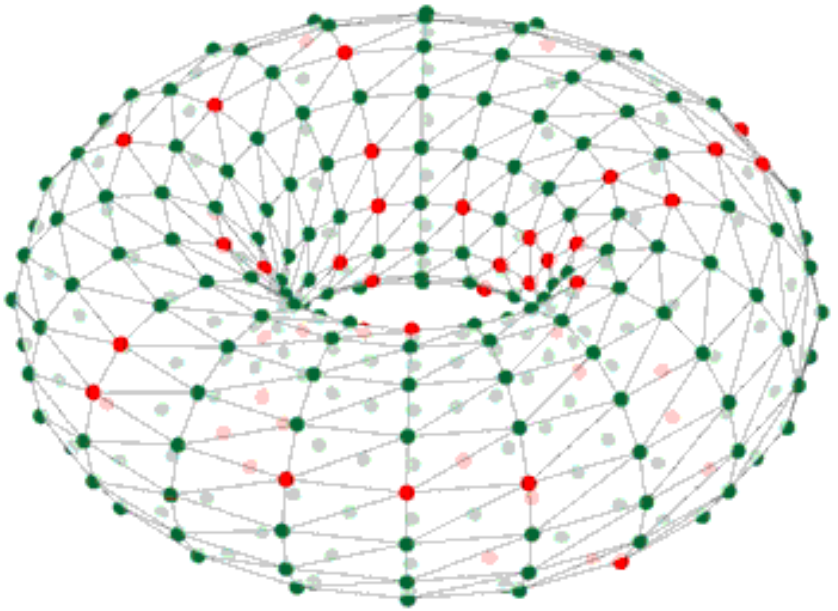
MANCHESTER 1824

Event-Driven Neural Simulation



Alex Rast

SpiNNaker Workshop, September 2015

erc
European Research Council
Established by the European Commission

Human Brain Project

EPSRC

SpiNNaker

MANCHESTER 1824

Session Outline

1. What happens When a Simulation Runs?
- a. Design Assumptions
- b. Tool Chain Instantiation
- c. On-System Startup
- d. Real-Time Execution

2. Responding to Events
- a. Packet Received
- b. User Event
- c. DMA Done
- d. Timer Tick

3. Time and Events
- a. Event Priorities
- b. What is "Real-Time"?
- c. Adjusting Time Resolution
- d. Using Retarded Time

4. So how do You Make a Working Model?
- a. Handling Events
- b. Memory Utilisation
- c. Debugging and (Lack of) Visibility
- d. What SpiNNaker Can Do

What Happens When A Simulation Runs?

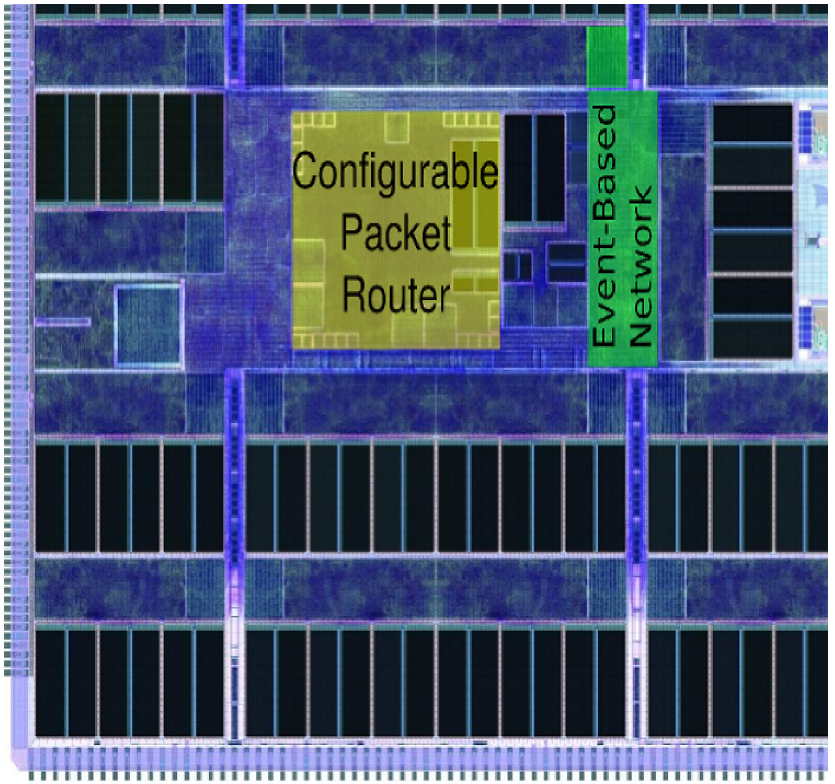


Fig1: SpiNNaker Chip

Design Assumptions

1. Memory: Cores only need access their own *limited* local memory. No global memory.

2. Communications: Via AER spikes only, multicast source-routed.

3. Event Rates: Real-time at "biologically meaningful" resolution. Hardware is much faster than any simulation time scale.

4. Model Dynamic Complexity: Very simple models are good enough. Most biological minutiæ don't matter.

5. Time Model: Execution is event-driven; time "models itself" (is implicit).

What Happens When A Simulation Runs?

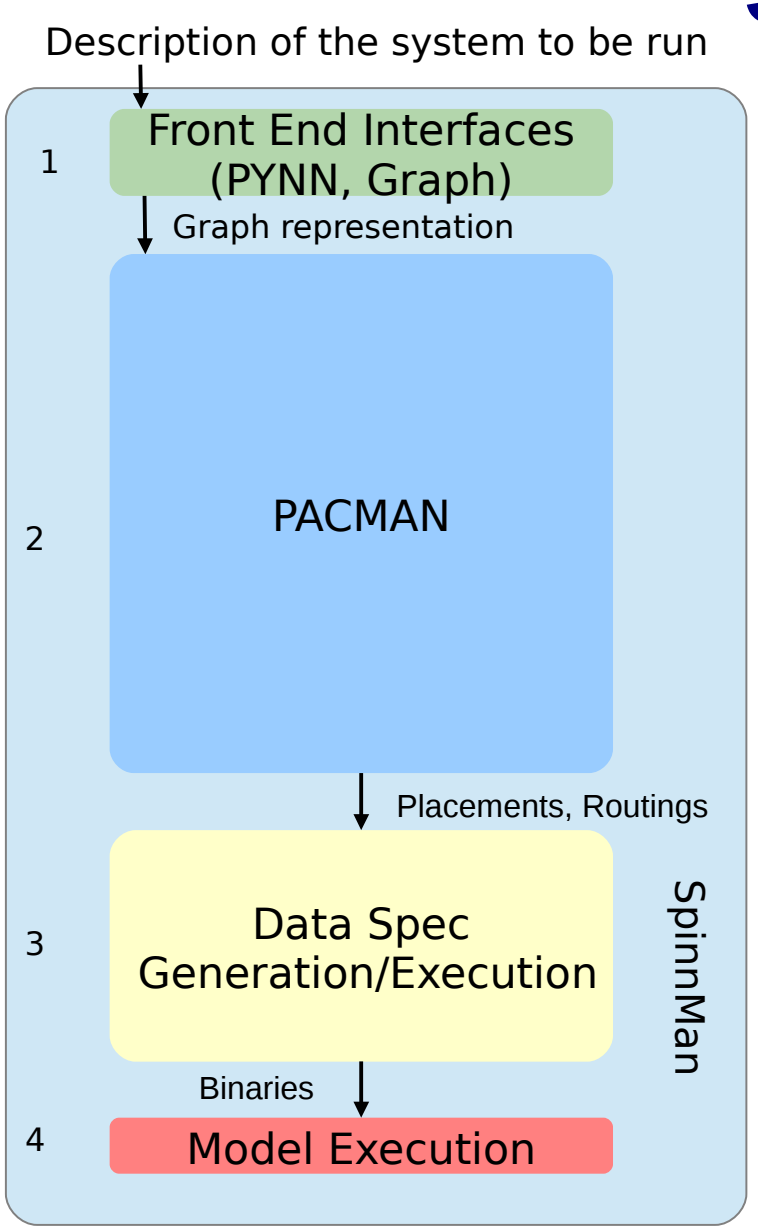


Fig2: The Tool Chain stack

Stage 1: Instantiation Through Tool Chain

1. Script binds to SpiNNaker front-end

2. Front-end converts the script into a set of nodes and edges

3. PACMAN partitions the nodes to cores and edges to routing table info and synaptic matrices in SDRAM

4. DSG transforms the partitioned graph into a set of instantiation specifications

5. DSE unpacks the specification into an on-chip executable

What Happens When A Simulation Runs?

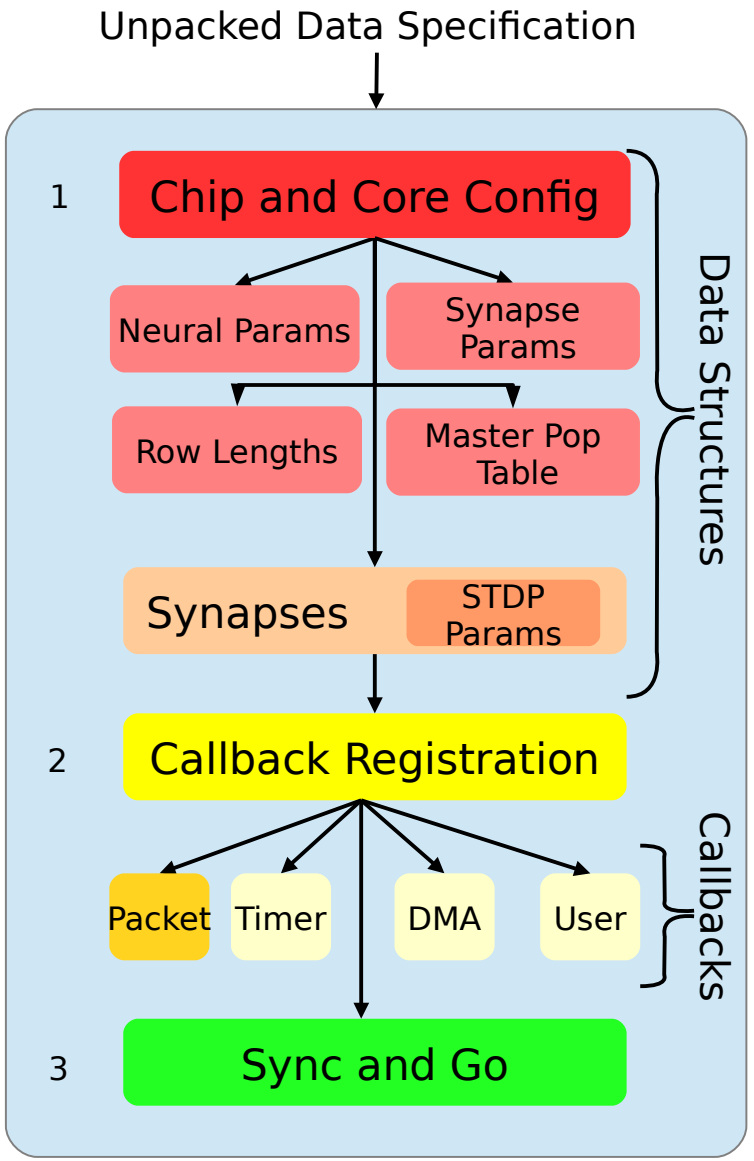


Fig3: Application Startup

Stage 2: On-System Startup

1. Generation of Core Data Structures:

- Neural Parameter Structures
- Synaptic Parameter Structures
- Synaptic Row Lengths
- Master Population Table
- Synaptic Connection Matrix
- STDP Parameters (if any)

2. Registering Callbacks

- Spike Received (buffer and ask for a DMA read)
- Timer Interrupt (start processing the next state update)
- DMA Complete (dump inputs into ring buffers and update SDTP)
- User Event (retrieve synapses from SDRAM)

3. Wait for Synchronisation, then Go!

What Happens When A Simulation Runs?

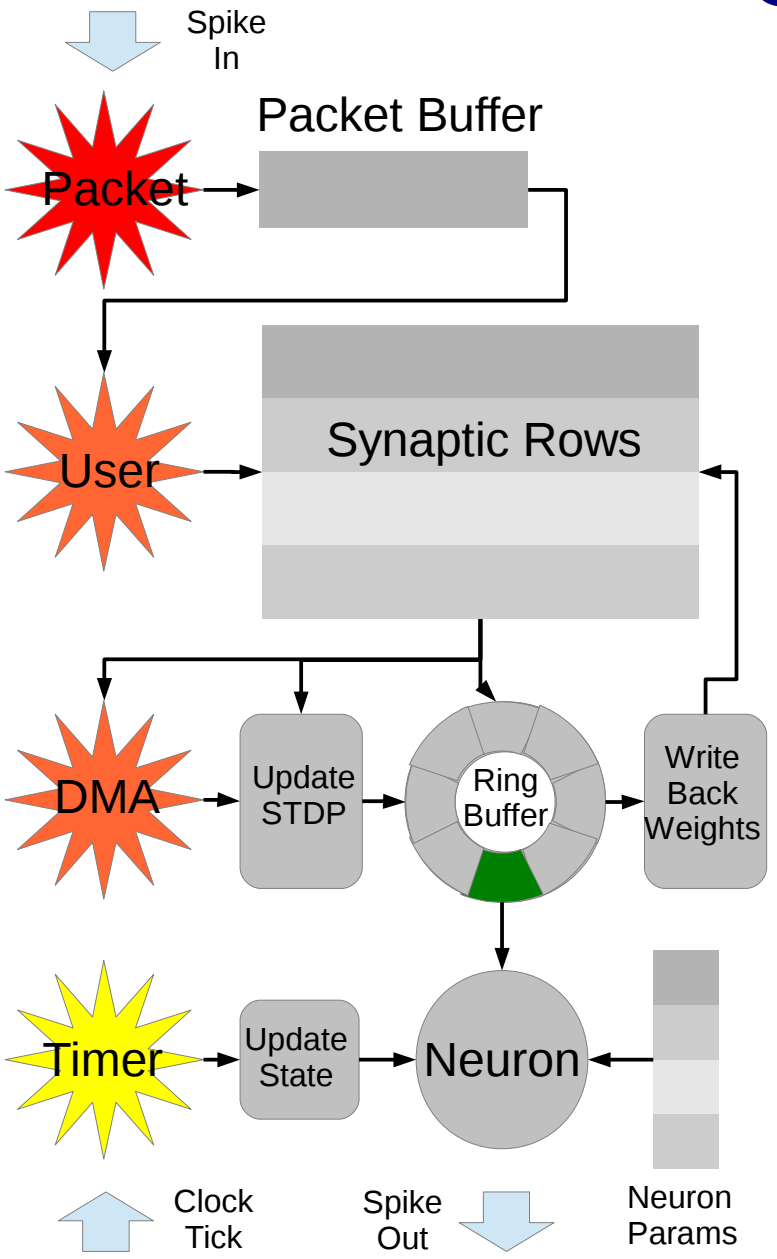


Fig4: Execution Model

Stage 3: Real-Time Execution

1. Packet Received [High Priority]:

2. User Event [Normal] (Request DMA)

3. DMA Completed [Normal]

4. Timer Tick [Low Priority]

Responding to Events

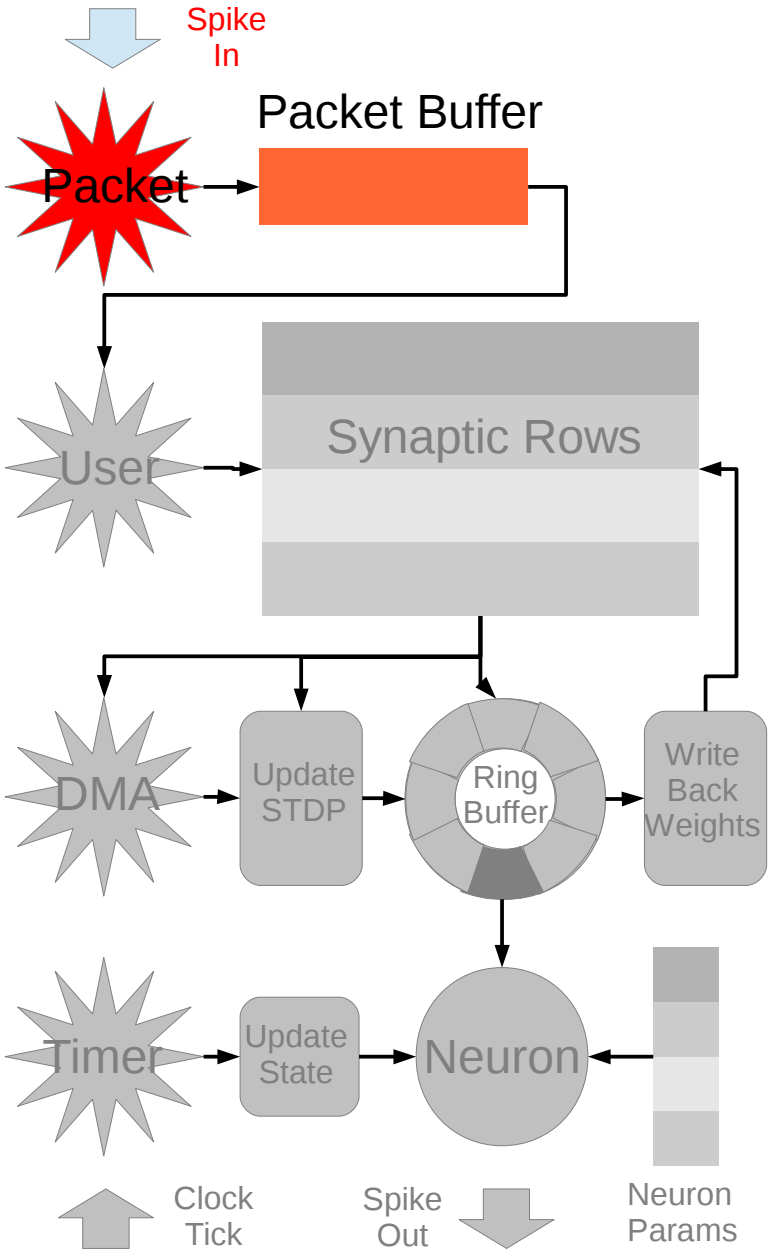


Fig4a: Execution Model

Packet Received

a) Dump packet into a buffer

b) Ask for a User event, if necessary

Responding to Events

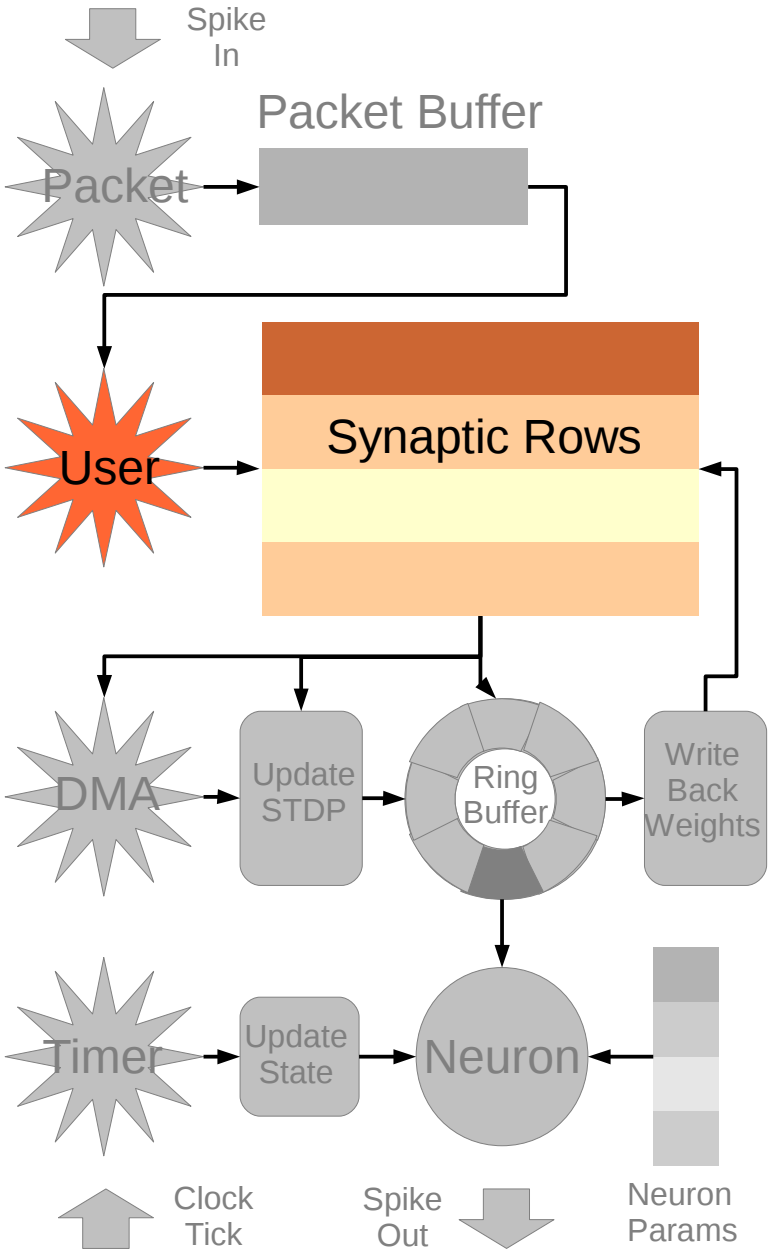


Fig4b: Execution Model

User Event

a) Retrieve next packet from buffer

b) Look up row address (in Master Population Table)

c) Set up the DMA in the controller

d) Start the next DMA transfer

e) Swap DMA buffers (for next transfer)

Responding to Events

DMA Completed

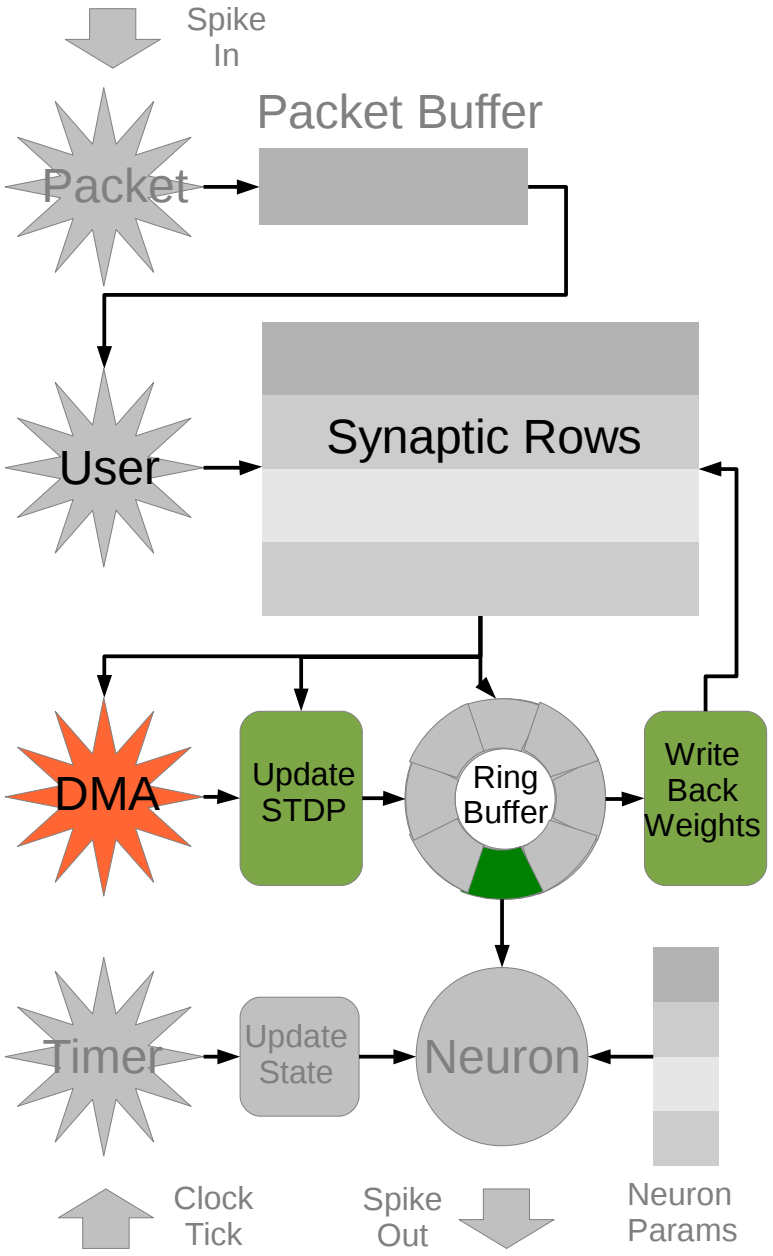


Fig4c: Execution Model

For each target in the row:

a) Start next DMA (everything under User Event)

b) Inject the current weight into the ring buffer at the delay indicated by the row's delay field

c) Update STDP, if enabled

d) Write back weight values to SDRAM via DMA

Responding to Events

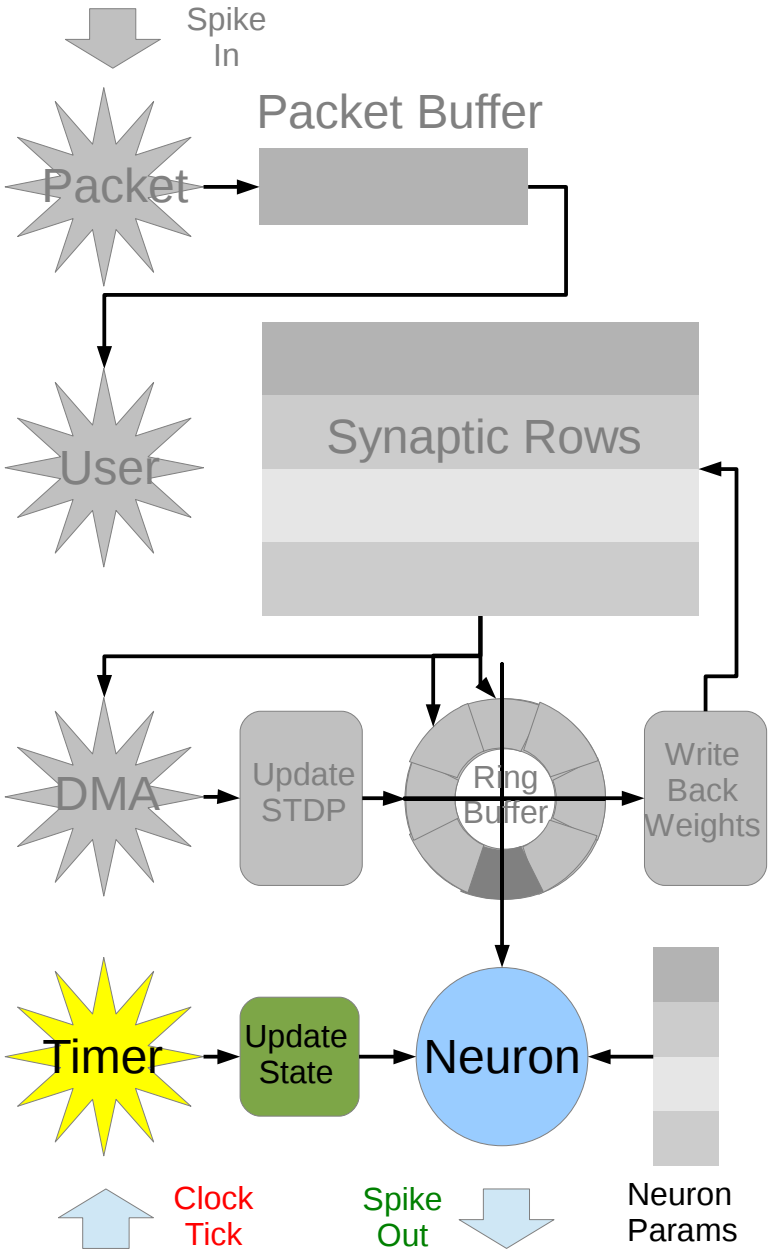


Fig4d: Execution Model

Timer Tick

For each neuron on the core:

a) Decay the ring-buffer entries

b) Inject the current ring-buffer entry onto the neuron

c) Perform the neural state update

d) If neuron has reached threshold, spike.

e) If STDP is enabled, update for any post-synaptic spikes.

Time And Events

Event Priorities

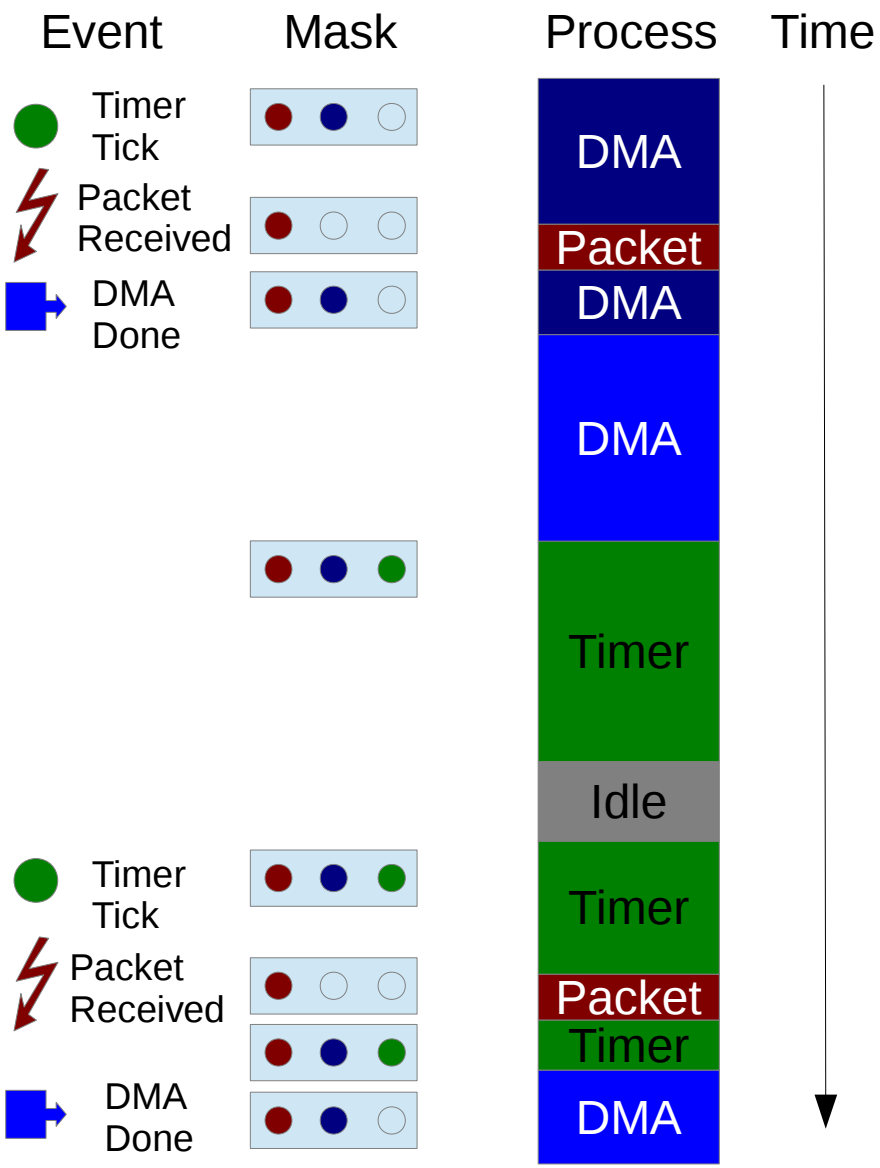


Fig5: Event Priority and Interrupt Servicing

Why? Events can overlap. Some events (packet received!) are critical. Priorities manage which events are serviced when.

Priority -1: Override priority. Can only assign to one event. MUST be serviced immediately. Assigned to packet received.

Priority ≥ 0: "Normal" priority. Maskable events with various priority levels. Assigned to all other events: DMA done (0), User (0), Timer (2)

How? Set up in API when callback registered for event using

```
spin1_callback_on(event, callback, priority)
```

Time And Events

What is "Real-Time"?

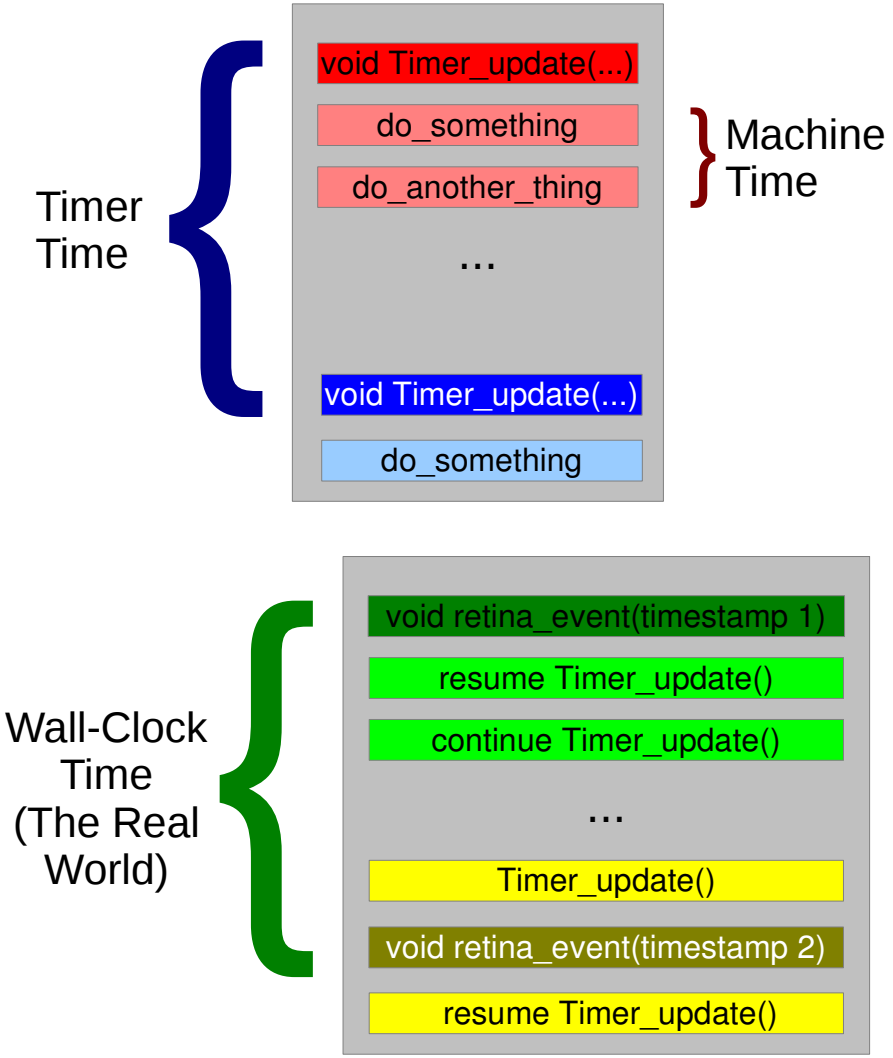


Fig6: Different Time Domains

Machine Time: Core clock time. Unique to each core; NOT system-global. Intervals much smaller than Timer or "real-time".

Timer Time: Time between Timer ticks. Typically 1 ms; can be changed with an API call. Speedable-up or slowable relative to "real time".

Wall-Clock Time: External reference time. May be used by external devices (e.g. robots). Real-world "real-time".

Time And Events

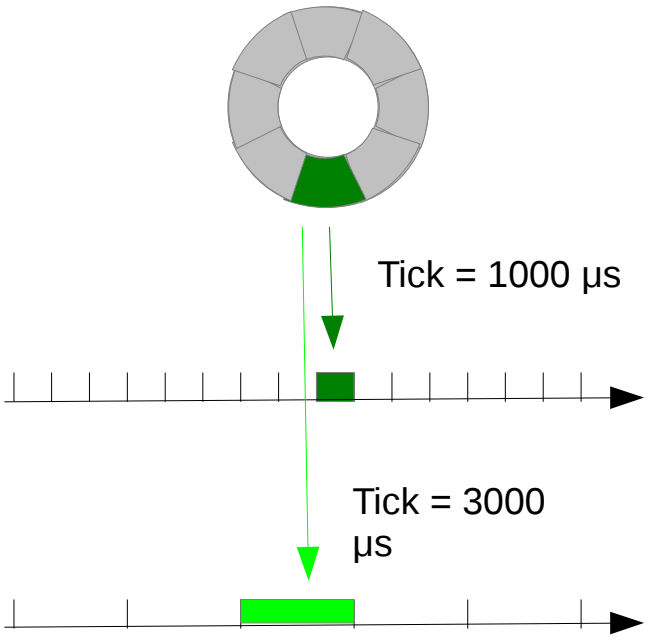


Fig7: Different Time Resolutions

Adjusting Time Resolution

Units: Timer resolution is currently in microsecond units

API: spin1_set_timer_tick() sets the time resolution (in µs).

Tool Chain: machineTimeStep in [Machine] section of spynnaker.cfg sets the time resolution (in µs).

Time And Events

Using Retarded Time

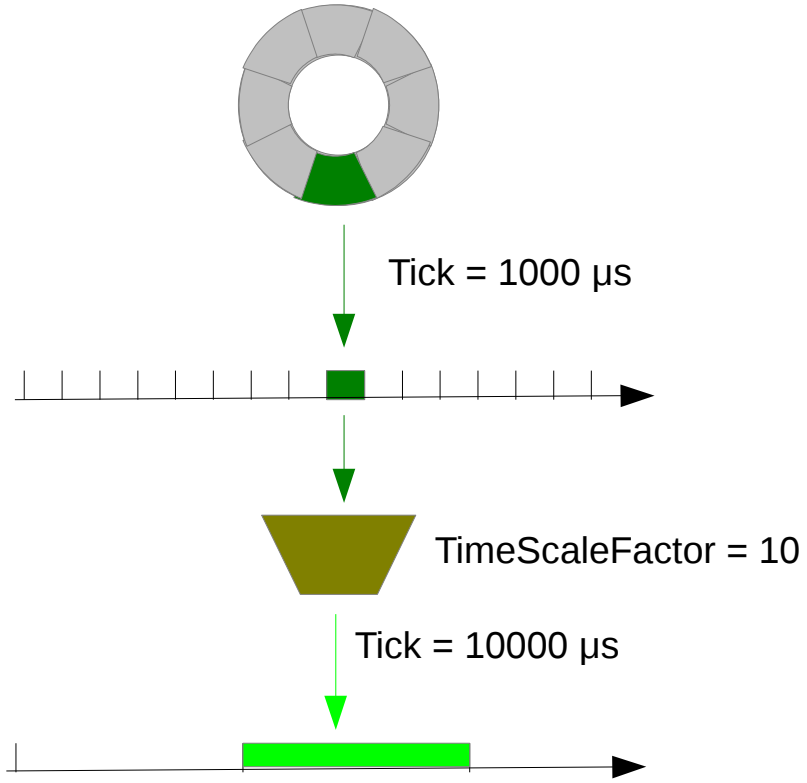


Fig8: Retarded Time

Timescale Factor (F): Scales down Timer time so that *n* timer ticks with machine time step *m* µs corresponds to *n***m**F real-world µs

Toolchain Only: Not reflected in running code on the machine; this applies the time scaling through the toolchain itself.

In spynnaker.cfg: set with timeScaleFactor in the [Machine] section.

So How do You Make a Working Model?

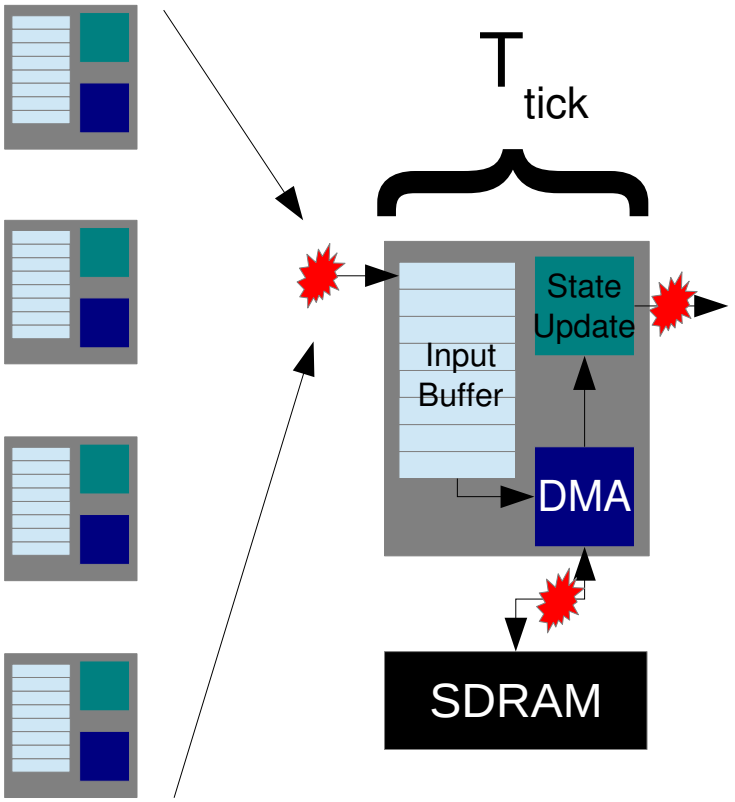


Fig9: Sources of Congestion

Handling Events

Unload the Fabric ASAP: Make the FIQ for packet-received efficient to prevent packet drops.

Issue Output Events As They Happen: Buffering output spikes only leads to bursty traffic and congestion.

Keep Time Resolution Coarse: Smaller timesteps drastically narrow the event receipt window.

DMA in Large Blocks: The controller is optimised for ~2k block size; small blocks may require a greater number of more inefficient transfers (DMA interrupts).

Slow Time, If Necessary: Slowing the time from real-time gives more slack for events to complete.

So How do You Make a Working Model?

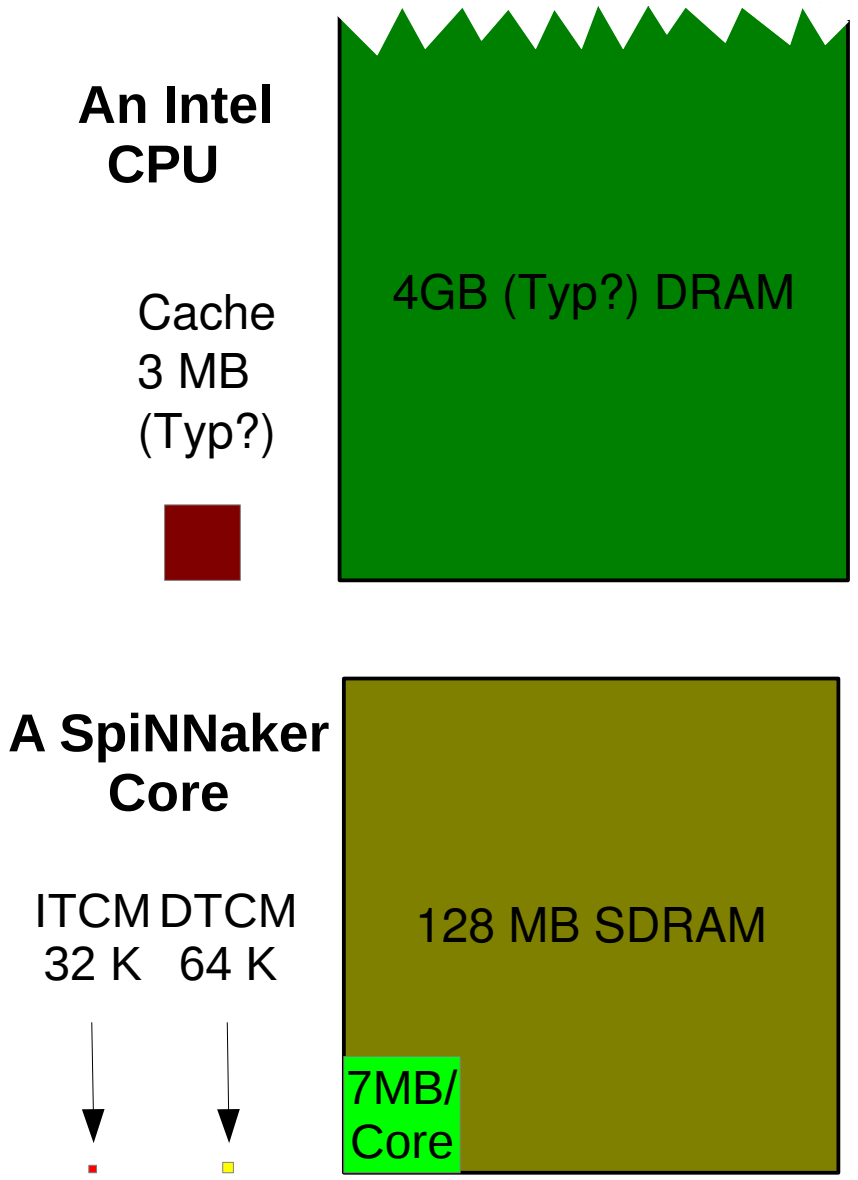


Fig10: What Cores See

Memory Limitations

Per Core:

64KB DTCM: All the neural parameters plus synaptic ring buffers PLUS temporary variables, lookup tables, etc. must fit in this space.

32KB ITCM: All the running code for a given model must fit in this space. This includes the SARK RTOS and the SpiNNaker API.

Per Chip:

128MB SDRAM: Partitioned amongst working cores. Synaptic weights, delays, and timestamps plus some system variables must fit here.

So How do You Make a Working Model?

Debugging and (Lack of) Visibility

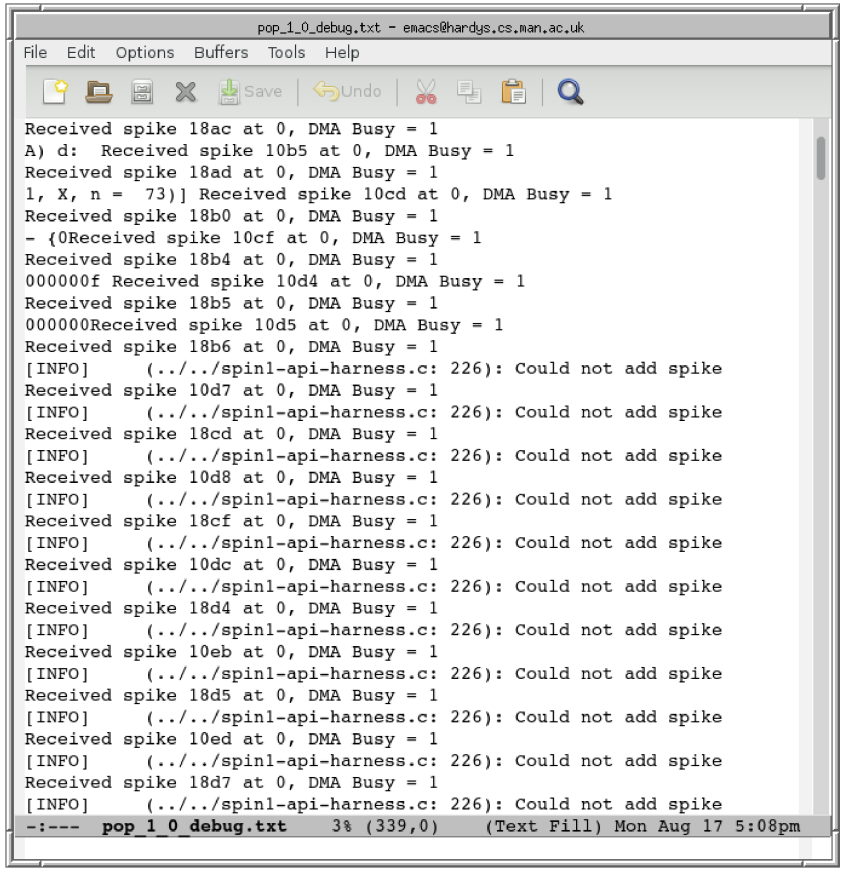


Fig11: Debug Output

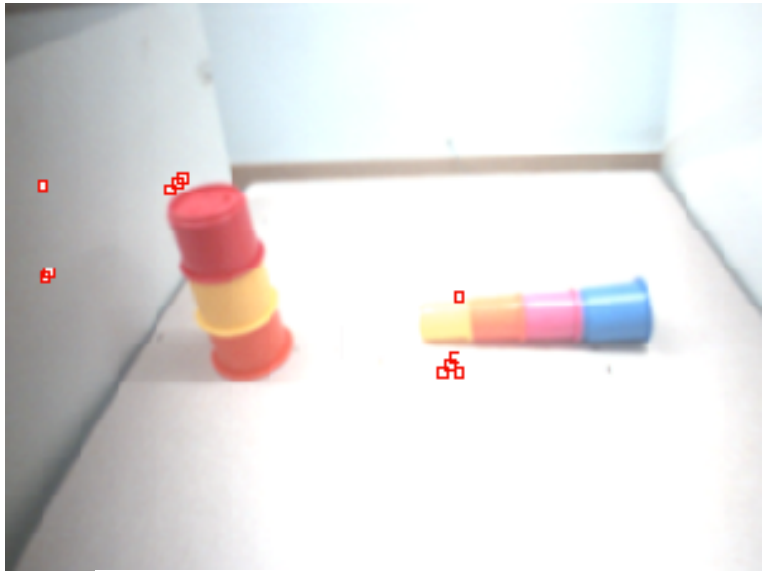
There is No Global State: Cores run in different time domains. It is not only infeasible to set a "global breakpoint", it is meaningless. *Inspect* one core at a time.

Interchip Timing Matters: Single-chip or -core results may not reveal everything. *Test* across multichip simulations to expose asynchronous bugs.

Debug Statements Alter Timing: Debug statements change the time events arrive and can cause breaking code to run. Use them, but be aware of the risks.

Events Interact: Because of different event priorities, some events may interrupt others in progress.

So How do You Make a Working Model?



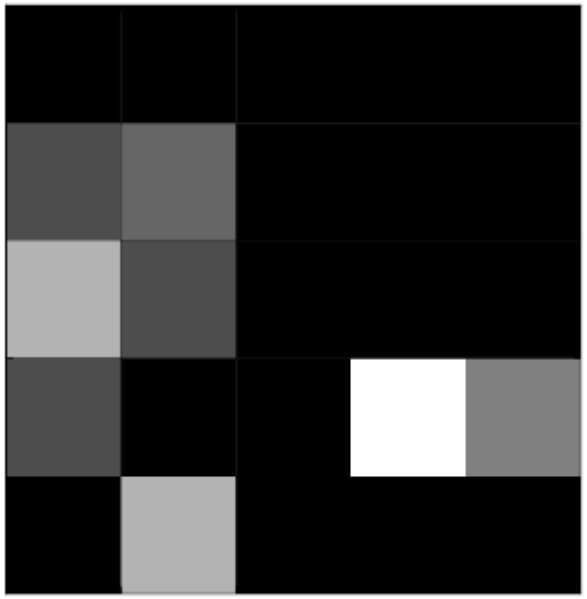


Fig12: A Model With Real-World Input

What SpiNNaker Can Do

Experimenting With Network Parameters: Small simulations run fast enough that several runs may not just tune parameters but reveal patterns that show what parameters mean.

Scaling Networks: Large-scale networks can be run that simply would take too long to simulate even on a substantial cluster.

Exploring Network Design: Beyond parameter scaling, it is possible to alter network structure radically or even underlying neural models and still run in real-time.

Real-World Networks: By integrating with external hardware or robots, it is possible to explore real-world behaviour of large-scale networks in live situations.