µCaspian

Technical Documentation

Parker Mitchell

Caspian Platform

- A Platform for developing Spiking Neural Networks which may be deployed onto neuromorphic architectures
- Includes:
  - Fixed-increment discrete event simulator (C++)
  - Network creation and manipulation API (C++ / Python)
  - Unified simulation & hardware API (C++ / Python)
- Currently building out novel neuromorphic hardware architectures
- µCaspian is the first of those architectures

µCaspian v1

- 256 neurons, 4096 synapses (on a Lattice iCE40 up5k FPGA)
- Integrate and Fire Neuron model
  - 8-bit thresholds, 16-bit charge state, 4-bit axonal delay (0-15 cycles)
- Sparse directed graph connectivity embedding
  - Outgoing synapses are stored as a contiguous range for each axon/neuron
  - Each synapse stores the id of its target neuron as well as an 8-bit signed weight value
- Deterministic, event-driven execution model
  - Exact match with software simulation
  - Each time step requires a variable execution time dependent on network activity
- Integrated with software ecosystem
  - Caspian simulator, EONS training algorithm, TENNLab framework, etc.
  - Hardware is usable through a C++ or Python API

The µCaspian Architecture

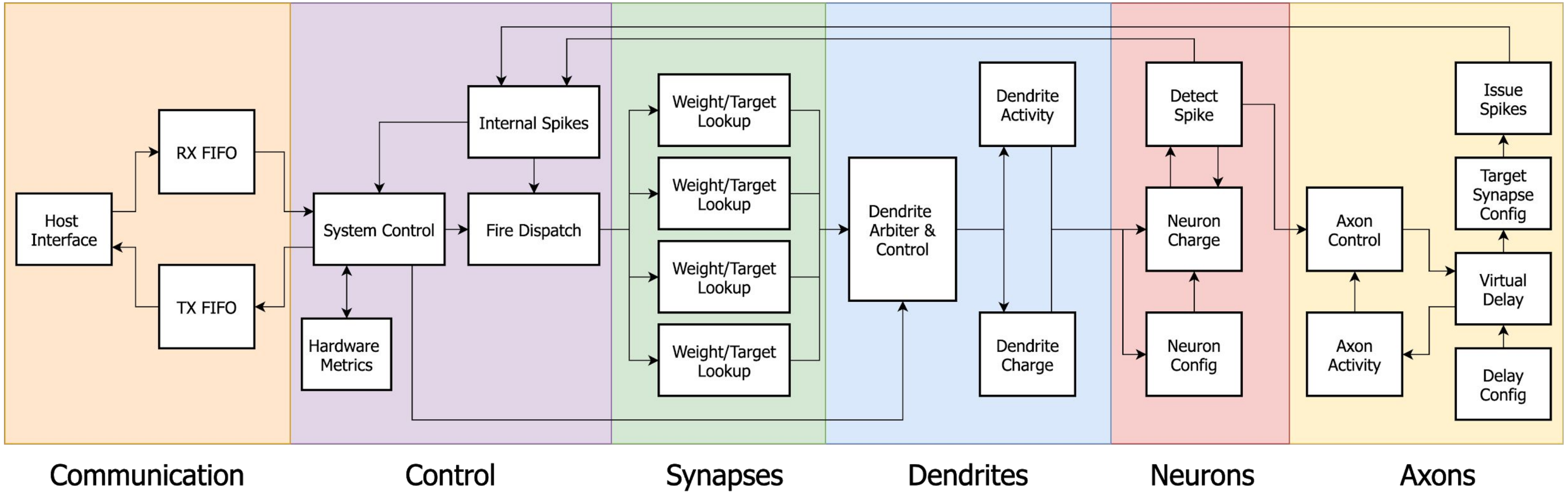
Architecture Concepts

“Pipeline of pipelines” - The architecture uses a dataflow pipeline of modules in which each module may be internally pipelined. Inter-module communication uses a standard 2-bit handshake flow control protocol (ready/valid).

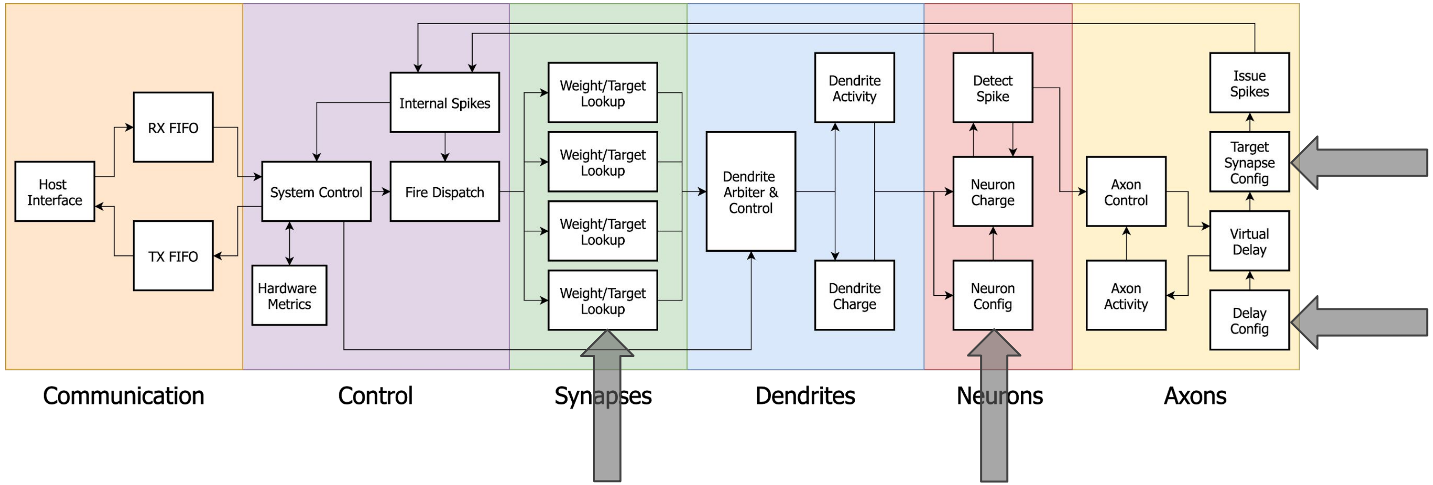
Event-driven computation - Where possible, the architecture aims to only process values as required to maintain correctness.

Localized memory - Each module is responsible for storing its own configuration and state using localized SRAM memories. No global or off-chip memory is used.

Architecture Overview



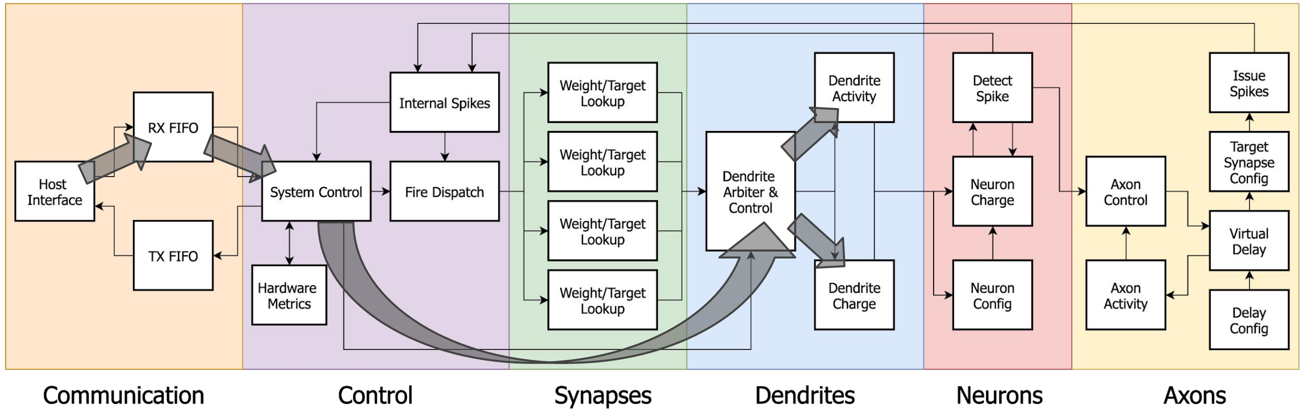
Architecture Overview



Step 1: The host sends commands to configure a network.

The arrows indicate where the network configuration is stored.

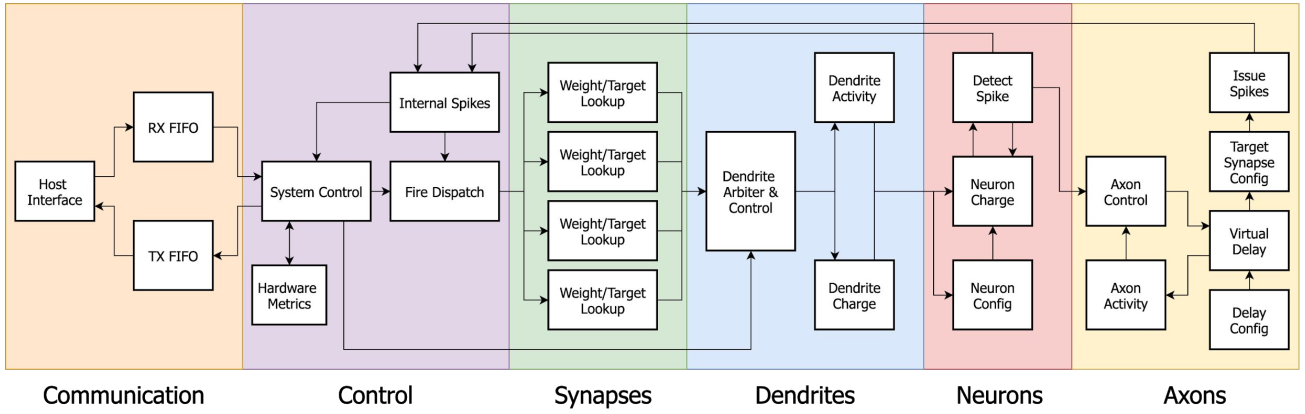
Architecture Overview



Step 2: The host sends inputs for the network which are accumulated into the dendrite units.

The arrows indicate the flow of data from host to dendrite

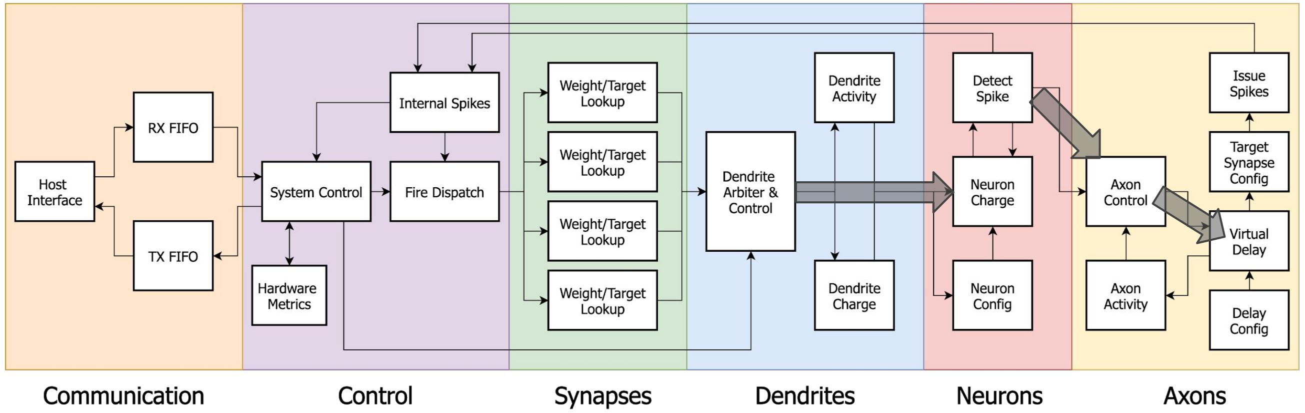
Architecture Overview



Step 3: The host sends a step forward command to advance time by *t* steps.

The host may send additional inputs in between step commands.

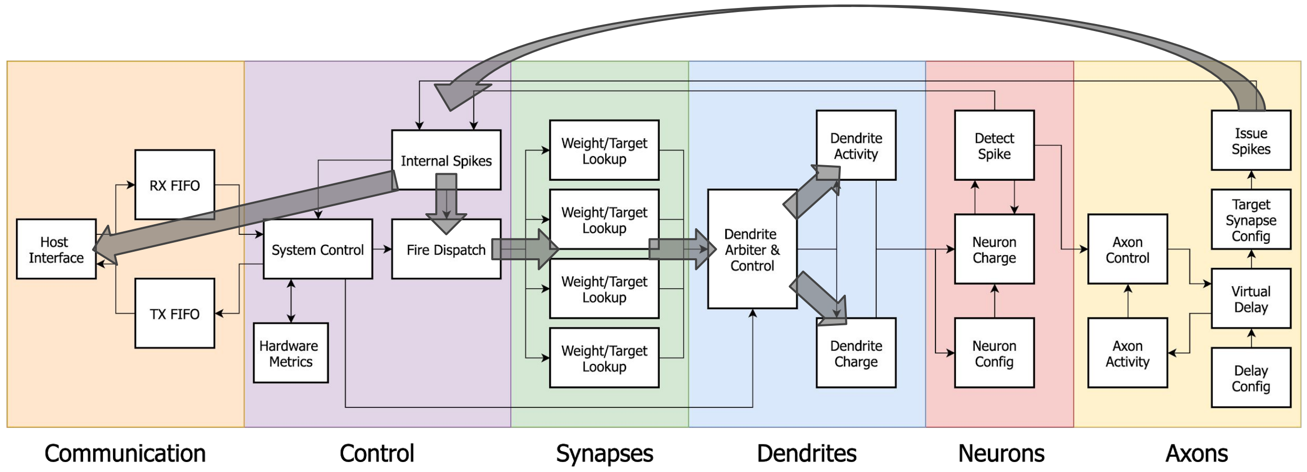
Architecture Overview



<u>Step 4a: The first phase of network execution</u>

Any stored charge in the dendrites is flushed to the neurons.
The neurons update state and determine if it should spike.
Neuron spikes are sent the axon unit.

Architecture Overview



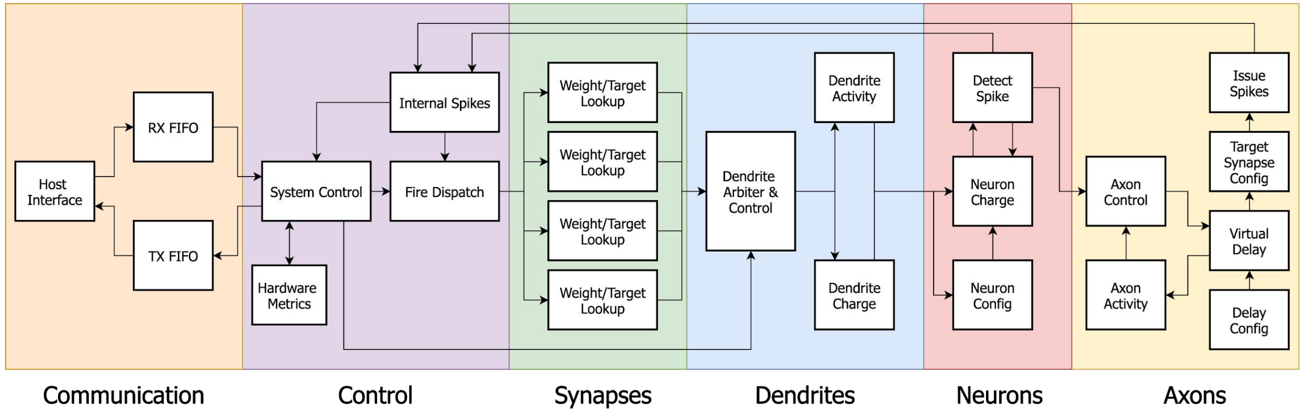
<u>Step 4b: The second phase of network execution</u>

The axon unit flushes any delayed spikes for the new timestep.

For each axon spike, a set of synapses will look up value/target pairs.

The dendrite unit will accumulate the value for the target neuron address.

Architecture Overview



Steps 2-4 repeat until the network has received all inputs and processed all desired time steps.

Note: Steps 4a and 4b overlap in execution during each timestep.

Architecture Overview

Steps to running a network:

- Step 1: Configure the neurons and synapses
- Step 2: Host sends input to the network
- Step 3: Advance time
- Step 4: Process all activity at time t
    - Flush charge from dendrite buffers to the neuron
    - Update neuron state and spike if charge > threshold
    - Axon accepts spikes from the neuron and tracks delay using virtual shift register in SRAM
    - If remaining delay is zero, transmit range of synapse to spike to the spike dispatch
    - Dispatch spikes to synapse units which will look up the weight & target neuron
    - Accumulate new charge from synapses into the dendrite buffers
- Repeat steps 2-4 as necessary until network is at the target time

A note about activity tracking

- Both dendrites and axons have activity tracking to reduce unnecessary work
- Each unique dendrite/axon is grouped into sets of 16
- A 16-bit register (16 bits * 16 per group) represents activity of each group
- Only groups with a set bit are evaluated for a timestep
- This approach is a balance of low area overhead and extra state processing
- Neuron activity is triggered by dendrite activity so tracking is not needed
- Synapse activity is triggered by axon activity so tracking is not needed

Communication with Host

- Dual 8-bit AXI4-Stream interfaces to the µCaspian core
- 8x512 FIFO for both send and receive channels
- Variable length packet format for minimizing required bandwidth
- Command packets are coupled to the core controller’s state machine
- Output spikes & command responses are sent through a separate state machine

Variable Length Packets

Command Packets

- Add Charge (2 bytes)
- Step Time (2 bytes)
- Get Metric (2 bytes)
- Clear Activity (1 byte)
- Clear Configuration (1 byte)
- Configure Neuron (7 bytes)
- Configure Synapse (5 bytes)
- Configure Multiple Synapses (5 bytes + 2 bytes per synapse)

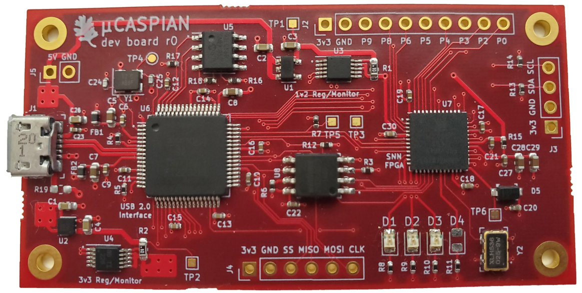
Response Packets

- Configuration Acknowledgement (1 byte)
- Clear Acknowledgement (1 byte)
- Metric Response (3 bytes)
- Time Update (5 bytes)
- Output Spike (2 bytes)

µCaspian Development Board

The Development Board

- 2.75” by 1.4” custom PCB design for Caspian development
- Includes Lattice iCE40 UltraPlus FPGA and FT2232H USB 2.0 HS bridge
- Can be reconfigured during runtime with new FPGA bitstreams
- On-board power monitoring for 3.3V (I/O) and 1.2V (FPGA Logic)
- Exposed I/O for configuration, application data, and power monitoring
- Could be built at volume for less than $35



Target FPGA: Lattice ice40 up5k

- 5280 Logic Cells consisting of a 4-input LUT and a DFF
- 30 4Kbit pseudo dual port SRAMs (16x256)
- 4 256Kbit single port SRAMs (16x16384)
- 8 MAC units (32 bit accumulator or 16 bit multiply-accumulate)
- Free closed source toolchain: Lattice Radiant
- Open source toolchain: yosys + nextpnr + icestorm

Target Communications: FTDI 2232H

- USB 2.0 High Speed Bridge Chip
- Two independent communication channels
- Channel A - FT245 Asynchronous FIFO for host to FPGA communication
- Channel B - SPI for configuring the bitstream on the ice40 FPGA
- Open source software library: libftdi + libusb

Annotated Development Board