pattern-specific dynamics. Each row shows input current (left), membrane voltage trajectory (center), and spike output (right) for individual neurons.

4.2.4 Large-Scale Population: 1000 Virtual Neurons

Figure 4.30 demonstrates scalability to 1000 virtual neurons with random pattern assignment across the first six spiking patterns. The raster plot reveals emergent population-level structure despite individual neuron diversity: tonic neurons (Patterns 1, 3, 5) create dense horizontal bands of sustained activity, phasic neurons (Patterns 2, 4) contribute sparse vertical event markers, and adaptation neurons (Pattern 6) produce temporally evolving firing rates. This spatiotemporal organization emerges naturally from the statistical distribution of pattern assignments without explicit network connectivity, illustrating how intrinsic neuronal diversity alone generates complex activity patterns.

The heterogeneous population approach demonstrated here provides a foundation for biologically realistic network modeling where cortical circuits comprise

∼80% excitatory neurons (regular spiking, intrinsically bursting) and ∼20% inhibitory interneurons (fast spiking, low-threshold spiking), each exhibiting distinct firing patterns [14]. By statistically assigning validated Izhikevich patterns to match biological proportions, we can emulate physiologically accurate population dynamics for computational neuroscience investigations on resource-constrained FPGA platforms.

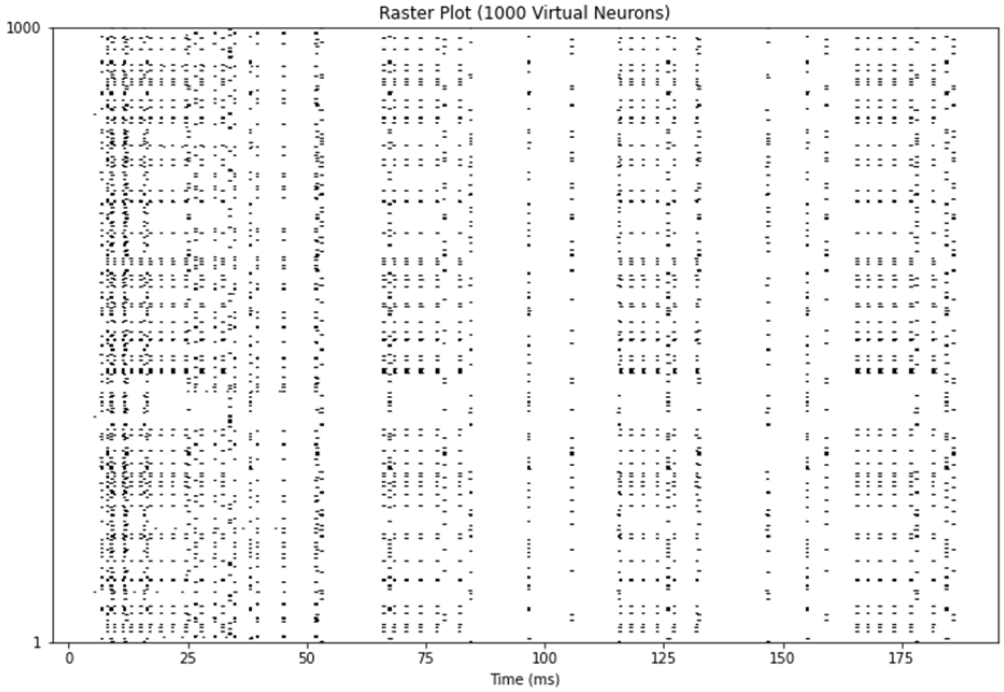


Figure 4.30: Large-scale raster plot of 1000 time-multiplexed virtual neurons with random pattern assignment across first six spiking patterns (tonic spiking, phasic spiking, tonic bursting, phasic bursting, mixed mode, spike frequency adaptation).

4.3 Mulit-Neuron Implementation for Parallel Operation

While Section 4.2 demonstrated resource-efficient scalability through time-multiplexing, we now explore the complementary approach of parallel multi-neuron implementation where multiple independent neuron cores execute simultaneously, trading hardware resources for computational throughput. This architecture instantiates the validated single-neuron IP core multiple times within Vivado block design, creating physically distinct hardware instances that operate concurrently without sequential scheduling overhead.

4.3.1 Parallel Architecture and Design Methodology

The parallel implementation methodology instantiates N independent neuron cores in FPGA fabric, each comprising dedicated arithmetic units, state registers (v, u), and control logic. Unlike time-multiplexing, where a single core sequentially processes multiple virtual states, parallel cores execute true simultaneous computation. All N neurons update their states within the same clock cycle, achieving $N\times$ computational throughput compared to sequential time-multiplexing.

We leverage Vivado IP Integrator block design to replicate the validated single-neuron IP core, connecting each instance to independent AXI4-Lite control interfaces for parameter configuration via Jupyter notebook. Each core receives individual input currents, maintains isolated state variables in dedicated registers (avoiding BRAM contention), and generates independent spike outputs. The Jupyter control framework addresses each neuron core through unique AXI base addresses, enabling heterogeneous configuration where different cores implement distinct spiking patterns from the validated repertoire of 20 biological dynamics.

The fundamental trade-off between time-multiplexing and parallel implementation is resource consumption versus latency: parallel implementation consumes N times the hardware resources (LUTs, FFs, DSPs, BRAMs) but achieves real-time biological simulation without accumulating update latency. For Zynq-7020 with 13.95% LUT utilization per neuron (Subsection 4.1.1), the theoretical maximum parallel capacity is $\sim$7 neurons before exhausting LUT resources ($100\%/13.95\% \approx 7.17$). However, due to the optimized resource-sharing capability, more than 7 parallel neurons can be implemented, as shown in Section 4.4.

4.3.2 Systematic Scalability Validation: 2-10 Parallel Neurons

We systematically validate the scalability of parallel implementation by incrementally increasing the neuron count from 2 to 10 cores, demonstrating the preservation of behavioral fidelity and resource scaling characteristics. Each configuration assigns heterogeneous spiking patterns to individual cores, creating diverse population dynamics analogous to biological neural circuits comprising multiple cell types.

2-Neuron Configuration (Fig. 4.31): The minimal parallel network pairs tonic spiking (N1, $\sim$40 Hz regular firing) with phasic spiking (N2, single spike at $\sim$35 ms). Individual neuron dynamics reveal that N1 exhibits sustained periodic firing with consistent interspike intervals of $\sim$25 ms, while N2 demonstrates edge detection with a transient response followed by silence. The raster plot confirms temporal independence: N1 spikes occur at regular intervals regardless of N2 activity, validating computational isolation between parallel cores. This baseline configuration establishes that dual-core instantiation preserves single-neuron fidelity, validated in Subsection 4.1.3.

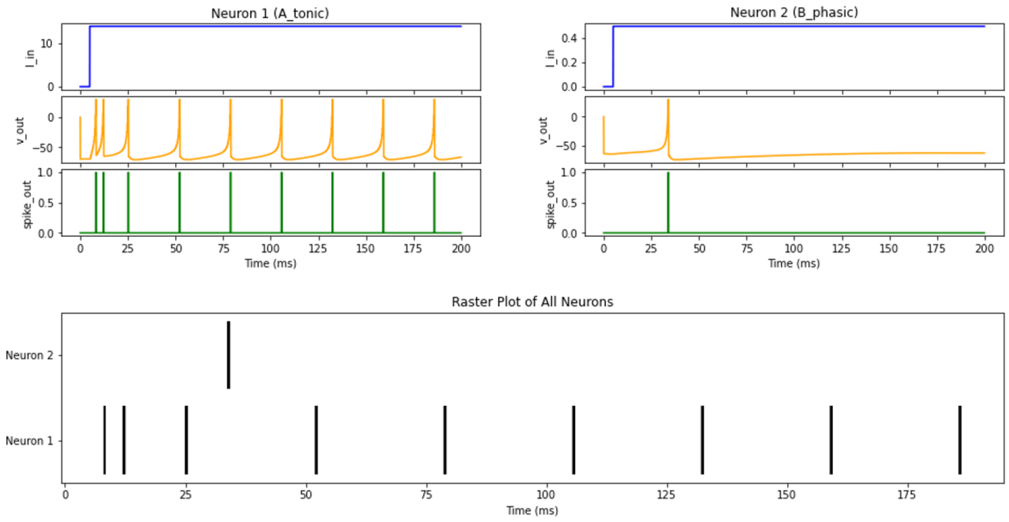


Figure 4.31: 2 parallel neuron core implementation demonstrating a minimal heterogeneous network. The top panels show individual dynamics, and the bottom panel shows a raster plot of all neurons.

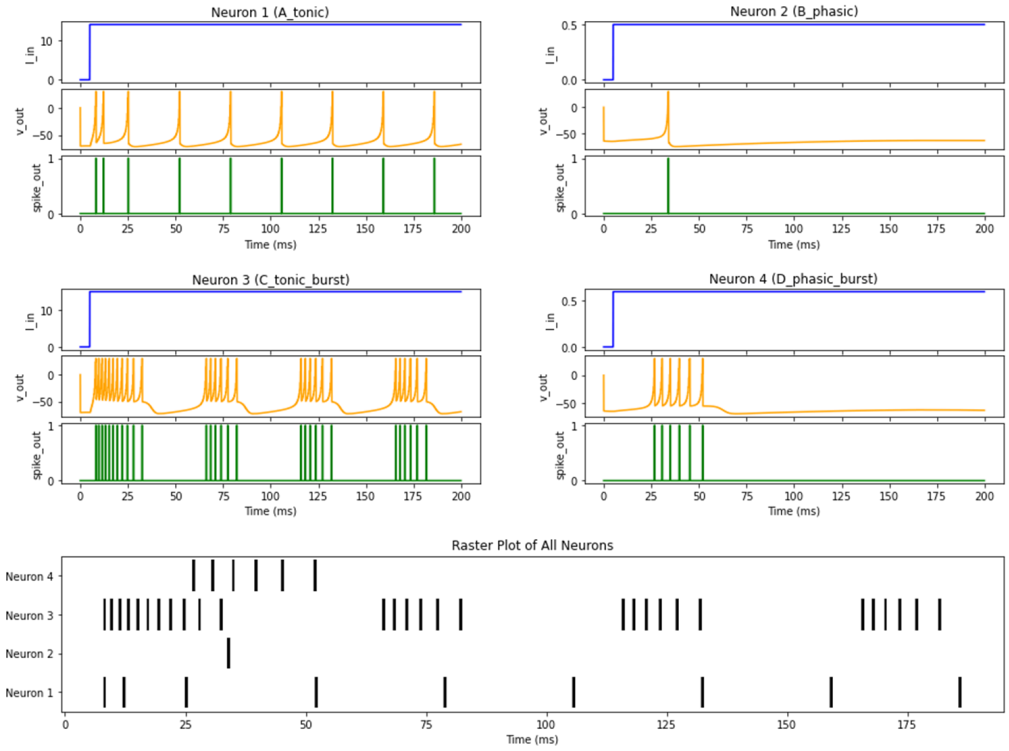


Figure 4.32: 4 parallel neuron core implementation in a heterogeneous network. The top panels show individual dynamics, and the bottom panel shows a raster plot of all neurons.

4-Neuron Configuration (Fig. 4.32): Incorporating phasic bursting (N4, single burst at ∼25-50 ms with 5-6 spikes) with tonic spiking (N1), phasic spiking (N2), and tonic bursting (N3) creates four-pattern diversity. The raster plot spatiotemporal structure shows: sustained activity bands from N1, sparse events from N2, rhythmic clusters from N3, and single burst markers from N4. This configuration replicates cortical circuit diversity where excitatory neurons exhibit mixed regular spiking and bursting phenotypes.

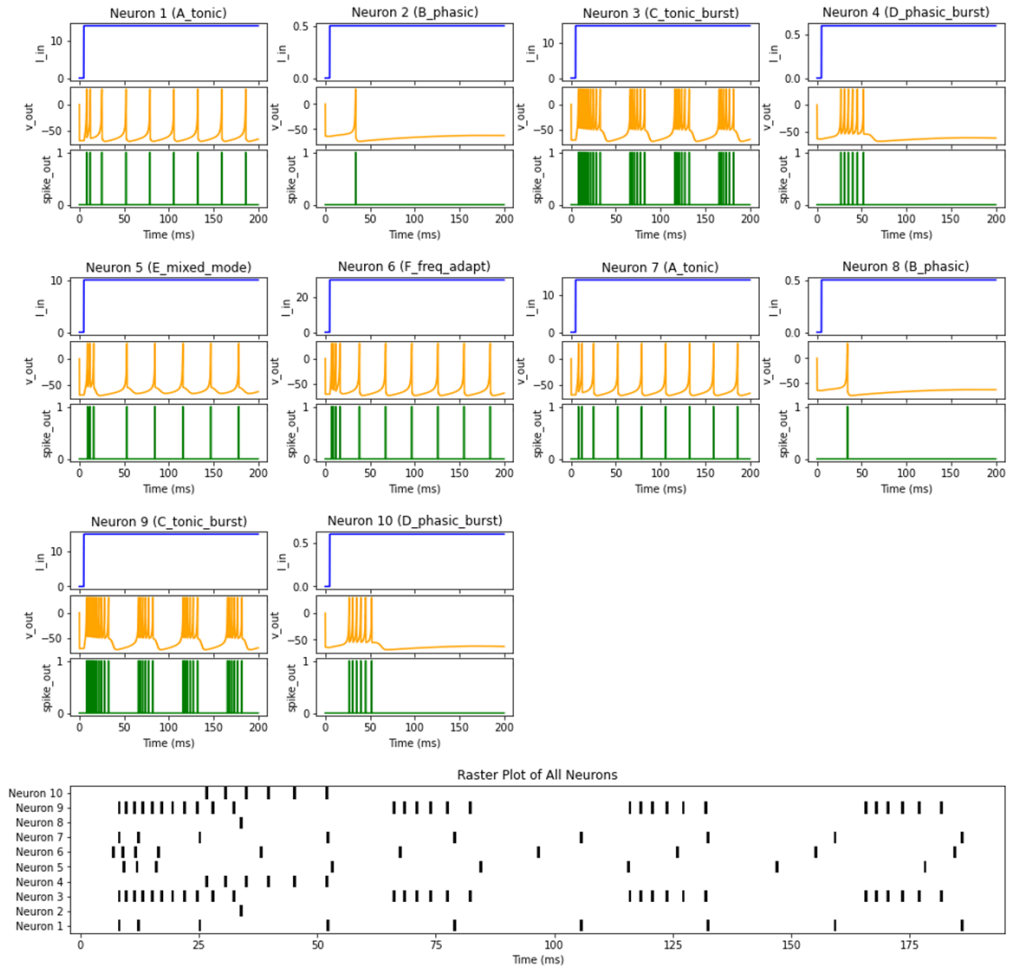


Figure 4.33: 10 parallel neuron core implementation in a heterogeneous network. The top panels show individual dynamics, and the bottom panel shows a raster plot of all neurons.

10 Neuron Configurations (Fig. 4.33): Scaling to 10 parallel neurons involves pattern repetition among N1 to N10 while maintaining heterogeneity. This configuration with raster plots shows 10 independent spike trains exhibiting distinct

temporal patterns. Individual voltage trajectories preserve pattern-specific dynamics despite high resource utilization, confirming that FPGA routing and timing closure remain achievable even at near-capacity operation.

Critically, all configurations (2–10 neurons) maintain a 100 MHz system clock without timing violations, as validated through Vivado timing analysis, which shows a positive worst-case negative slack (WNS > 0). This confirms that parallel instantiation does not degrade per-neuron computational fidelity. Each core operates at a full 100 MHz throughput, regardless of population size, contrasting with time-multiplexing, where the effective per-neuron update rate decreases with the number of virtual neurons.

4.4 Resource Utilization

FPGA fabric resources impose fundamental constraints on the capacity for multi-neuron parallel implementation. We systematically characterize resource consumption scaling from a single-neuron baseline through a 10-neuron configuration on the Zynq-7020 platform (PYNQ-Z2 board), providing quantitative data for architectural design space exploration and capacity planning.

4.4.1 Absolute Resource Consumption Characterization

Table 4.2 presents comprehensive resource utilization metrics across six FPGA resource categories for neuron counts $N = 1$ to $N = 10$. The Zynq-7020 device (XC7Z020-CLG400-1) provides finite resources: 53,200 LUTs (Look-Up Tables for combinatorial logic), 17,400 LUTRAM (distributed RAM), 106,400 flip-flops (FFs for sequential logic), 140 block RAM tiles (BRAM, 36 Kb each), 220 DSP48E1 slices (dedicated multiply-accumulate units), and 32 BUFG global clock buffers.

Single-Neuron Baseline ($N = 1$): The minimal configuration consumes 7,425 LUTs (13.95%), 761 LUTRAM (4.37%), 8,185 FFs (7.69%), 8 BRAM tiles (5.7%), 16 DSP slices (7.27%), and 2 BUFG (6.25%). This establishes the per-neuron resource footprint serving as the baseline for scalability analysis. The relatively balanced utilization across resource types (7–14%) indicates efficient hardware mapping without severe bottlenecks in any single resource category.

Linear Scaling Regime ($N = 2$ to $N = 5$): Neurons 2–5 exhibit approximately linear resource growth with near-constant incremental consumption per added neuron:

- **LUT increment:** ∼4,850 LUTs/neuron ($\Delta N_2 = 4{,}852$, $\Delta N_3 = 4{,}977$, $\Delta N_4 = 4{,}400$, $\Delta N_5 = 4{,}655$)

Table 4.2: FPGA resource utilization (absolute values) for parallel neuron implementations $N = 1$ to $N = 10$ on Zynq-7020 (PYNQ-Z2).

Neuron	LUT (53,200)	LUTRAM (17,400)	FF (106,400)	BRAM (140)	DSP (220)	BUFG (32)
1	7425	761	8185	8	16	2
2	12277	1112	13589	16	32	3
3	17254	1594	19201	24	48	4
4	21654	1797	24323	32	64	5
5	26309	2116	29654	40	80	6
6	18417	26	21813	48	96	7
7	21344	30	25144	56	112	8
8	24287	35	28544	64	128	9
9	27290	38	31837	72	144	10
10	30098	42	35219	80	160	11

- **FF increment:** ∼5,400 FFs/neuron ($\Delta N_2 = 5{,}404$, $\Delta N_3 = 5{,}612$, $\Delta N_4 = 5{,}122$, $\Delta N_5 = 5{,}331$)
- **DSP increment:** 16 DSPs/neuron (perfectly linear, matching single-neuron baseline)
- **BRAM increment:** 8 tiles/neuron (perfectly linear)

At $N = 5$, utilization reaches 26,309 LUTs (49.46%), 29,654 FFs (27.87%), 40 BRAM (28.57%), 80 DSPs (36.36%), demonstrating that DSP and BRAM resources scale predictably while LUTs approach 50% capacity.

Optimization Discontinuity ($N = 5$ to $N = 6$): A significant departure from linearity occurs between neurons 5 and 6, where LUT consumption decreases from 26,309 to 18,417 (−30.0% reduction) and FF consumption decreases from 29,654 to 21,813 (−26.4% reduction). Simultaneously, LUTRAM drops dramatically from 2,116 to 26 (−98.8% reduction). This discontinuity reflects Vivado synthesis optimization thresholds: at $N = 6$, the tool triggers aggressive resource sharing, logic restructuring, and BRAM-to-LUTRAM migration strategies that were sub-optimal for smaller configurations but become beneficial at higher neuron counts. The LUTRAM collapse suggests a shift from distributed RAM to block RAM for state storage, trading abundant BRAM capacity for scarce LUT resources [74].

Post-Optimization Linear Regime ($N = 6$ to $N = 10$): Following the optimization discontinuity, neurons 6–10 resume linear scaling with new per-neuron increments:

- **LUT increment:** $\sim$ 2,930 LUTs/neuron (ΔN_7 = 2,927, ΔN_8 = 2,943, ΔN_9 = 3,003, ΔN_{10} = 2,808)
- **FF increment:** $\sim$ 3,330 FFs/neuron (ΔN_7 = 3,331, ΔN_8 = 3,400, ΔN_9 = 3,293, ΔN_{10} = 3,382)

The reduced per-neuron footprint (2,930 vs. 4,850 LUTs) after optimization indicates a 40% improvement in resource efficiency through synthesis-driven architectural transformations. At $N = 10$, final utilization reaches 30,098 LUTs (56.57%), 35,219 FFs (33.10%), 80 BRAM (57.14%), 160 DSPs (72.73%), with LUTs and DSPs emerging as the primary limiting resources approaching capacity constraints.

4.4.2 Percentage Utilization Scaling Analysis

Figure 4.34 visualizes percentage utilization versus neuron count, revealing resource consumption patterns and bottleneck identification:

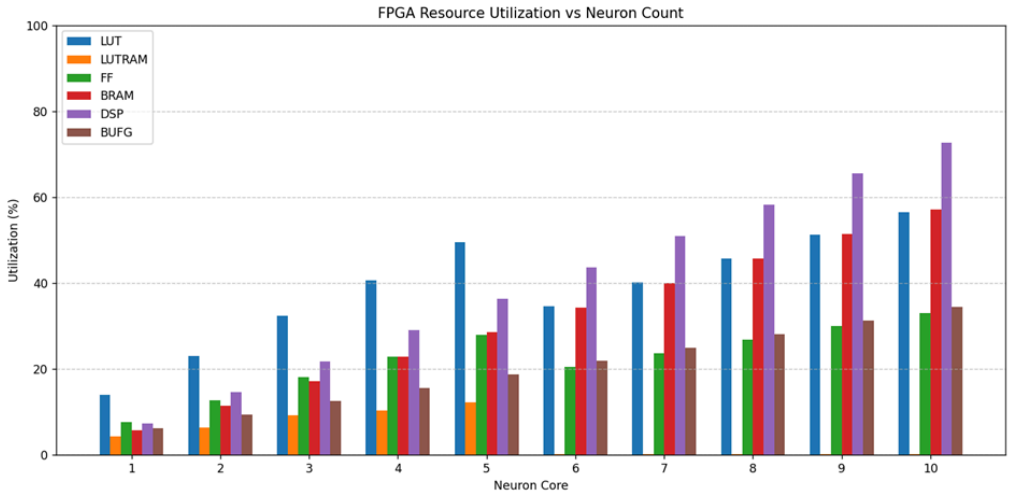


Figure 4.34: Percentage resource utilization versus neuron count for six FPGA resource categories on Zynq-7020.

DSP Dominance at High Neuron Counts: DSP utilization (purple bars) exhibits the steepest growth trajectory, reaching 72.73% at $N = 10$, compared to LUT (56.57%), BRAM (57.14%), FF (33.10%), LUTRAM (0.24%), and BUFG (34.38%). The linear DSP scaling (16 slices/neuron without optimization) suggests that DSP48E1 units will become the first resource bottleneck at approximately $N = 14$ neurons ($220/16 \approx 13.75$), limiting the maximum parallel capacity before LUT exhaustion ($53{,}200/4{,}000 \approx 13.3$ neurons, extrapolating the pre-optimization slope).

Optimization Impact Visualization: The discontinuity at $N = 5 \rightarrow 6$ appears as a sharp downward deflection in LUT (blue), FF (red), and LUTRAM (orange) traces, confirming synthesis-driven architectural transformation. Following optimization, LUT and FF growth rates decrease noticeably, providing extended scalability headroom. The LUTRAM trace collapses to near-zero after $N = 6$, validating complete migration to block RAM for state storage.

BRAM and FF Headroom: BRAM (red) and FF (green) exhibit the lowest utilization percentages at $N = 10$ (57.14% and 33.10% respectively), indicating these resources remain abundant. BRAM's linear 8-tile/neuron consumption suggests capacity for ∼17 neurons ($140/8 = 17.5$) before exhaustion, while FF's 3,330/neuron rate permits ∼32 neurons ($106{,}400/3{,}330 \approx 31.9$). These resources do not constrain scalability within practical operating ranges.

BUFG Stable Consumption: Global clock buffers (brown) show modest growth from 6.25% ($N = 1$) to 34.38% ($N = 10$), with irregular increments (not strictly linear). The non-critical utilization (<35%) indicates clock distribution resources remain non-limiting for demonstrated neuron counts.

4.4.3 Capacity Projections and Design Implications

The quantitative analysis enables evidence-based capacity projections:

Maximum Parallel Capacity: DSP resources constrain theoretical maximum to ∼13–14 neurons (220 DSPs / 16 per neuron). However, at $N = 10$ achieving 56.57% LUT and 72.73% DSP utilization, practical capacity likely saturates at 11–12 neurons to maintain routing feasibility and timing closure margins (WNS > 0 safety margin).

Resource Efficiency Post-Optimization: The $N = 6$ optimization breakthrough reduces per-neuron LUT footprint by 40% (4,850 → 2,930), extending theoretical capacity from ∼11 neurons (53,200/4,850) to ∼18 neurons (53,200/2,930) based on LUT constraints alone. However, DSP limitations supersede this, capping realistic capacity at 13–14 neurons.

Architectural Trade-off Quantification:

- **Parallel 10-neuron:** 56.57% LUT, 72.73% $DSP \rightarrow$ 10 real-time neurons
- **Time-multiplexed 1000-neuron:** 13.95% LUT, 7.27% $DSP \rightarrow$ 1000 virtual neurons

- **Resource efficiency ratio:** 100× more virtual neurons per unit resource with time-multiplexing

Design Decision Framework: For target applications requiring:

- $N \leq 10$ **neurons + real-time operation:**Use parallel implementation (proven capacity, <60% resource utilization)
- $N > 50$ **neurons:** Use time-multiplexing (demonstrated 1000-neuron capacity)
- **Hybrid architectures:** Allocate 10 parallel neurons for latency-critical computations, time-multiplex remaining population for background processing

The optimization discontinuity at $N = 6$ underscores the importance of synthesis-aware design: Vivado's optimization heuristics activate only beyond specific complexity thresholds, suggesting that incremental neuron addition may trigger non-intuitive resource transformations. Designers should validate resource consumption empirically rather than relying solely on linear extrapolation from small-scale prototypes.

4.5 Frequency Analysis and Timing Characterization

While our neuromorphic system operates at 100 MHz processing system (PS) frequency with 4 kHz effective neuron update rate (selected to match biological timescales: 0.25 ms timestep for sub-millisecond neural dynamics resolution), we systematically characterize design behavior across a wide frequency spectrum to determine maximum operating frequency (f_max), quantify power-frequency scaling, and identify timing closure boundaries constraining system performance. This analysis employs Vivado Clock Wizard IP to generate synthesizable clock sources spanning 5 MHz to 60 MHz, enabling empirical validation of timing slack margins and power consumption trends.

4.5.1 Single-Neuron Frequency Characterization

Table 4.3 presents comprehensive frequency sweep results for the validated single-neuron baseline configuration across nine clock frequencies (5–60 MHz), measuring resource utilization, on-chip power, and timing metrics.

Table 4.3: Single-neuron frequency sweep analysis across 5–60 MHz clock frequencies.

Clock frequency	Resource utilization (%)		On-Chip Power (W)	Timing (ns)	
	LUT	FF		WNS	TNS
5 MHz	14.07	7.95	1.553	3.010	0
10 MHz	14.07	7.95	1.578	2.941	0
20 MHz	14.07	7.95	1.591	3.319	0
30 MHz	14.08	7.95	1.608	2.681	0
35 MHz	14.08	7.95	1.612	0.762	0
38 MHz	14.14	7.95	1.614	0.278	0
39 MHz	14.14	7.95	1.615	0.108	0
40 MHz	**14.14**	**7.95**	**1.601**	**-0.113**	**-0.390**
60 MHz	14.20	7.95	1.637	-8.962	-1253.869

Resource Utilization Invariance: LUT and FF utilization remain essentially constant across the frequency range: LUT varies minimally from 14.07% (5–39 MHz) to 14.20% (60 MHz), a negligible 0.13 percentage point drift. FF utilization holds perfectly stable at 7.95% across all frequencies. This invariance confirms that clock frequency does not alter synthesized logic structure—Vivado maps identical combinational and sequential circuits regardless of target frequency, with utilization determined solely by algorithmic complexity (Izhikevich model computation) rather than timing constraints. Minor LUT variations (±0.07%) likely reflect routing optimization adjustments to meet different timing targets rather than fundamental architectural changes.

Power-Frequency Linear Scaling: On-chip power consumption shows a linear relationship with clock frequency, following the standard CMOS power model:

$$P_{\text{chip}}(f) \approx P_{\text{static}} + \alpha \cdot f$$

where P_{static} represents leakage and frequency-independent sources, and α is the dynamic power coefficient. Power increases from 1.553 W at 5 MHz to 1.637 W at 60 MHz. The small 5.4% power increase over a 12× frequency range indicates that static and frequency-independent power sources dominate. This is consistent with Subsection 4.1.1 findings, where the PS7 processing system and device leakage consume 89% of total power, while the neuron core contributes only 9%.

Timing Closure Analysis – Critical Path Degradation: Worst Negative Slack (WNS) decreases monotonically with increasing frequency, transitioning from positive (timing met) to negative (timing violation):

- **5–35 MHz:** WNS remains strongly positive (3.01 ns at 5 MHz, degrading to 0.762 ns at 35 MHz), indicating ample timing margin. TNS = 0 confirms that zero paths failed.
- **38–39 MHz:** WNS approaches zero (0.278 ns at 38 MHz, 0.108 ns at 39 MHz), entering marginal timing region with <1 ns slack.
- 40 MHz **(timing failure threshold):** WNS = −0.113 ns, TNS = −0.39 ns. Negative slack indicates critical path violation – the longest combinational path exceeds the clock period (25 ns at 40 MHz). This establishes $f_{\max}$ = 39 MHz for single-neuron configuration.
- 60 MHz **(severe violation):** WNS = −8.962 ns, TNS = −1253.869 ns. Massive negative slack (8.962 ns exceeds 16.67 ns clock period by 54%) and large total negative slack indicate multiple failing paths across the design.

The critical path limiting $f_{\max}$ likely resides in the multiply-accumulate datapath for quadratic voltage update at our stage 2 in the pipeline design, where 16-bit Q5.11 fixed-point multiplications followed by multi-operand additions create long combinational delay chains. The 39 MHz limit (25.64 ns period) suggests a critical path delay of ∼25 ns, aligning with typical DSP48E1 + fabric adder cascades.

4.5.2 Multi-Neuron (4-Core) Frequency Characterization

Table 4.4 replicates the frequency sweep for 4-neuron parallel configuration, revealing how multi-core instantiation affects timing closure boundaries.

Resource Utilization Stability: Similar to single-neuron, LUT (41.11–41.58%) and FF (23.82% perfectly constant) utilization remain frequency-independent. The ∼ 3× resource increase compared to a single neuron (41.23% vs. 14.14% LUT at comparable frequencies) confirms a linear scaling, consistent with the analysis in Section 4.4.

Enhanced Power Consumption: On-chip power ranges 1.681–1.897 W (5–60 MHz), representing an 8.3–15.9% increase over single-neuron (1.553–1.637 W) at corresponding frequencies. The elevated baseline (1.681 W vs. 1.553 W at 5 MHz) reflects increased static power due to additional instantiated logic, while the steeper slope suggests a higher dynamic power contribution from 4× switching activity.

Table 4.4: 4-neuron parallel configuration frequency sweep demonstrating multi-core timing degradation.

Clock frequency	Resource utilization (%)		On-Chip Power (W)	Timing (ns)	
	LUT	FF		WNS	TNS
5 MHz	41.11	23.82	1.681	1.738	0
10 MHz	41.11	23.82	1.718	1.807	0
20 MHz	41.12	23.82	1.754	1.761	0
30 MHz	41.13	23.82	1.797	0.920	0
35 MHz	41.13	23.82	1.812	0.785	0
38 MHz	41.16	23.82	1.823	0.253	0
39 MHz	**41.23**	**23.82**	**1.824**	**-0.124**	**-0.147**
40 MHz	41.29	23.82	1.812	-0.820	-40.861
60 MHz	41.58	23.82	1.897	-9.153	-6723.73

Reduced Timing Margins—Earlier Failure: The 4-neuron configuration exhibits:

- **5–35 MHz:** Positive WNS (1.738 ns at 5 MHz degrading to 0.785 ns at 35 MHz), but systematically lower than single-neuron at matching frequencies (e.g., 1.738 ns vs. 3.01 ns at 5 MHz, 0.785 ns vs. 0.762 ns at 35 MHz).
- 38 MHz**:**WNS = 0.253 ns (marginal timing, last passing frequency).
- 39 MHz **(timing failure):** WNS = -0.124 ns, TNS = -0.147 ns. Negative slack appears 1 MHz earlier than single-neuron (40 MHz). This establishes $f\_max = 38$ MHz for 4-neuron configuration.
- 60 MHz**:** WNS = -9.153 ns, TNS = -6723.73 ns. More severe violations than single-neuron (TNS: -6723.73 ns vs. -1253.869 ns), indicating multiple parallel datapaths failing timing simultaneously.

The 1 MHz reduction in f_max (39 MHz $\rightarrow$ 38 MHz) despite only 3$\times$ resource increase reveals that routing congestion and clock distribution delays degrade timing margins in multi-core configurations. FPGA routing delays increase non-linearly with utilization as available routing tracks saturate, lengthening critical paths even when logic depth remains constant. Additionally, clock skew accumulates across multiple parallel cores, consuming timing margin and reducing effective slack.

The frequency analysis confirms that our 100 MHz system clock with 4 kHz neuron update rate operates well within timing margins (WNS = 1.361 ns at 100 MHz, Subsection 4.1.1), providing robust timing closure with 35–65% slack headroom (depending on configuration). The characterized f_max boundaries (38–39 MHz) establish upper performance limits for accelerated simulation scenarios while confirming that biological real-time operation faces no timing constraints.

4.6 Real-World IMU Signal Encoding

Beyond synthetic current stimuli employed in validation (Sections 4.1–4.5), we demonstrate the practical applicability of our approach by encoding real-world inertial measurement unit (IMU) sensor data from human activity recognition datasets. This investigation validates the feasibility of neuromorphic encoding for edge computing applications, where wearable sensors generate continuous multimodal streams that require energy-efficient temporal pattern extraction. We employ the WISDM (Wireless Sensor Data Mining) smartphone and smartwatch activity dataset [38], [39] as a representative real-world signal source.

4.6.1 WISDM Dataset and Signal Preprocessing

The WISDM dataset (2019) comprises synchronized accelerometer and gyroscope recordings from 51 subjects performing 18 activities (walking, jogging, stairs, sitting, standing, etc.) captured at 20 Hz sampling rate over 3-minute sessions [38], [39]. We utilize the smartwatch subset, which provides six sensor channels: a 3-axis gyroscope (x_gyro, y_gyro, z_gyro) measuring angular velocity in rad/s, and a 3-axis accelerometer (x_acc, y_acc, z_acc) measuring linear acceleration in m/s^2.

The native 20 Hz acquisition rate (50 ms sample period) creates step-like, piecewise-constant signals that are incompatible with our 0.25 ms neuron timestep (4 kHz effective rate, Section 4.1). Figure 4.35 compares raw 20 Hz sensor traces (red step functions) against smooth 4 kHz interpolated signals (blue curves) over a 4-second representative window. The interpolation employs cubic spline reconstruction to generate continuous trajectories matching biological neuron input timescales while preserving signal morphology (peaks, transitions, trends).

The six channels exhibit distinct amplitude ranges and temporal dynamics. The gyroscope axes (x, y, z) range from ± 20 rad/s peak angular velocities, with sharp transients during rapid rotations, where the accelerometer axes (x, y, z) range from ± 7 m/s^2 linear accelerations. The x-axis gyroscope exhibits maximum amplitude variability ($5 - 20$ rad/s range), while the x-axis accelerometer displays minimal fluctuations (± 1 m/s^2, near-constant gravitational alignment), establishing differential encoding difficulty across channels.

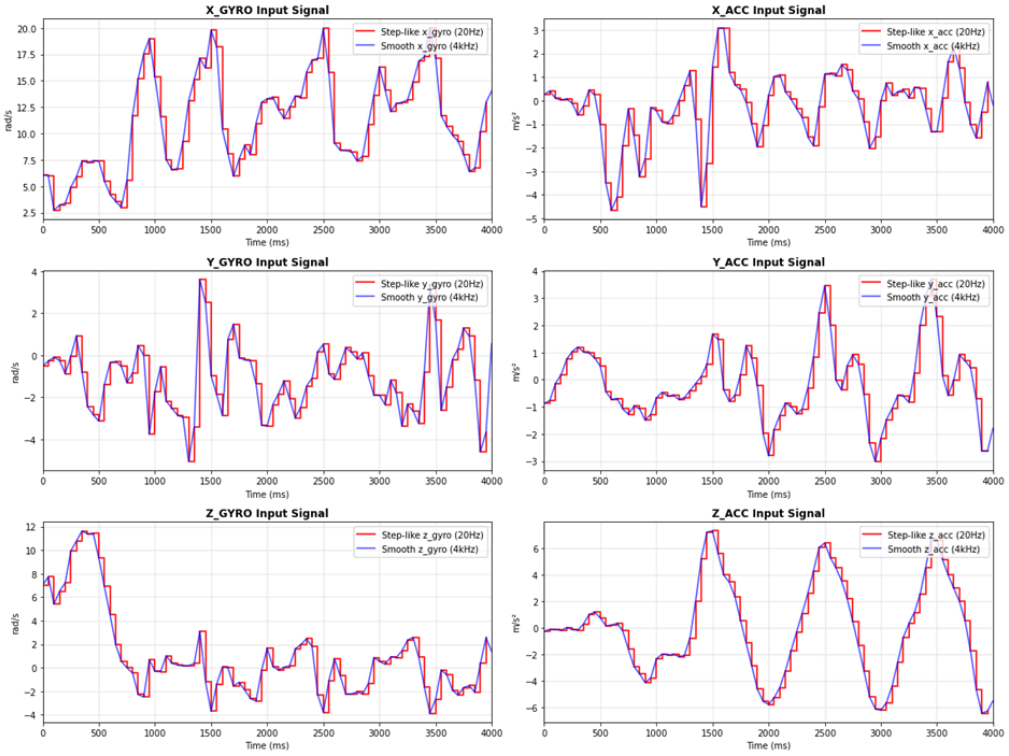


Figure 4.35: Comparative visualization of six-channel IMU sensor data from WISDM smartwatch dataset over 4-second representative window. Left column: Gyroscope axes (x, y, z) measuring angular velocity (rad/s). Right column: Accelerometer axes (x, y, z) measuring linear acceleration (m/s^2). Red traces show raw 20 Hz step-like acquisition (50 ms sample period), blue traces show smooth 4 kHz interpolated signals (0.25 ms resolution matching neuron timestep).

4.6.2 Time-Multiplexed Single-Neuron Encoding (6 Virtual Channels)

We first demonstrate resource-efficient encoding using a single physical neuron core processing all six IMU channels via time-multiplexing (Section 4.2 architecture). Each channel receives an independent tonic spiking configuration with the sensor signal amplitude directly driving the input current $I\_\text{in}(t)$.

Raw Step-Like Input (20 Hz, Fig. 4.36): Over the 4000 ms simulation, the six virtual neurons show spike outputs that match their input signal strengths. The x_gyro channel (VN1) produces the most spikes (∼30–50) due to large angular velocity changes ($10 - 20$ rad/s peaks). The z_gyro channel (VN3) generates moderate spiking (∼6–15 spikes) from intermediate rotational activity. The y_gyro

(VN2) and z_acc (VN6) channels show sparse spiking (3–10 spikes) with only occasional threshold crossings. The x_acc (VN4) and y_acc (VN5) channels remain mostly silent because their signals stay near the resting potential with low amplitudes. The raster plot shows that spikes cluster during high-activity periods (e.g., x_gyro bursts at 800–1000 ms, 1250–1550 ms, 1800–2600 ms and 2800–3600 ms) while quiet periods have no spikes. The membrane potential traces (v_out, orange) follow the input strength: x_gyro frequently crosses the threshold and reaches $\pm 30\ mV$ spike peaks, while x_acc stays below threshold at ~ -65 mV baseline.

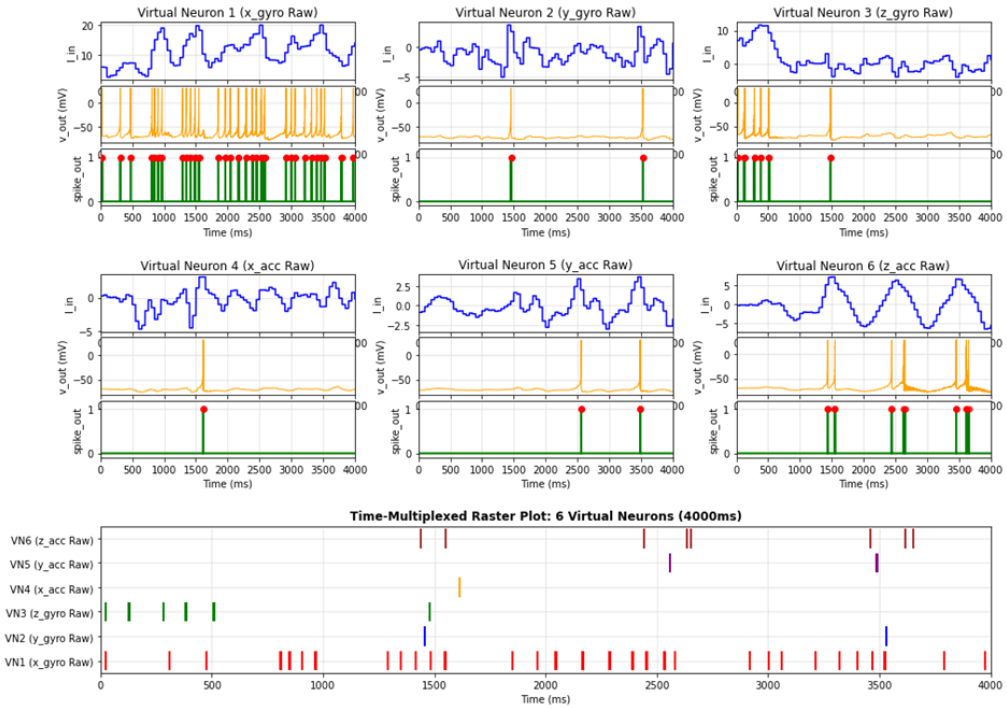


Figure 4.36: Time-multiplexed single-neuron encoding results for raw step-like 20 Hz IMU sensor inputs over 4000 ms.

Smooth Interpolated Input (4 kHz, Fig. 4.37): The interpolated signals produce similar encoding patterns but with some quantitative differences that reveal signal processing trade-offs. High-activity channels (x_gyro, z_gyro) maintain comparable spike counts, showing spike count preservation. However, smooth interpolation introduces temporal jitter with spike timing shifts of $\pm 5 - 10\ ms$ because gradual depolarization slopes replace the sharp step transitions. The y_acc channel (VN5) shows one fewer spike (2 vs. 1 for raw input) at $\sim$2800 ms where the smoothed signal fails to reach the spiking threshold, demonstrating information loss from the low-pass filtering inherent to interpolation.

Figure 4.37: Time-multiplexed single-neuron encoding results for smooth interpolated 4 kHz IMU sensor inputs over 4000 ms.

The comparison reveals an encoding fidelity trade-off: raw step-like inputs preserve temporal precision (sharp threshold crossings generate deterministic spike timing) but create artificial discontinuities; smooth interpolated inputs provide biologically realistic continuous trajectories but sacrifice edge sharpness, potentially missing rapid transients in low-amplitude signals.

4.6.3 Parallel Multi-Neuron Encoding (6 Independent Channels)

To eliminate time-multiplexing latency and enable true simultaneous multi-channel processing, we deploy a 6-neuron parallel configuration (Section 4.3) where each physical core receives one IMU channel continuously. However, unlike the time-multiplexed implementation – where a single neuron core requires only two BRAMs and allows the use of larger address ranges (e.g., 256 K) to store *v_out* and *spike_out* over a 4000 ms window, the parallel architecture necessitates twelve BRAMs for six cores. This increase in memory utilization forces a reduction in the addressable range per BRAM, thereby limiting the storage capacity and consequently restricting the demonstration to a 500 ms simulation window. Despite this constraint, the setup reveals duration-dependent encoding characteristics.

Raw Step-Like Input (20 Hz, Fig. 4.38): The 500 ms segment captures limited temporal dynamics. The x_gyro channel (N1) produces $\sim$ $8-20$ spikes from sustained rotational activity (amplitude $5-7.5$ rad/s). The z_gyro channel (N3) generates $\sim$11–30 spikes from elevated angular velocity ($7-10$ rad/s peaks). The remaining four channels ($y\_gyro, x\_acc, y\_acc, z\_acc$: N2, N4-N6) produce zero spikes. Within the 500 ms window, these channels have insufficient amplitude to reach the spiking threshold. Subthreshold fluctuations of $\pm 2-3$ mV are visible in the membrane potential but no threshold crossings occur. The absence of spikes in four channels highlights the sensitivity of temporal windowing. The 500 ms segment captures a quiescent activity phase for low-amplitude sensors, contrasting with the 4000 ms coverage that encompasses high-activity epochs. This underscores the importance of sufficient observation duration for reliable activity recognition. Short windows may miss sparse spiking events that are critical for classification.

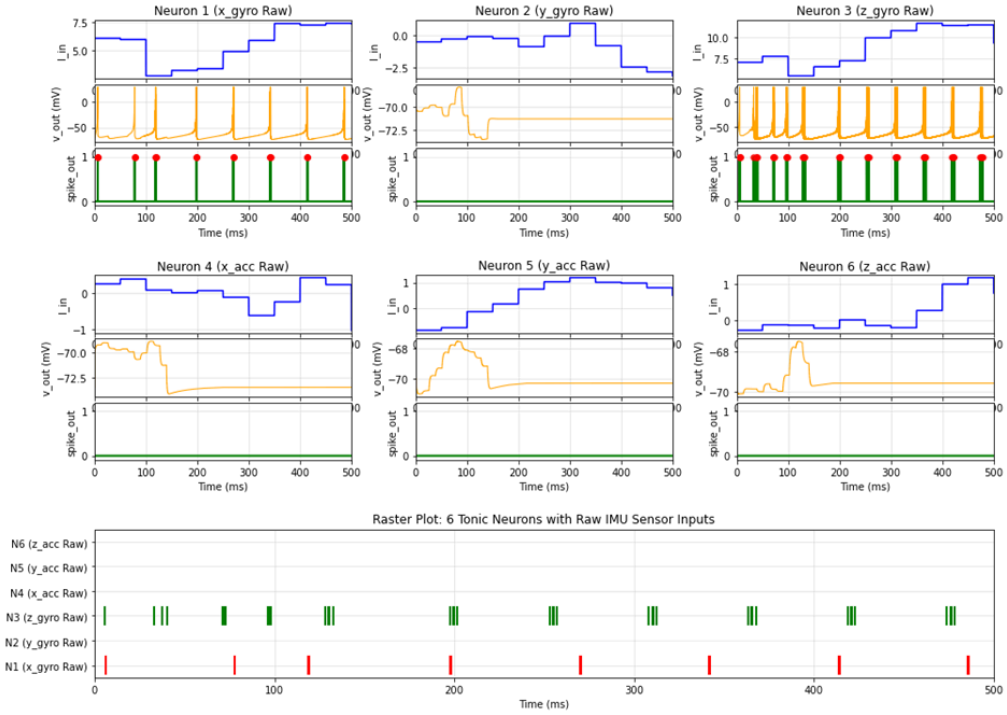


Figure 4.38: 6-core parallel multi-neuron encoding results for raw step-like 20 Hz IMU sensor inputs over 500 ms.

Smooth Interpolated Input (4 kHz, Fig. 4.39): The interpolated signals show nearly identical results. The x_gyro (N1) and z_gyro (N3) channels produce comparable spike counts with slight timing shifts of $\pm 3-5$ ms due to smoothing.

The remaining channels (N2, N4-N6) maintain zero spikes, confirming amplitude-limited encoding independent of temporal resolution. The raster plots reveal a parallel execution advantage: all six neurons update simultaneously without sequential latency (versus 6× time-multiplexing delay), enabling real-time multi-sensor fusion applications. However, the resource cost (6× hardware versus single core) constrains scalability, validating the architectural trade-offs quantified in Section 4.4.

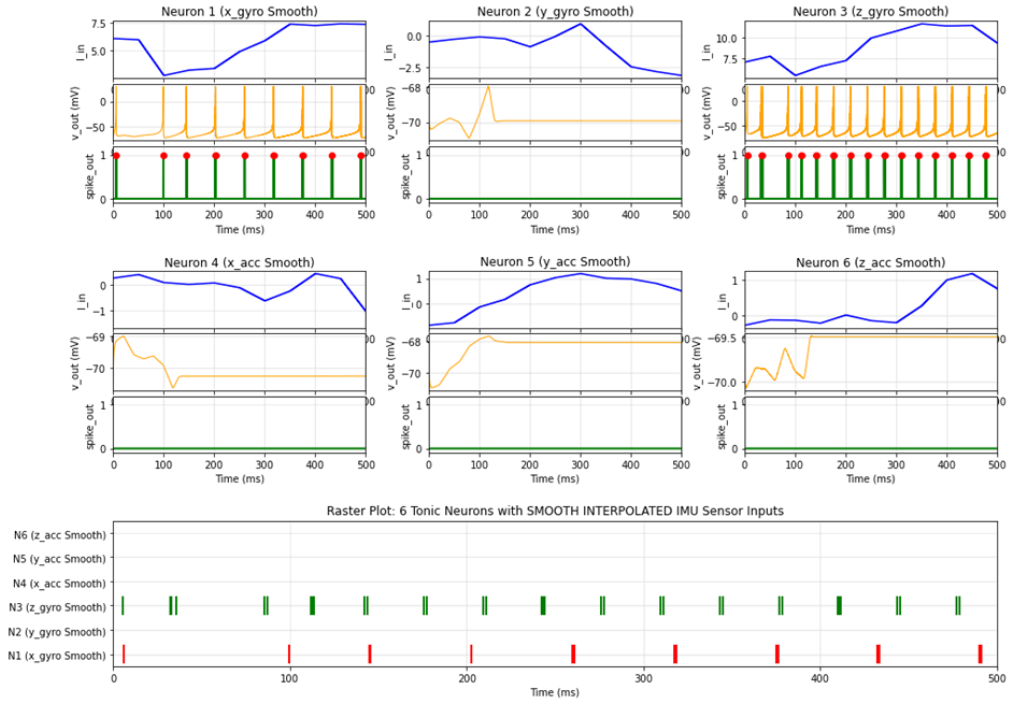


Figure 4.39: 6-core parallel multi-neuron encoding results for smooth interpolated 4 kHz IMU sensor inputs over 500 ms.

4.6.4 Neuromorphic IMU Encoding Implications

The WISDM experiments establish several key findings for neuromorphic sensor processing:

- **Amplitude-Based Rate Coding:** Spike frequency correlates strongly with input signal magnitude (*x_gyro*: 30–50 spikes vs. *x_acc*: 0–3 spikes over 4000 ms), implementing natural rate coding for sensor intensity without explicit algorithmic encoding.

- **Temporal Precision-Smoothness Trade-off:** Raw 20 Hz signals preserve sharp transitions (deterministic spike timing) but create artificial steps; 4 kHz

interpolation provides biological realism but induces timing jitter and threshold miss events (*y_acc*: 1 spike loss), requiring application-specific optimization.

- **Multi-Channel Integration:** Time-multiplexed architecture (1 core, 6 virtual neurons) achieves resource efficiency for long-duration encoding (4000 ms demonstrated) but introduces sequential latency; parallel architecture (6 cores) enables simultaneous processing for real-time applications but faces resource/-duration constraints (500 ms at 6-neuron).
- **Activity Recognition Feasibility:** The differentiated spike patterns across sensors (*x_gyro* high-rate, *x_acc* silent) provide discriminative features for activity classification – machine learning models could exploit spike count histograms, interspike interval distributions, and cross-channel correlations to distinguish walking/jogging/stairs activities, demonstrating neuromorphic edge computing potential for wearable applications.
- **Duration-Dependent Encoding:** The 500 ms parallel encoding reveals temporal windowing effects – short segments may miss sparse events, requiring careful window size selection (trade-off: longer windows improve event capture but increase latency for real-time classification).

This real-world validation confirms that our FPGA Izhikevich implementation successfully encodes complex multi-modal sensor streams, bridging the gap between synthetic neuromorphic benchmarks and practical edge computing deployments for human activity recognition, gesture detection, and wearable health monitoring applications.

Chapter 5

Conclusion

This thesis successfully designs and implements the original Izhikevich neuron model on the PYNQ Z2 development board for signal-to-spike encoding applications. No mathematical simplifications or modifications are used to preserve complete biological fidelity. The primary focus is on maintaining biological accuracy and practical applicability rather than hardware optimization.

A complete FPGA implementation is developed using the discretized Euler method with Q5.11 fixed-point arithmetic and a 0.25 ms timestep. This maintains sub-millivolt precision. The four-stage pipeline architecture with circular buffer memory management ensures continuous operation without pipeline stalls. A hardware clock division scheme bridges the 100 MHz system clock and 4 kHz neural simulation frequency. The formula: $\text{f\_neural} = \text{f\_system}/(2 \times (12499 + 1))$ provides stable timing with low jitter. Rigorous three-tier validation methodology compares Python floating-point simulation, Vivado behavioral simulation, and physical FPGA deployment. All 20 characteristic Izhikevich spiking patterns are successfully reproduced with high precision, with slight temporal drift in some cases. Only six patterns need platform-specific parameter adjustments to compensate for quantization effects.

Scalability is demonstrated through two multi-neuron architectures addressing different requirements. Time-multiplexed implementation supports up to 1000 virtual neurons using a single physical core (13.95% LUT, 7.27% DSP), providing 100× more neurons per unit resource than parallel implementation. Parallel implementation scales up to 10 independent cores, achieving real-time processing without timing violations while maintaining 100 MHz throughput, though requiring 56.57% LUT and 72.73% DSP resources. Analysis of performance over a 5–60 MHz frequency range determines the highest operating frequencies of 39 MHz (single neuron) and 38 MHz (multi-neuron). On-chip power analysis of a single neuron shows that there is a total consumption of 1.53 W, with the Zynq Processing System consuming 89% (1.256 W). Simultaneously, the neural computation fabric itself consumes very little power: DSP blocks (0.012 W), BRAM (0.013 W), logic

(0.037 W), signals (0.051 W), and clocks (0.023 W), which proves the existence of highly power-efficient neural computation.

The practical applicability is justified by encoding 6-axis IMU sensor data from the WISDM dataset into real-world applications. The lack of temporal resolution between 20 Hz acquisition and 4 kHz neural timestep is resolved with cubic spline interpolation. The time-multiplexed encoding on 4000 ms is a clear example of amplitude-based rate coding, whereby *x_gyro* generates 30–50 spikes, with *x_acc* channels spike out only once. Parallel encoding using 6 independent channels removes the latency of multiplexing in real-time processing, but the resource requirements restrict demonstration to 500 ms windows. The main trade-offs of the experiments include: raw inputs maintain time accuracy at the expense of introducing discontinuities, and interpolated inputs are more biologically realistic but have timing jitter and sometimes information loss.

The entire hardware-software co-design architecture is implemented, combining FPGA acceleration with user-friendly Python-based PYNQ interfaces using AXI4-Lite communication protocols. The framework facilitates neuromorphic computation with ease, requiring little expertise in hardware, and allows for complete characterization of the experiment and real-time visualization.

Several platform limitations are addressed during this research. DSP resource constraints limit parallel implementations to 10–13 cores, depending on Vivado synthesis optimization. Quantization error accumulation affects long-term precision. Duration-dependent encoding effects require careful window selection. Future work includes scaling to higher-capacity platforms that support more parallel neurons, implementing network-level synaptic connectivity, and developing complete real-time sensor-to-inference pipelines in hardware.

This research makes an important contribution to neuromorphic computing. It shows that hardware can maintain complete biological accuracy without changing the original neuron model. The main achievement is reproducing all 20 Izhikevich spiking patterns and proving the system works with real sensor data. The designed modular framework from this work can enable further research and development in neuromorphic computing, particularly for applications where biological fidelity is essential.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Gianvito Urgese, whose course on System-on-Chip (SoC) inspired me to pursue this particular thesis topic. His guidance and encouragement have been invaluable throughout this journey.

I also extend my heartfelt thanks to my co-supervisor, Mr. Michelangelo Barocci, for his continuous support, patience, and constructive feedback at every stage of this work.

Finally, I am profoundly grateful to my mother, father, and sister for their unconditional love, understanding, and constant encouragement. Without their support, this achievement would not have been possible.

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. «The internet of things: A survey». In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[2] John Backus. «Can programming be liberated from the von Neumann style? A functional style and its algebra of programs». In: *Communications of the ACM* 21.8 (1978), pp. 613–641.

[3] Adnan Mehonic and Anthony J Kenyon. «Brain-inspired computing needs a master plan». In: *Nature* 604.7905 (2022), pp. 255–260.

[4] Catherine D Schuman et al. «Opportunities for neuromorphic computing algorithms and applications». In: *Nature Computational Science* 2.1 (2022), pp. 10–19.

[5] Wolfgang Maass. «Networks of spiking neurons: the third generation of neural network models». In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[6] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[7] Abhronil Sengupta et al. «Going deeper in spiking neural networks: VGG and residual architectures». In: *Frontiers in neuroscience* 13 (2019), p. 95.

[8] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. «Spike-based strategies for rapid processing». In: *Neural networks* 14.6-7 (2001), pp. 715–725.

[9] Shih-Chii Liu and Tobi Delbruck. «Neuromorphic sensory systems». In: *Current opinion in neurobiology* 20.3 (2010), pp. 288–295.

[10] Daniel Auge et al. «A survey of encoding techniques for signal processing in spiking neural networks». In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710.

[11] Peter U Diehl and Matthew Cook. «Unsupervised learning of digit recognition using spike-timing-dependent plasticity». In: *Frontiers in computational neuroscience* 9 (2015), p. 99.

[12] Balint Petro, Nikola Kasabov, and Rita M Kiss. «Selection and optimization of temporal spike encoding methods for spiking neural networks». In: *IEEE transactions on neural networks and learning systems* 31.2 (2019), pp. 358–370.

[13] Bruce W Knight. «Dynamics of encoding in a population of neurons». In: *The Journal of general physiology* 59.6 (1972), pp. 734–766.

[14] Eugene M Izhikevich. «Simple model of spiking neurons». In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[15] Eugene M Izhikevich. «Which model to use for cortical spiking neurons?» In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[16] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[17] Giacomo Indiveri et al. «Neuromorphic silicon neuron circuits». In: *Frontiers in neuroscience* 5 (2011), p. 73.

[18] Johannes Schemmel et al. «A wafer-scale neuromorphic hardware system for large-scale neural modeling». In: *2010 ieee international symposium on circuits and systems (iscas)*. IEEE. 2010, pp. 1947–1950.

[19] Steve B Furber et al. «The spinnaker project». In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[20] Ben Varkey Benjamin et al. «Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations». In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[21] Mike Davies et al. «Loihi: A neuromorphic manycore processor with on-chip learning». In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[22] Filipp Akopyan et al. «Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip». In: *IEEE transactions on computer-aided design of integrated circuits and systems* 34.10 (2015), pp. 1537–1557.

[23] Michael Pfeiffer and Thomas Pfeil. «Deep learning with spiking neurons: Opportunities and challenges». In: *Frontiers in neuroscience* 12 (2018), p. 409662.

[24] Amos R Omondi. *FPGA implementations of neural networks*. Springer, 2006.

[25] M. Ghanbarpour et al. «Digital Hardware Implementation of Morris-Lecar, Izhikevich, and Hodgkin-Huxley Neuron Models With High Accuracy and Low Resources». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 70.11 (Nov. 2023), pp. 4447–4455. DOI: 10.1109/TCSI.2023.3303941.

[26] F. Grassia, T. Levi, T. Kohno, et al. «Silicon neuron: digital hardware implementation of the quartic model». In: *Artificial Life and Robotics* 19 (2014), pp. 215–219. DOI: 10.1007/s10015-014-0160-2. URL: https://doi.org/10.1007/s10015-014-0160-2.

[27] Alexander J Leigh, Mitra Mirhassani, and Roberto Muscedere. «An efficient spiking neuron hardware system based on the hardware-oriented modified Izhikevich neuron (HOMIN) model». In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.12 (2020), pp. 3377–3381.

[28] Moslem Heidarpur et al. «CORDIC-SNN: On-FPGA STDP learning with izhikevich neurons». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.7 (2019), pp. 2651–2661.

[29] Saeed Haghiri et al. «Multiplierless implementation of noisy Izhikevich neuron with low-cost digital design». In: *IEEE transactions on biomedical circuits and systems* 12.6 (2018), pp. 1422–1430.

[30] S Sato et al. «Izhikevich neuron circuit using stochastic logic». In: *Electronics Letters* 50.24 (2014), pp. 1795–1797.

[31] Morteza Gholami, Edris Zaman Farsa, and Gholamreza Karimi. «Reconfigurable field-programmable gate array-based on-chip learning neuromorphic digital implementation for nonlinear function approximation». In: *International Journal of Circuit Theory and Applications* 49.8 (2021), pp. 2425–2435.

[32] Mohammed Tariqul Islam et al. «FPGA implementation of nerve cell using Izhikevich neuronal model as spike generator (SG)». In: *IEEE Access* 12 (2023), pp. 2303–2312.

[33] Andrew Cassidy and Andreas G Andreou. «Dynamical digital silicon neurons». In: *2008 IEEE biomedical circuits and systems conference*. IEEE. 2008, pp. 289–292.

[34] Shiyu Yang et al. «An efficient fpga implementation of Izhikevich neuron model». In: *2020 International SoC Design Conference (ISOCC)*. IEEE. 2020, pp. 141–142.

[35] Hamid Soleimani, Arash Ahmadi, and Mohammad Bavandpour. «Biologically inspired spiking neurons: Piecewise linear models and digital implementation». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 59.12 (2012), pp. 2991–3004.

[36] Mohammad Ali Hedayatpour, Mohammad Azim Karami, and Jafar Shamsi. «Implementation of Izhikevich neuron based on stochastic computing using a novel inspired Omega-Flip stochastic number generator». In: *International Journal of Circuit Theory and Applications* 50.9 (2022), pp. 3104–3118.

[37] Louise H Crockett et al. *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014.

[38] Gary Weiss. *WISDM Smartphone and Smartwatch Activity and Biometrics Dataset*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HK59. 2019.

[39] Gary M Weiss, Kenichi Yoneda, and Thaier Hayajneh. «Smartphone and smartwatch-based biometrics using activities of daily living». In: *Ieee Access* 7 (2019), pp. 133190–133202.

[40] Eric R Kandel et al. *Principles of neural science*. Vol. 4. McGraw-hill New York, 2000.

[41] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

[42] Wulfram Gerstner et al. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[43] Wikimedia Commons contributors. *Action potential (illustration)*. `https://commons.wikimedia.org/wiki/File:Action_potential.svg`. Accessed 5 October 2025. Licensed under CC BY-SA 3.0. Original author: User:Domdomegg, based on original work by User:Chris 73. 2025.

[44] Alan L Hodgkin and Andrew F Huxley. «A quantitative description of membrane current and its application to conduction and excitation in nerve». In: *The Journal of physiology* 117.4 (1952), p. 500.

[45] Alan Lloyd Hodgkin. «The ionic basis of electrical activity in nerve and muscle». In: *Biological Reviews* 26.4 (1951), pp. 339–409.

[46] Rufin Van Rullen and Simon J Thorpe. «Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex». In: *Neural computation* 13.6 (2001), pp. 1255–1283.

[47] FE BORST Aand THEUNISSEN. «Information theory and neural coding». In: *Nature Neurosci* 2 (1999), pp. 947–957.

[48] Larry F Abbott. «Lapicque's introduction of the integrate-and-fire model neuron (1907)». In: *Brain research bulletin* 50.5-6 (1999), pp. 303–304.

[49] Richard FitzHugh. «Impulses and physiological states in theoretical models of nerve membrane». In: *Biophysical journal* 1.6 (1961), pp. 445–466.

[50] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. «An active pulse transmission line simulating nerve axon». In: *Proceedings of the IRE* 50.10 (2007), pp. 2061–2070.

[51] Catherine Morris and Harold Lecar. «Voltage oscillations in the barnacle giant muscle fiber». In: *Biophysical journal* 35.1 (1981), pp. 193–213.

[52] James L Hindmarsh and RM Rose. «A model of neuronal bursting using three coupled first order differential equations». In: *Proceedings of the Royal society of London. Series B. Biological sciences* 221.1222 (1984), pp. 87–102.

[53] Shaghayegh Gomar and Arash Ahmadi. «Digital multiplierless implementation of biological adaptive-exponential neuron model». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.4 (2013), pp. 1206–1219.

[54] Barry W Connors and Michael J Gutnick. «Intrinsic firing patterns of diverse neocortical neurons». In: *Trends in neurosciences* 13.3 (1990), pp. 99–104.

[55] Jay R Gibson, Michael Beierlein, and Barry W Connors. «Two networks of electrically coupled inhibitory neurons in neocortex». In: *Nature* 402.6757 (1999), pp. 75–79.

[56] Charles M Gray and David A McCormick. «Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex». In: *Science* 274.5284 (1996), pp. 109–113.

[57] John E Lisman. «Bursts as a unit of neural information: making unreliable synapses reliable». In: *Trends in neurosciences* 20.1 (1997), pp. 38–43.

[58] Eugene M Izhikevich et al. «Bursts as a unit of neural information: selective communication via resonance». In: *Trends in neurosciences* 26.3 (2003), pp. 161–167.

[59] Alan L Hodgkin. «The local electric changes associated with repetitive action in a non-medullated axon». In: *The Journal of physiology* 107.2 (1948), p. 165.

[60] John Rinzel and G Bard Ermentrout. «Analysis of neural excitability and oscillations». In: *Methods in neuronal modeling* 2 (1998), pp. 251–292.

[61] Eugene M Izhikevich. «Resonate-and-fire neurons». In: *Neural networks* 14.6-7 (2001), pp. 883–894.

[62] Uwe Meyer-Baese. *Digital signal processing with field programmable gate arrays*. Springer, 2007.

[63] Pong P Chu. *FPGA prototyping by Verilog examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[64] Steve Kilts. *Advanced FPGA design: architecture, implementation, and optimization*. John Wiley & Sons, 2007.

[65] AMD (Advanced Micro Devices). *Zynq-7000 SoC Technical Reference Manual*. Doc. UG585, Rev. 1.14 (as of June 30, 2023). [Online]. Available: `https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM` [Accessed: October 5, 2025]. AMD. San Jose, CA, 2023.

[66] AMD (Advanced Micro Devices). *Zynq-7000 SoC Data Sheet: Overview*. Doc. UG190, Rev. 1.11.1 (as of July 02, 2018). [Online]. Available: `https://https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview` [Accessed: October 5, 2025]. AMD. San Jose, CA, 2018.

[67] Advanced Micro Devices, Inc. *PYNQ*. Software. Version 3.1.2. BSD-3-Clause license. Released on 2025-09-24. https://pynq.io. Repository: https://github.com/Xilinx/PYNQ. Sept. 2025.

[68] Vineeth C Johnson et al. «Fpga-based hardware acceleration using pynq-z2». In: *2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*. IEEE. 2023, pp. 1–4.

[69] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[70] Kawthar Dellel et al. «Differentiable Selection of Bit-Width and Numeric Format for FPGA-Efficient Deep Networks». In: *Electronics* 14.18 (2025), p. 3715.

[71] Nonel Thirer et al. «Improvement of pipelines implementations in FPGA designs». In: *Infrared and Photoelectronic Imagers and Detector Devices II*. Vol. 6294. SPIE. 2006, pp. 229–235.

[72] Bruce Jacob, Spencer W. Ng, and David T. Wang. «The Cache Layer». In: *Memory Systems*. Ed. by Bruce Jacob, Spencer W. Ng, and David T. Wang. Morgan Kaufmann, 2008, pp. 731–745. ISBN: 9780123797513. DOI: 10.1016/B978-012379751-3.50024-2. URL: https://www.sciencedirect.com/science/article/pii/B9780123797513500242.

[73] Lizhe Tan and Jean Jiang. «Chapter 14 - Hardware and Software for Digital Signal Processors». In: *Digital Signal Processing (Third Edition)*. Ed. by Lizhe Tan and Jean Jiang. Academic Press, 2019, pp. 727–784. ISBN: 9780128150719. DOI: 10.1016/B978-0-12-815071-9.00014-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128150719000142.

[74] AMD Xilinx. *Vivado Design Suite User Guide: Synthesis (UG901)*. Version 2023.1. Accessed: October 11, 2025. AMD. 2023. URL: https://docs.amd.com/r/en-US/ug901-vivado-synthesis.