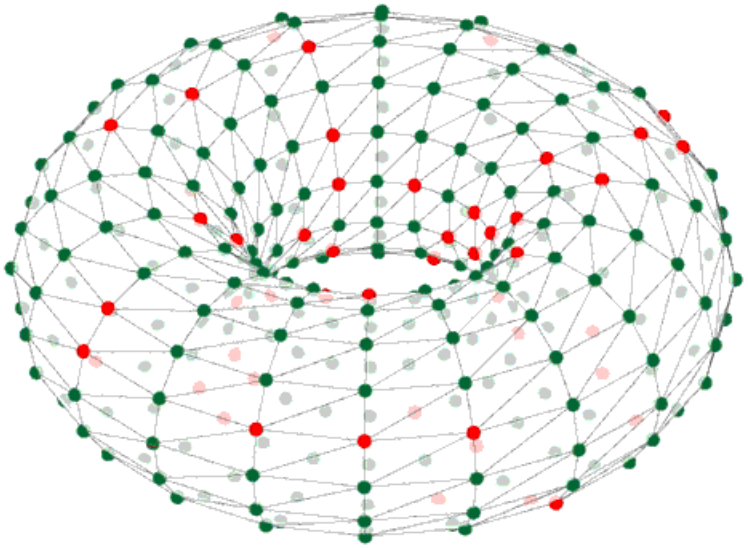
External Devices



Alex Rast, Sergio Davies, Alan Stokes

SpiNNaker Workshop
September 2015

European Research Council
Established by the European Commission

Human Brain Project

EPSRC

SpiNNaker

MANCHESTER 1824

Outline

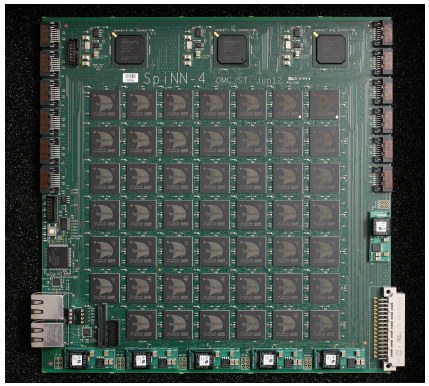
Ethernet Protocol
- Protocol Format
- The EIEIO Implementation

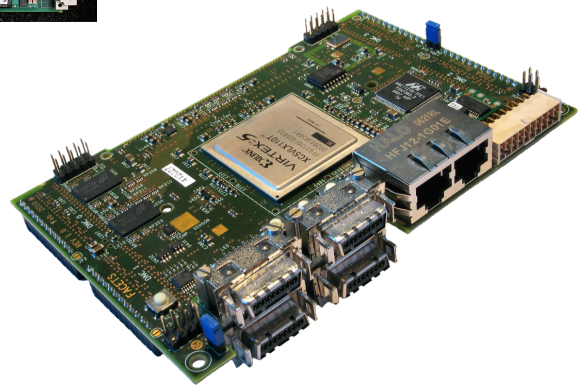
External Devices - Ethernet
- Live Output
- Live Input
- Prefixes and Keys
- The Database Interface

Worked Example: spike_io.py
- Setup
- Populations and Projections
- Setting up a Host Device
- Getting Input/Output

External Devices - SpiNNaker Link
- Hardware Connection
- Toolchain Modifications
- PyNN Instantiation

Connecting to External Devices





1: Spike Injection From a Synthetic Source Useful for very large spike lists that won't fit on-chip.

2: Live Spike Output for Visualisation, Etc. Useful for interactive display of results.

3: Spikes From an External AER Device Useful for Hardware With UDP Support

4: Spikes To an External AER Device Useful for SpiNNaker-based preprocessing

5: Bidirectional Communication Useful for ad-hoc real-time interactive systems

Fig. 1: Real-Time Systems

Available Interfaces

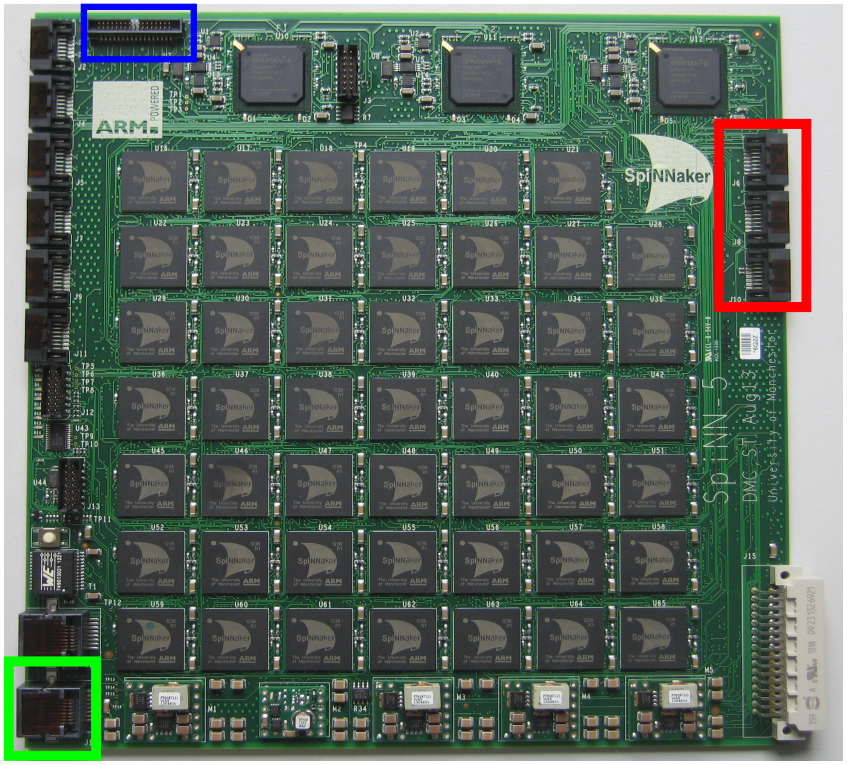


Fig. 2: SpiNNaker Interfaces

1: Ethernet Relatively slow-speed connections using a UDP protocol. Relatively "plug-and-play".

2: SpiNNaker Link Direct native connection to the SpiNNaker fabric. Usually uses an intervening FPGA. Some support available in tools but expect some low-level work.

3: "SpiNNLink" Not a SATA-compliant interface but uses SATA connectors and cabling. High-speed. "Roll-your-own"!

Ethernet AER Protocol

Packet Features

Header	Command Information
15 0	...

Header	Key	...	Key
15 0	31/15 0	...	31/15 0

Header	Key	Payload	...	Key	Payload
15 0	31/15 0	31/15 0	...	31/15 0	31/15 0

Header	Prefix	Key	...	Key
15 0	15 0	31/15 0	...	31/15 0

Header	Prefix	Key	Payload	...	Key	Payload
15 0	15 0	31/15 0	31/15 0	...	31/15 0	31/15 0

Header	Fixed Payload	Key	...	Key
15 0	31/15 0	31/15 0	...	31/15 0

Header	Payload Base	Key	Payload	...	Key	Payload
15 0	31/15 0	31/15 0	31/15 0	...	31/15 0	31/15 0

Header	Prefix	Fixed Payload	Key	...	Key
15 0	15 0	31/15 0	31/15 0	...	31/15 0

Header	Prefix	Payload Base	Key	Payload	...	Key	Payload
15 0	15 0	31/15 0	31/15 0	31/15 0	...	31/15 0	31/15 0

Fig. 3: AER Packet Formats

Multiple Spikes/Packet:
Up to 256 Spikes (64 on SpiNNaker)

Optional Payloads:
Each spike may include a second word as a payload

Embeddable Commands:
Device-specific commands can be put in the stream

Selectable Word Width:
Addresses and payloads can be 16- or 32-bit

Fire and Forget:
No handshaking support.

AER Protocol Headers

0	0	D	T	W	Y	Tag		Count		Data	
15	14	13	12	11	10	9	8	7	0	15	0

P	0	D	T	W	Y	Tag		Count		Key Prefix (if P) or Payload Prefix (if D)	
15	14	13	12	11	10	9	8	7	0	15	0

1	F	1	T	W	Y	Tag		Count		Key Prefix		Payload Prefix	
15	14	13	12	11	10	9	8	7	0	15	0	31/15	0

Fig. 4: AER Data Packet Headers

0	1	Command		(Device-specific)
15	14	13	0	...

Fig. 5: AER Command Packet Header

Spike Count:
Bits 7-0 of the header

Word Width:
Bit 11 (32-bit = 1)

Data Payloads:
Bit 10 (1 = Enable)

Command Packets:
Bits 15-14 (Command = 01)

Key Prefixes:
ORs a fixed key offset. Bit 15 enables, Bit 14 sets which halfword to OR with. Prefix follows header.

Payload Prefixes:
Enables a fixed payload OR pattern. Bit 13 enables. Prefix follows any key prefix.

Timestamps:
Bit 12 (1 = Payloads are timestamps)

EIEIO: An AER Compliant Implementation

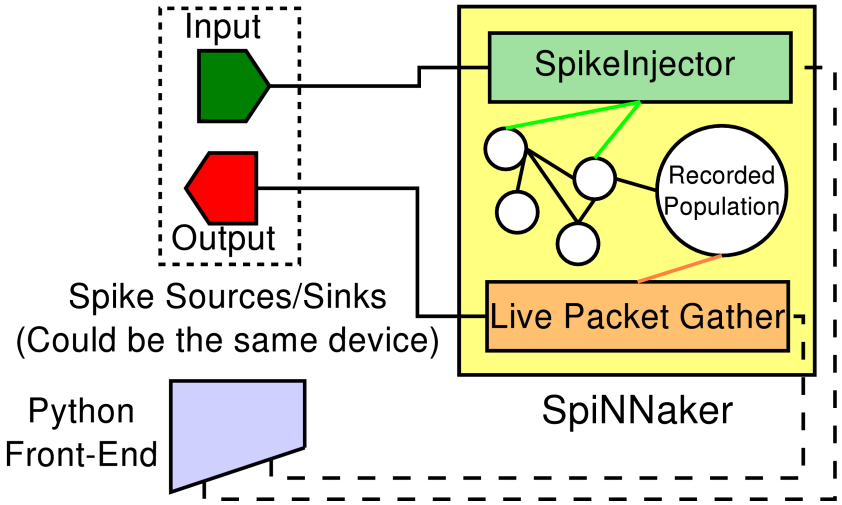


Fig. 3: EIEIO Architecture

2 Main Modules Plus a Python Front-End

SpikeInjector:
UDP-over-Ethernet AER interface for input from external devices

LivePacketGatherer:
UDP-over-Ethernet spike capture and external retransmission

Python Front End:
Two methods:
activate_live_output_for(pop)
sets a LivePacketGatherer.
SpikeInjector(n_neurons, **args)
instantiates a
ReverseIPTagMulticastSource

Live output functionality

Example PyNN script:

```
import pyNN.spiNNaker as p
p.setup (timestep = 1.0)
spike_times = {'spike_times': [[0]]}
spike_source = p.Population(
                1, p.SpikeSourceArray, spike_times, label="ss1")
import spynnaker_external_devices_plugin.pyNN as ext_dev
ext_dev.activate_live_output_for(spike_source)
p.run(100)
```

Default output: 32-bit EIEIO packets with timestamp prefix to identify the time of the event(s)

User-configurable

Configuration options

Live output functionality can be configured:

Parameter	Default	Description
`population`	`-`	PyNN population that will send spikes to the output world
`host, port`	`From .cfg`	Target host IP and UDP port for the packets sent
`use_prefix`	`False`	Use a key prefix in packet header
`key_prefix`	`None`	Key prefix to use
`prefix_type`	`None`	Upper / Lower halfword for key prefix
`message_type`	`32 bit`	16 / 32 bit keys and payloads
`right_shift`	`0`	If keys to be sent are 16 bits, 32-bit keys from SpiNNaker may be right-shifted
`payload_as_time_stamps`	`True`	Payload indicate time stamp of events
`use_payload_prefix`	`True`	Use a payload prefix in packet header
`payload_prefix`	`None`	Payload prefix to use

Processing received spikes

SpynnakerLiveSpikesConnection provides a Python host-based virtual device

Example PyNN script:

```
import pyNN.spiNNaker as p
from spynnaker_external_devices_plugin.pyNN.connections.\
                        spynnaker_live_spikes_connection import
                        SpynnakerLiveSpikesConnection

def processing(label, time, neuron_ids):
    print label, time, neuron_ids

p.setup (timestep = 1.0)
spike_source = p.Population(
       1, p.SpikeSourceArray, {'spike_times': [[0]]}, label='ss1')
import spynnaker_external_devices_plugin.pyNN as ext_dev
ext_dev.activate_live_output_for(spike_source)
live_spikes_connection = SpynnakerLiveSpikesConnection(
                                        receive_labels=['ss1'])
live_spikes_connection.add_receive_callback('ss1', processing)
p.run(100)
```

Configuring live connections

SpynnakerLiveSpikesConnection allows a few parameters for configuration:

Parameter	Default	Description
`receive_labels`	`None`	Label(s) of the population(s) from which spikes are received
`send_labels`	`None`	Label(s) of the population(s) to which spikes will be sent
`local_port`	`19999`	(optional) UDP port on which to listen for incoming messages
`local_host`	`None`	IP address of the interface to listen for incoming messages

Live spike injection

SpikeInjector is a piece of simulation software which runs on SpiNNaker and receives data packet from the external world

Example PyNN script:

```
import pyNN.spiNNaker as p
import spynnaker_external_devices_plugin.pyNN as ext_dev
p.setup (timestep = 1.0)
spike_source = p.Population(
          1, ext_dev.SpikeInjector, {'port':17899}, label='ls1')

ext_dev.activate_live_output_for(spike_source)
p.run(100)
```

In this example, SpiNNaker listens on UDP port 17899 for EIEIO packets to parse.

Synchronising the injector

The injector may be synchronised with the start of the simulation on SpiNNaker using a callback set up for the purpose. Example:

```
def start_sender(label, sender):
    print 'start sending', label, sender

import pyNN.spiNNaker as p
import spynnaker_external_devices_plugin.pyNN as ext_dev
from spynnaker_external_devices_plugin.pyNN.connections.\
                        spynnaker_live_spikes_connection import
                        SpynnakerLiveSpikesConnection

p.setup (timestep = 1.0)
spike_source = p.Population(
           1, ext_dev.SpikeInjector, {'port':17899}, label='ls1')
ext_dev.activate_live_output_for(spike_source)
live_spikes_connection = SpynnakerLiveSpikesConnection(
                                        send_labels=['ls1'])
live_spikes_connection.add_start_callback('ls1', start_sender)
p.run(100)
```

Operations on live connections

Function	Description
`add_receive_callback`	Add a function call upon receiving a message from SpiNNaker
`add_start_callback`	Add a function call upon start of the simulation
`send_spike`	Send an EIEIO packet with a single key to SpiNNaker
`send_spikes`	Send an EIEIO packet with multiple keys to SpiNNaker

Neuron ID vs Routing key

- In a population, neurons are numbered from 0
- In the SpiNNaker machine neurons are globally identified by their routing key (32 bit)
- Remaining bits identify the source population ID between populations

Translation

How to perform the translation?

- Send only the neuron ID to the population (conversion performed on the board)
  (sending spikes to SpiNNaker)

- Use “send_spike” and “send_spikes” functions (conversion performed on the host)
  (sending spikes to SpiNNaker)

- Interface to the database (conversion performed on the host)
  (sending AND receiving spikes to/from SpiNNaker)

Neuron ID transmission

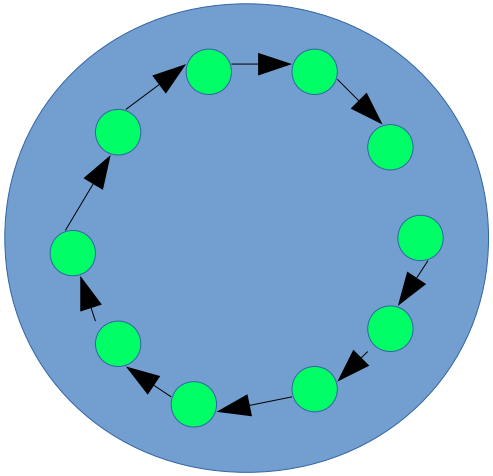
- Key 32 bit: No translation performed – assumed to be fully specified key
- Key 16 bit, key prefix: No translation performed – the combination of prefix and key specifies the key
- Key 16 bit, no key prefix: Translation performed to the virtual key space assigned to the injector population

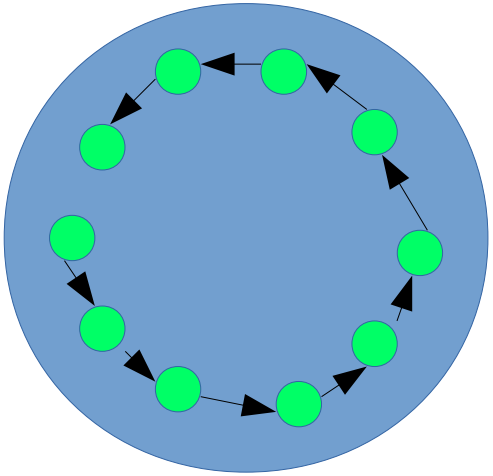
Example – spike_io.py – 1

```
import spynnaker.pyNN as Frontend
import spynnaker_external_devices_plugin.pyNN as ExternalDevices
from spynnaker_external_devices_plugin.pyNN.connections\
    .spynnaker_live_spikes_connection import SpynnakerLiveSpikesConnection

Frontend.setup(timestep=1.0, min_delay=1.0, max_delay=144.0)

n_neurons = 100

cell_params_lif = {'cm': 0.25,
                   'i_offset': 0.0,
                   'tau_m': 20.0,
                   'tau_refrac': 2.0,
                   'tau_syn_E': 5.0,
                   'tau_syn_I': 5.0,
                   'v_reset': -70.0,
                   'v_rest': -65.0,
                   'v_thresh': -50.0}

cell_params_spike_injector = {'port': 12345}

cell_params_spike_injector_with_key = {
    'port': 12346,
    'virtual_key': 0x70000}
```

Example – spike_io.py – 2

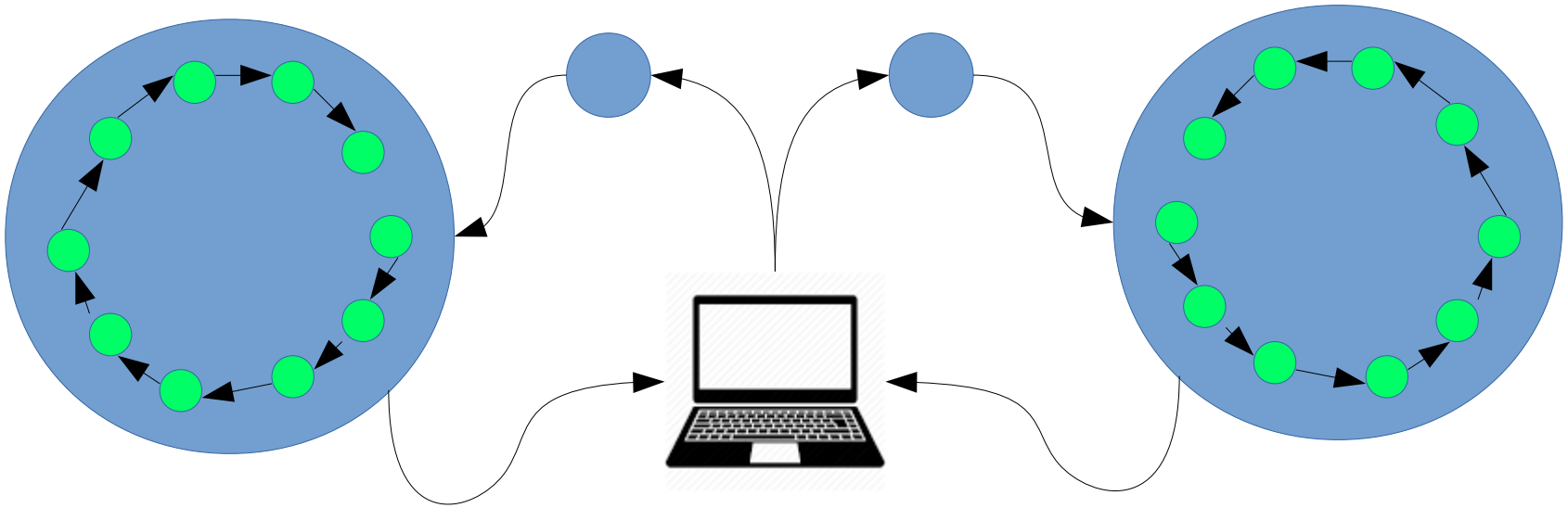
```
pop_forward = Frontend.Population(n_neurons, Frontend.IF_curr_exp,
                                  cell_params_lif, label='pop_forward')
pop_backward = Frontend.Population(n_neurons, Frontend.IF_curr_exp,
                                   cell_params_lif, label='pop_backward')

loop_forward = list()
loop_backward = list()
for i in range(0, n_neurons - 1):
    loop_forward.append((i, (i + 1) % n_neurons, weight_to_spike, 3))
    loop_backward.append(((i + 1) % n_neurons, i, weight_to_spike, 3))
Frontend.Projection(pop_forward, pop_forward,
                    Frontend.FromListConnector(loop_forward))
Frontend.Projection(pop_backward, pop_backward,
                    Frontend.FromListConnector(loop_backward))
```





Example – spike_io.py – 3

```
injector_forward = Frontend.Population(n_neurons, ExternalDevices.SpikeInjector,
    cell_params_spike_injector_with_key, label='spike_injector_forward')
injector_backward = Frontend.Population(n_neurons, ExternalDevices.SpikeInjector,
    cell_params_spike_injector, label='spike_injector_backward')

Frontend.Projection(injector_forward, pop_forward,
                    Frontend.OneToOneConnector(weights=weight_to_spike))
Frontend.Projection(injector_backward, pop_backward,
                    Frontend.OneToOneConnector(weights=weight_to_spike))

ExternalDevices.activate_live_output_for(pop_forward,
database_notify_host="localhost", database_notify_port_num=19996)
ExternalDevices.activate_live_output_for(pop_backward,
database_notify_host="localhost", database_notify_port_num=19996)
```



Example – spike_io.py – 4

```
live_spikes_connection_sending = SpynnakerLiveSpikesConnection(
    receive_labels=None, local_port=19999,
    send_labels=["spike_injector_forward", "spike_injector_backward"])

live_spikes_connection_sending.add_start_callback("spike_injector_forward",
                                                  send_input_forward)
live_spikes_connection_sending.add_start_callback("spike_injector_backward",
                                                  send_input_backward)



live_spikes_connection_receive = SpynnakerLiveSpikesConnection(
    receive_labels=["pop_forward", "pop_backward"],
    local_port=19996, send_labels=None)

# Set up callbacks to occur when spikes are received
live_spikes_connection_receive.add_receive_callback("pop_forward",
                                                    receive_spikes)
live_spikes_connection_receive.add_receive_callback("pop_backward",
                                                    receive_spikes)
```

Example – spike_io.py – 5

```
# Create a sender of packets for the forward population
def send_input_forward(label, sender):
    for neuron_id in range(0, 100, 20):
        time.sleep(random.random() + 0.5)
        print_condition.acquire()
        print "Sending forward spike", neuron_id
        print_condition.release()
        sender.send_spike(label, neuron_id, send_full_keys=True)


# Create a sender of packets for the backward population
def send_input_backward(label, sender):
    for neuron_id in range(0, 100, 20):
        real_id = 100 - neuron_id - 1
        time.sleep(random.random() + 0.5)
        print_condition.acquire()
        print "Sending backward spike", real_id
        print_condition.release()
        sender.send_spike(label, real_id)


# Create a receiver of live spikes
def receive_spikes(label, time, neuron_ids):
    for neuron_id in neuron_ids:
        print_condition.acquire()
        print "Received spike at time", time, "from", label, "-", neuron_id
        print_condition.release()
```

Example – spike_io.py – 6

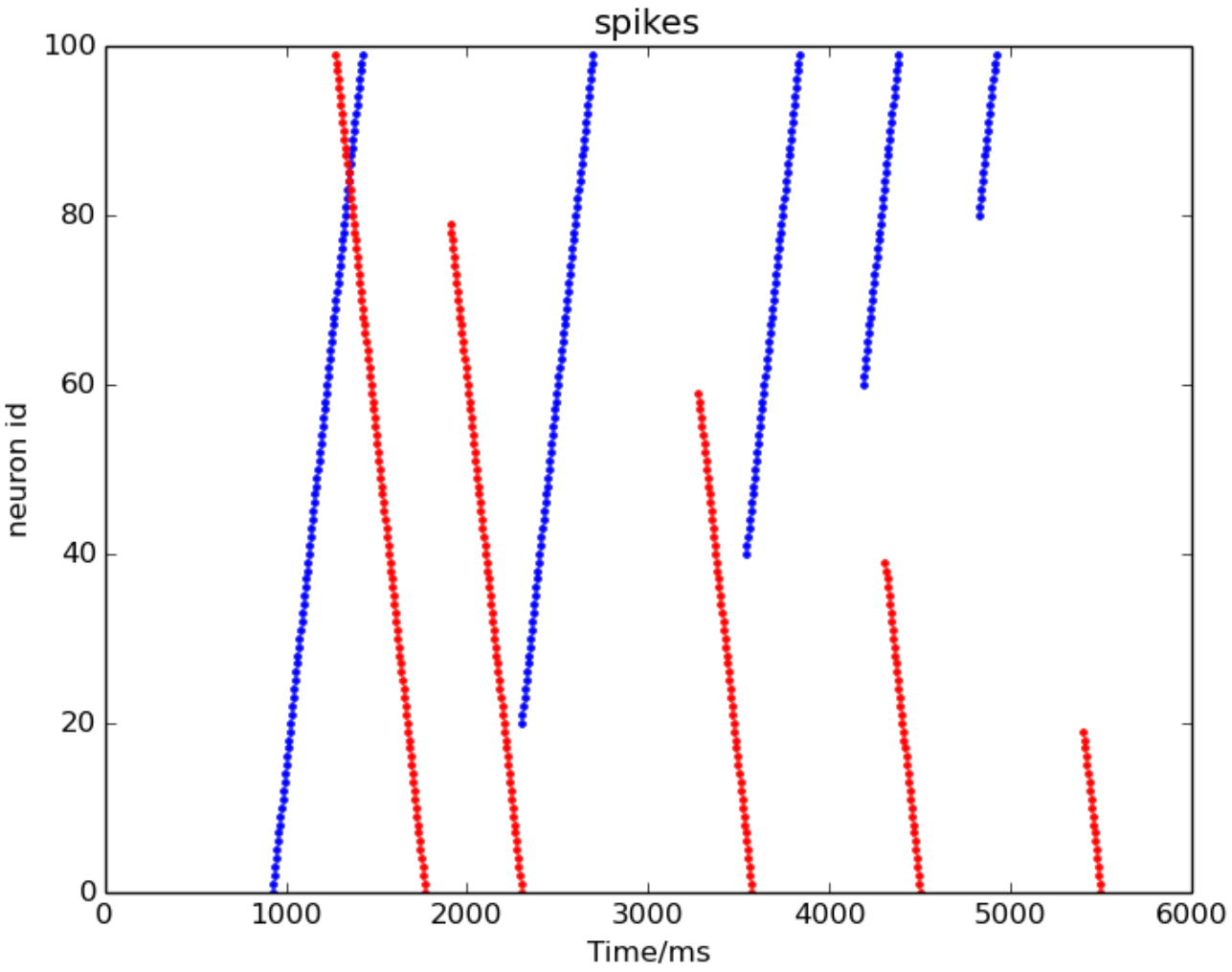


Fig4: Output of the spike_io.py network

External Devices Through The SpiNNaker Link

- A method of direct connection using native AER links
- Usually requires an FPGA interface board
- Supported in the toolchain via "virtual chips"

Fig4: FPGA connector

How to connect devices to a SpiNNaker board

- Connect the device to the SpiNNaker link connector

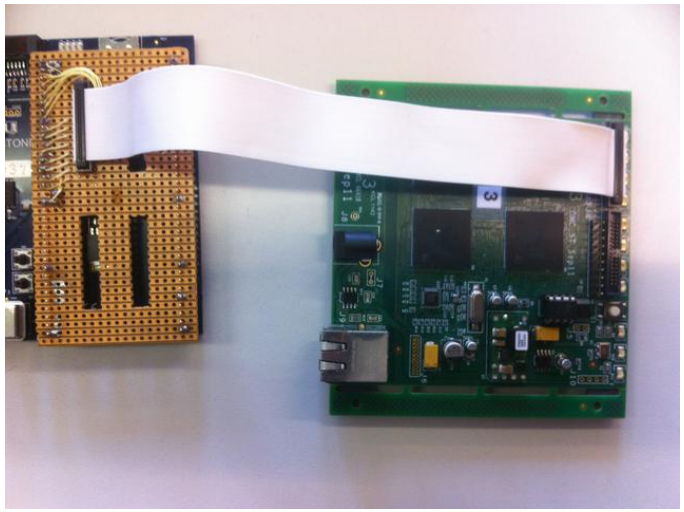


Fig5: connecting to a spinn-3 Board

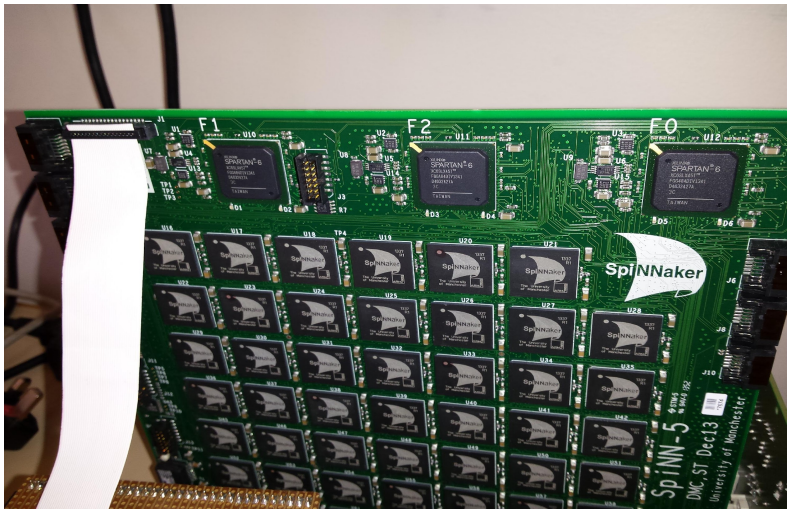


Fig6: connecting to a spinn-5 Board

How external devices are represented in sPyNNaker

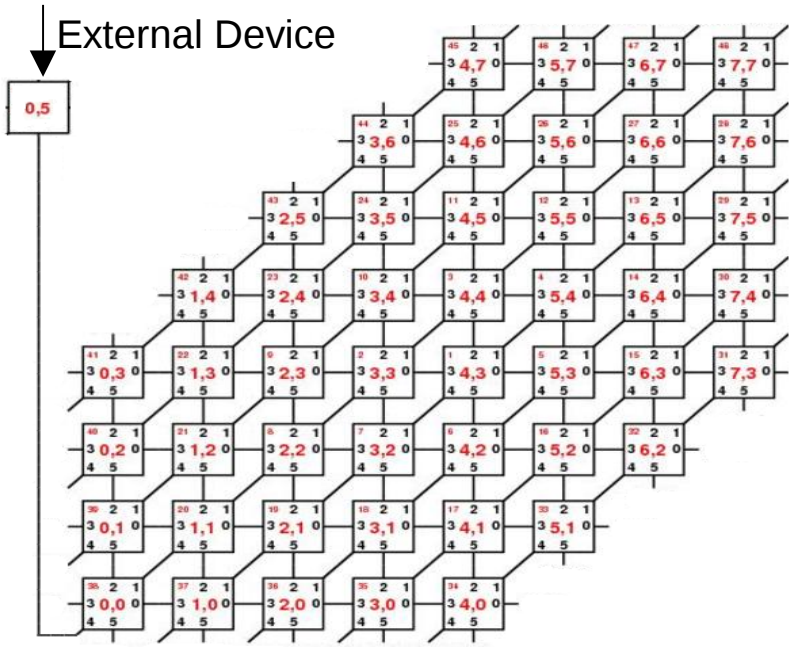


Fig7: Structure for either a Spinn-4 or Spinn-5 boards

1. External devices are represented in sPyNNaker by virtual ID's.

2. Each virtual ID must occupy a chip address space not currently used by the machine ([0,5],[0,6],[0,7],[5,2], etc.)

3. Each virtual chip is routed to a real chip (chip [0,0] in this case)

4. The connection to the real chip must be through a link not currently used by the real chip (links [W,SW,S] in this case)

External devices: Pacman

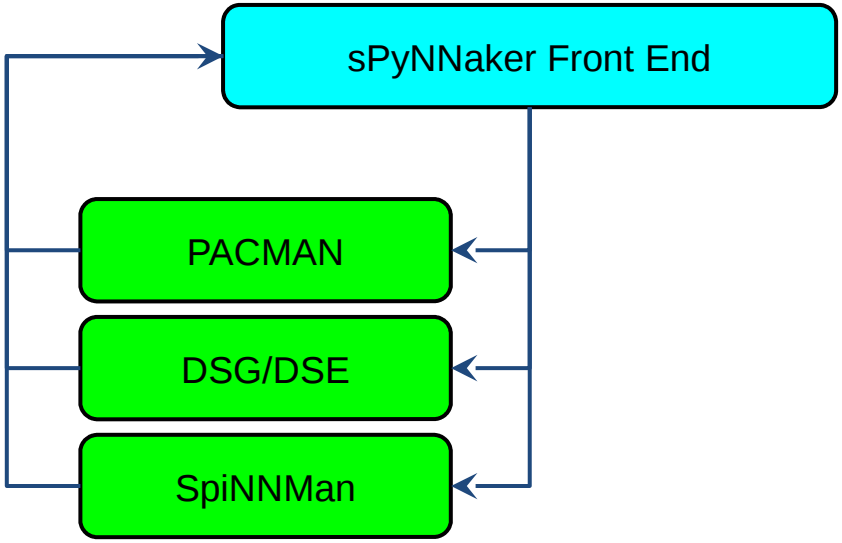


Fig8: The sPyNNaker stack

2.1 Partitioning would produce one subvertex *SV* with all the neurons

2.2 Placement would place *SV* into the virtual address range defined for the external device.

2.3 Routing will treat the virtual addresses as if they were normal ones and route stuff accordingly.

External devices: DSG/SpinnMan

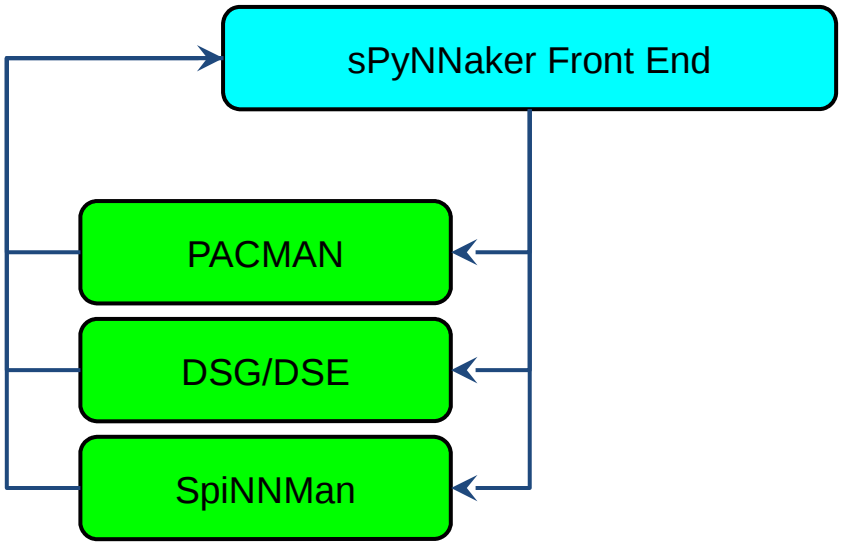


Fig8 (*redux*): The sPyNNaker stack

3.1 No data spec is generated for the virtual device

3.2.1 No data spec is therefore executed for models running on the external device

3.2.2 Data specs are generated for models running on real spinnaker cores

3.3 Should occur correctly.

SpinnMan does not change

Adding a new external device

Calls from PYNN

```
import pynn.spinnaker as p
p.setup( timestep=1.0,
         min_delay = 1.0,
         max_delay = 32.0)
external_device_requirements = {
   ‘spinnaker_link’:0,
     'XXXXXXX'=XXXXXX}
external_device =
   p.Population(1, p.New_External_device,
                    external_device_requirements,
                    label='External retina'))
external_device.record()
p.run(10000)
```

An example script

Differences

Step3:
Add any new parameters that the new device model requires

Step4:
Initializes the new device as a population

Coming Soon...

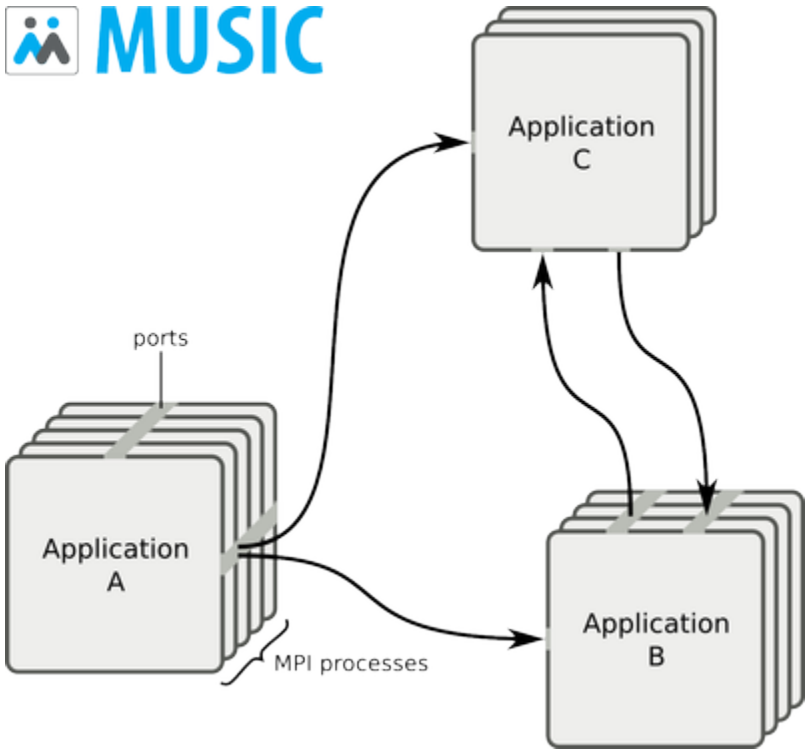


Fig. 9 MUSIC
(Image from Mikael Djurfelt and the MUSIC Project 2015)

Updated AER Protocol:
Will support keys, payloads, and timestamps in the same packet, 64-bit data types.

MUSIC interface:
A universal protocol for intersystem communications