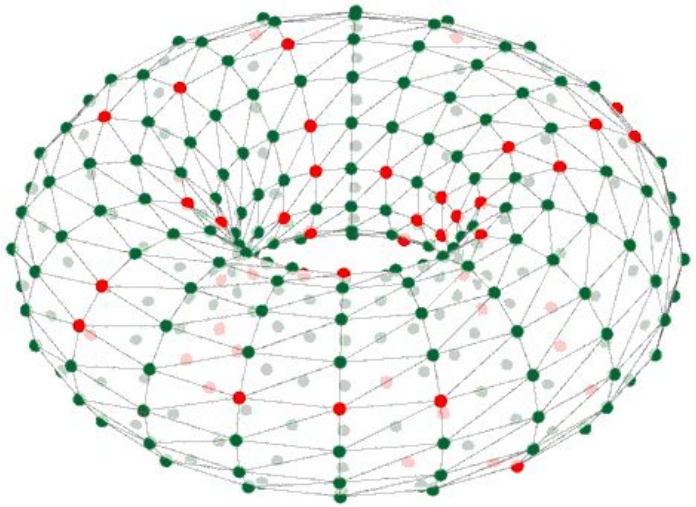
External Devices



Alan Stokes, Andrew Rowley

SpiNNaker Workshop
October 2017

European Research Council
Established by the European Commission

Human Brain Project

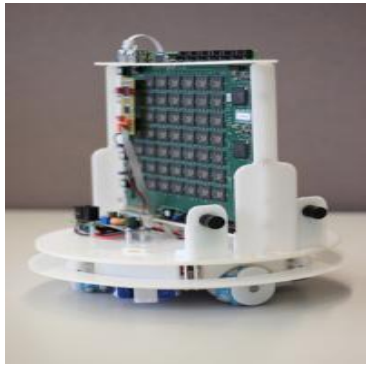
EPSRC

MANCHESTER 1824

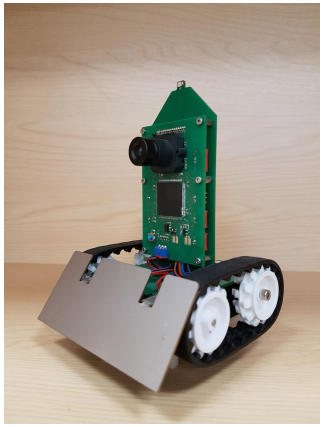
Contents

1. How to add external devices that communicate through the SpiNNaker Link into your PyNN scripts.
2. How to use a Push Bot in your PyNN scripts.
3. How to add external devices that communicate through the FPGA/SATA connector into your PyNN scripts.
4. How FPGA's are used within multi board systems.
5. The underlying protocol external devices need to implement.
6. How to build your own external device.

Real time systems?

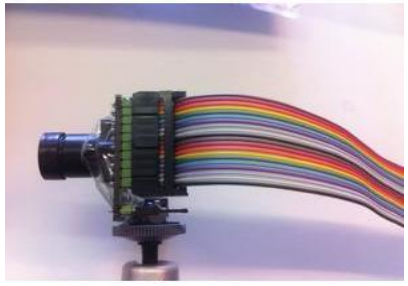


SpOmnibot
(Retinas & Motors)

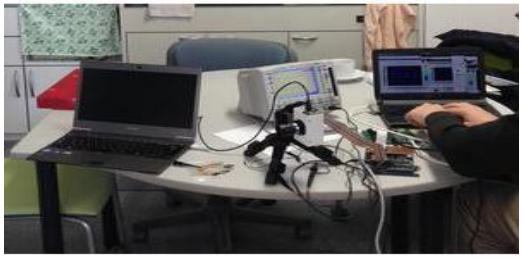


Push bot
(Retinas, Motors, lasers, speaker, leds, gyro, etc)

FPGA connector



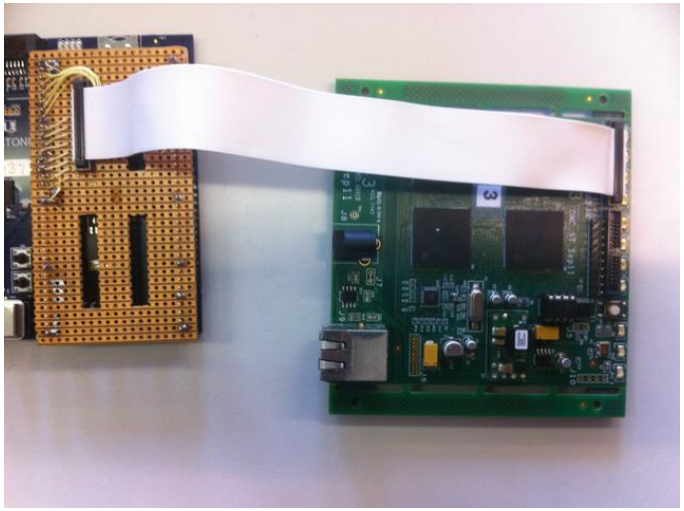
A Retina



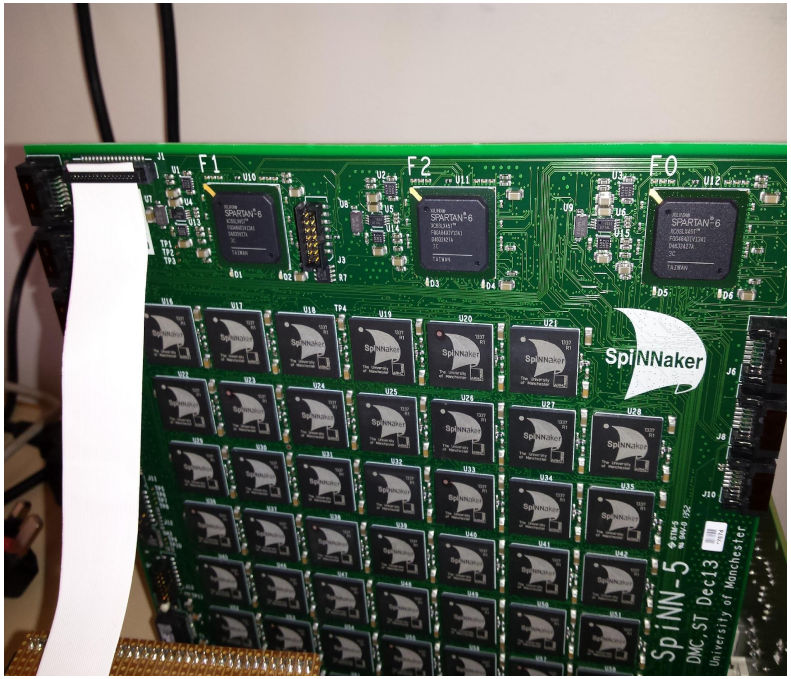
A Osaka retina

A Cochlea

How to connect devices to a spiNNaker board?

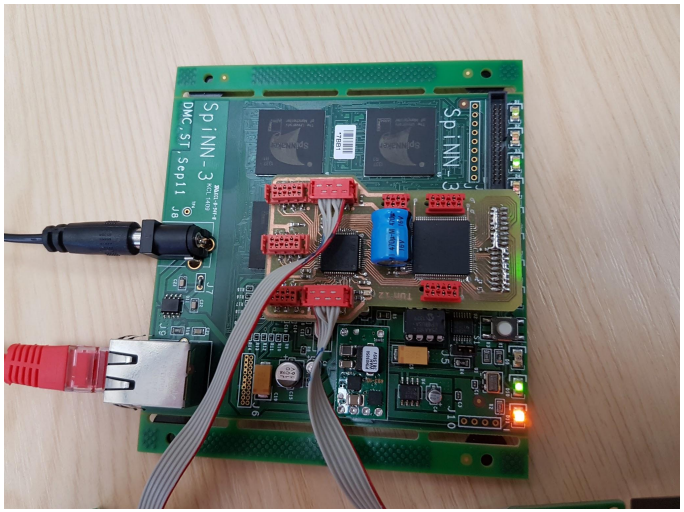


connecting to a spinn-3 Board

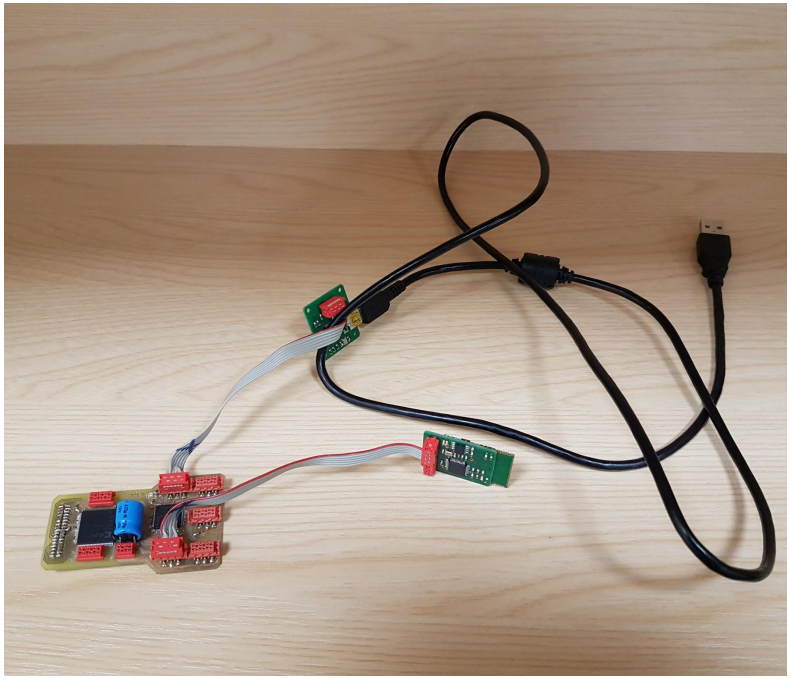


connecting to a spinn-5 Board

How to connect push bot to a spiNNaker board



connecting to a Board



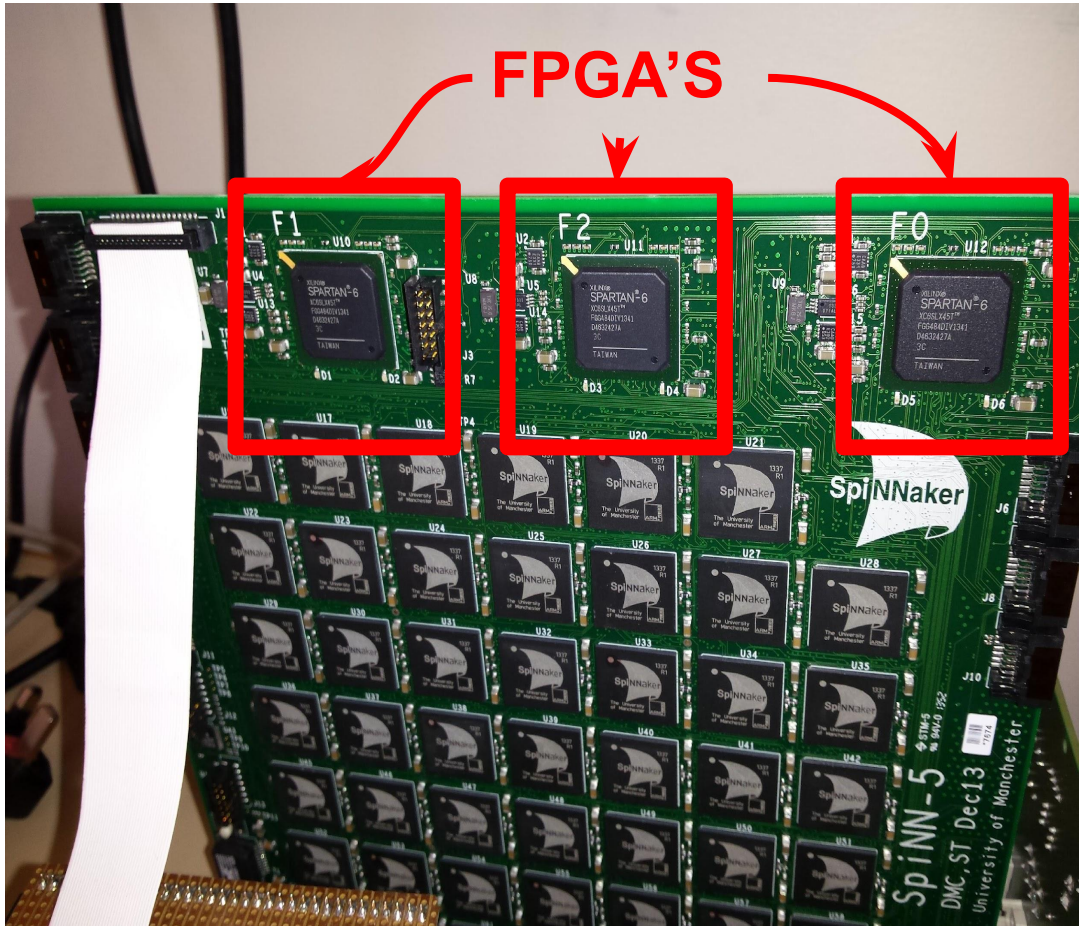
WiFi transmitter module

Can also be connected via your laptop wifi and the ethernet. But not recommended!!!!!

FPGA Programming for SpiNN-5

1. The FPGA's need repogramming to support external device plugin.

2. This reprogramming is not done by the tools to date.



Using an external device: Calls from PYNN

```
import spynnaker8 as p
p.setup( timestep=1.0,
         min_delay = 1.0,
         max_delay = 32.0)

# set up populations
pop =  p.Population(
     1, p.IfCurExp, {} label='pop1'))



# set populations to record spikes
pop.record()

# run the simulation for 10000 ms
p.run(10000)
```

Using an external device: Calls from PYNN

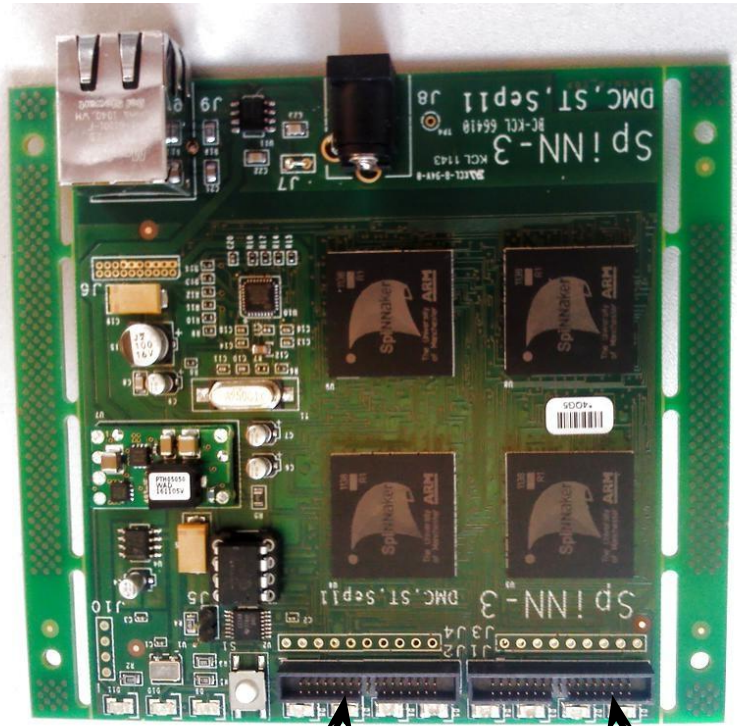
```
import spynnaker8 as p
p.setup( timestep=1.0,
         min_delay = 1.0,
         max_delay = 32.0)

# set up populations
pop =  p.Population(
      1, p.external_devices.MyExternalDevice, {
            'spinnaker_link':0,
             'board_address':None OR "192.168.0.253"}, label='pop1'))

# set populations to record spikes
pop.record()
```

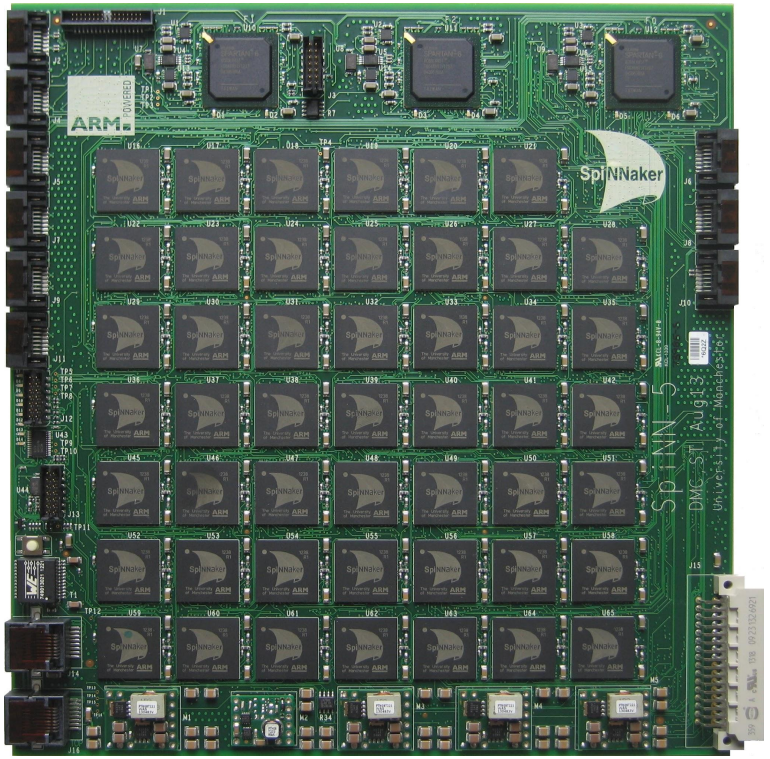
External devices can't be recorded

```
# run the simulation for 10000 ms
p.run(10000)
```

Which SpiNNaker Link is which?

SpiNNakerLinkID = 0



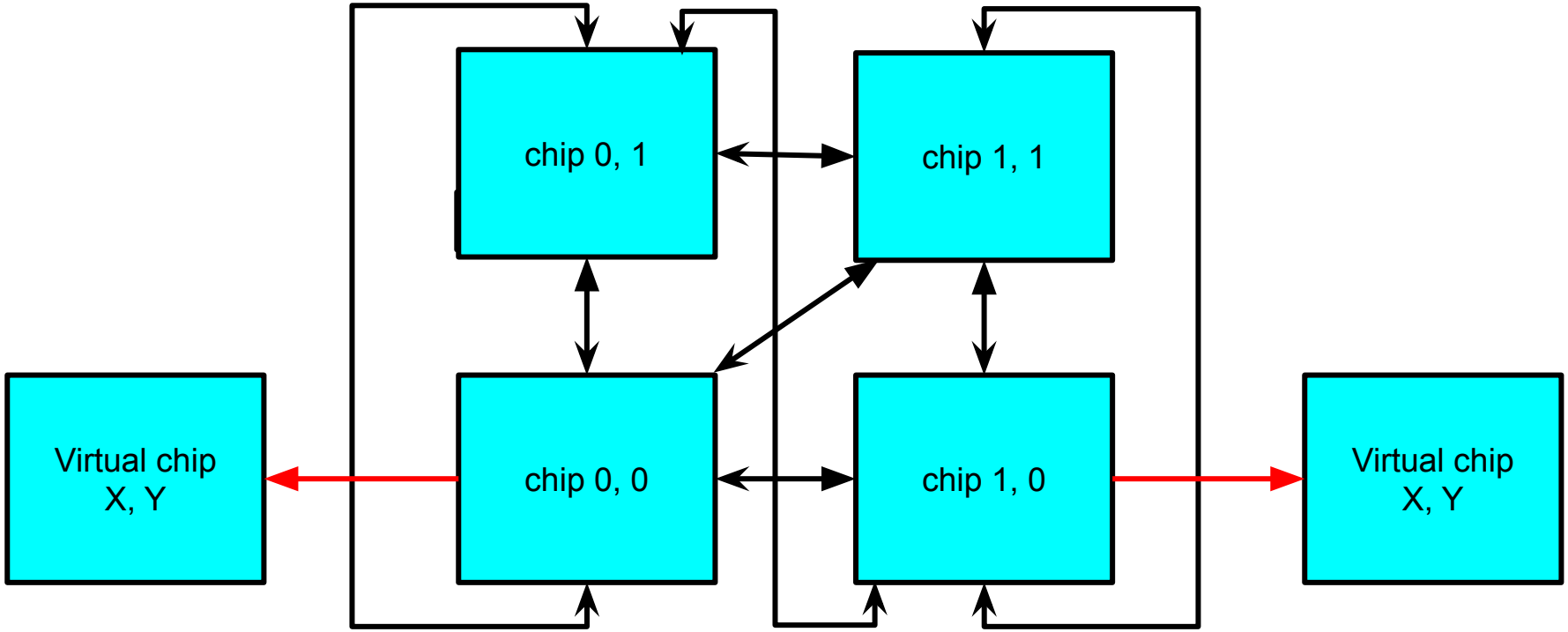


SpiNNakerLinkID = 0

SpiNNakerLinkID = 1

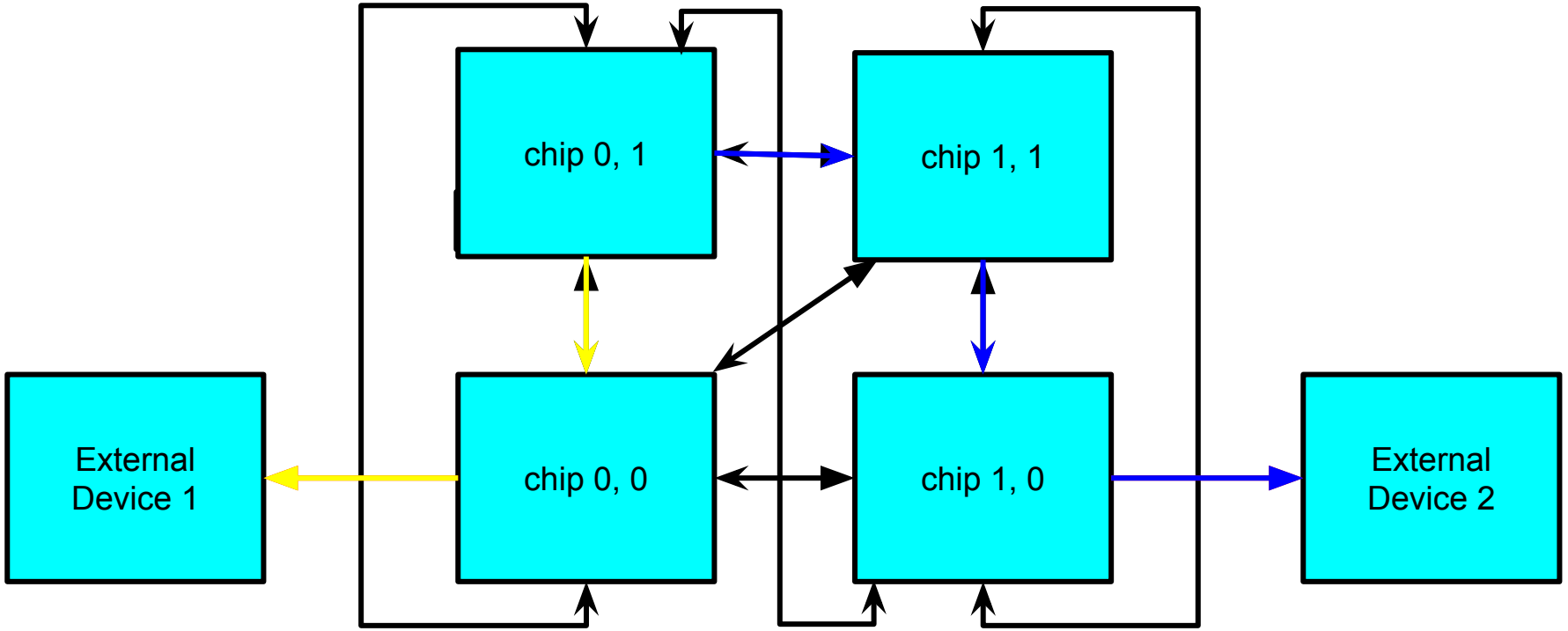
How this works in detail

1. Every Spinnaker link is defined as a link to a virtual chip
2. Your device vertex is then placed within this virtual chip.



Why does it matter?

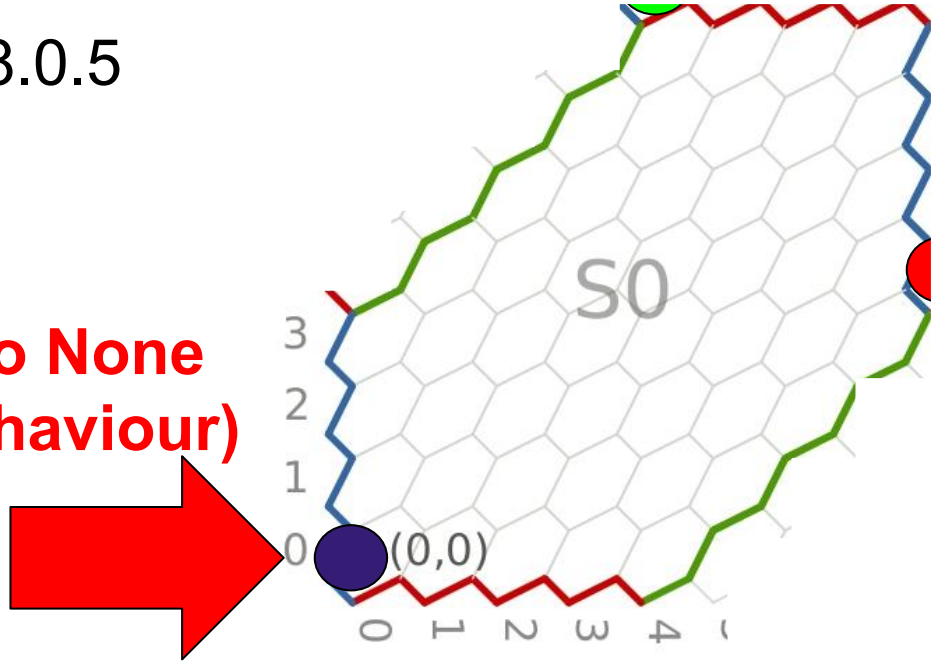
1. Routing won't work if mixed up



Board Address?

- 192.168.0.1
- 192.168.0.3
- 192.168.0.5

When set to None (default behaviour)



SpiNNaker link Supported populations! 1 of 3

```
import spynnaker8 as p

pop1 = p.Population(
    size=6,
    cellclass=p.external_devices.MunichMotorDevice(
        spinnaker_link_id=0, board_address=None))
```

SpiNNaker link Supported populations! 2 of 3

```
pop2 = p.Population(
    size=None,
    cellclass=p.external_devices.MunichRetinaDevice(
        spinnaker_link_id=0, board_address=None, retina_key=0x5,
        position=p.external_devices.MunichRetinaDevice.LEFT_RETINA,
        polarity=p.external_devices.ExternalFPGARetinaDevice.DOWN_POLARITY})
```

Position could be: **LEFT_RETINA**
RIGHT_RETINA

Polarity could be: **UP_POLARITY**
DOWN_POLARITY
MERGED_POLARITY

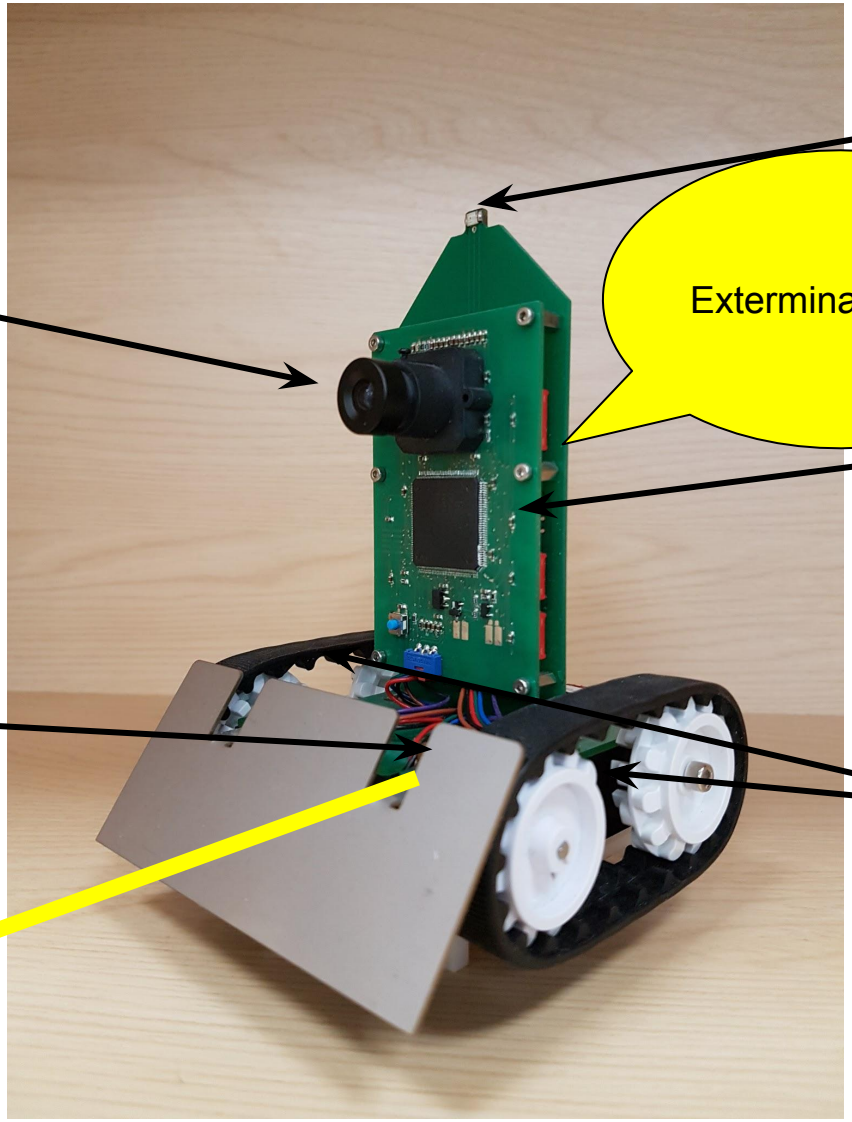
SpiNNaker link Supported populations! 3 of 3

```
pop4 = p.Population(
    size=None, cell_class=p.external_devices.ExternalFPGARetinaDevice(
        spinnaker_link_id=0, board_address=None, retina_key=0x5,
        mode=p.external_devices.ExternalFPGARetinaDevice.MODE_128,
        polarity=p.external_devices.ExternalFPGARetinaDevice.DOWN_POLARITY))
```

Mode could be : **MODE_128**
MODE_64
MODE_32
MODE_16

Polarity could be: **UP_POLARITY**
DOWN_POLARITY
MERGED_POLARITY

Push Bot!



Push Bot: SpiNNaker link 1 of 10 Full script! AHHHHH

Found at

https://github.com/SpiNNakerManchester/PyNN8Examples/blob/master/examples/external_devices_examples/pushbot_spinnaker_link_example.py

Looks like below:

```
import spynnaker8 as p

p.setup(1.0)

# Set up the PushBot devices
pushbot_protocol = p.external_devices.MunichIoSpiNNakerLinkProtocol(
    mode=p.external_devices.MunichIoSpiNNakerLinkProtocol.MODES.PUSH_BOT,
    uart_id=0)
spinnaker_link = 0
board_address = None
motor_0 = p.external_devices.PushBotSpiNNakerLinkMotorDevice(
    p.external_devices.PushBotMotor.MOTOR_0_PERMANENT, pushbot_protocol,
    spinnaker_link, board_address=board_address)
motor_1 = p.external_devices.PushBotSpiNNakerLinkMotorDevice(
    p.external_devices.PushBotMotor.MOTOR_1_PERMANENT, pushbot_protocol,
    spinnaker_link, board_address=board_address)
speaker = p.external_devices.PushBotSpiNNakerLinkSpeakerDevice(
    p.external_devices.PushBotSpeaker.SPEAKER_TONE, pushbot_protocol,
    spinnaker_link, board_address=board_address)
laser = p.external_devices.PushBotSpiNNakerLinkLaserDevice(
    p.external_devices.PushBotLaser.LASER_ACTIVE_TIME, pushbot_protocol,
    spinnaker_link, board_address=board_address, start_total_period=1000)
led_front = p.external_devices.PushBotSpiNNakerLinkLEDDevice(
    p.external_devices.PushBotLED.LED_FRONT_ACTIVE_TIME, pushbot_protocol,
    spinnaker_link, board_address=board_address,
    start_total_period=1000)
led_back = p.external_devices.PushBotSpiNNakerLinkLEDDevice(
    p.external_devices.PushBotLED.LED_BACK_ACTIVE_TIME, pushbot_protocol,
    spinnaker_link, board_address=board_address,
    start_total_period=1000)

weights = {
    motor_0: 10.0,
    motor_1: 10.0,
    speaker: 100.0,
    laser: 100.0,
    led_front: 100.0,
    led_back: 100.0,
}
```

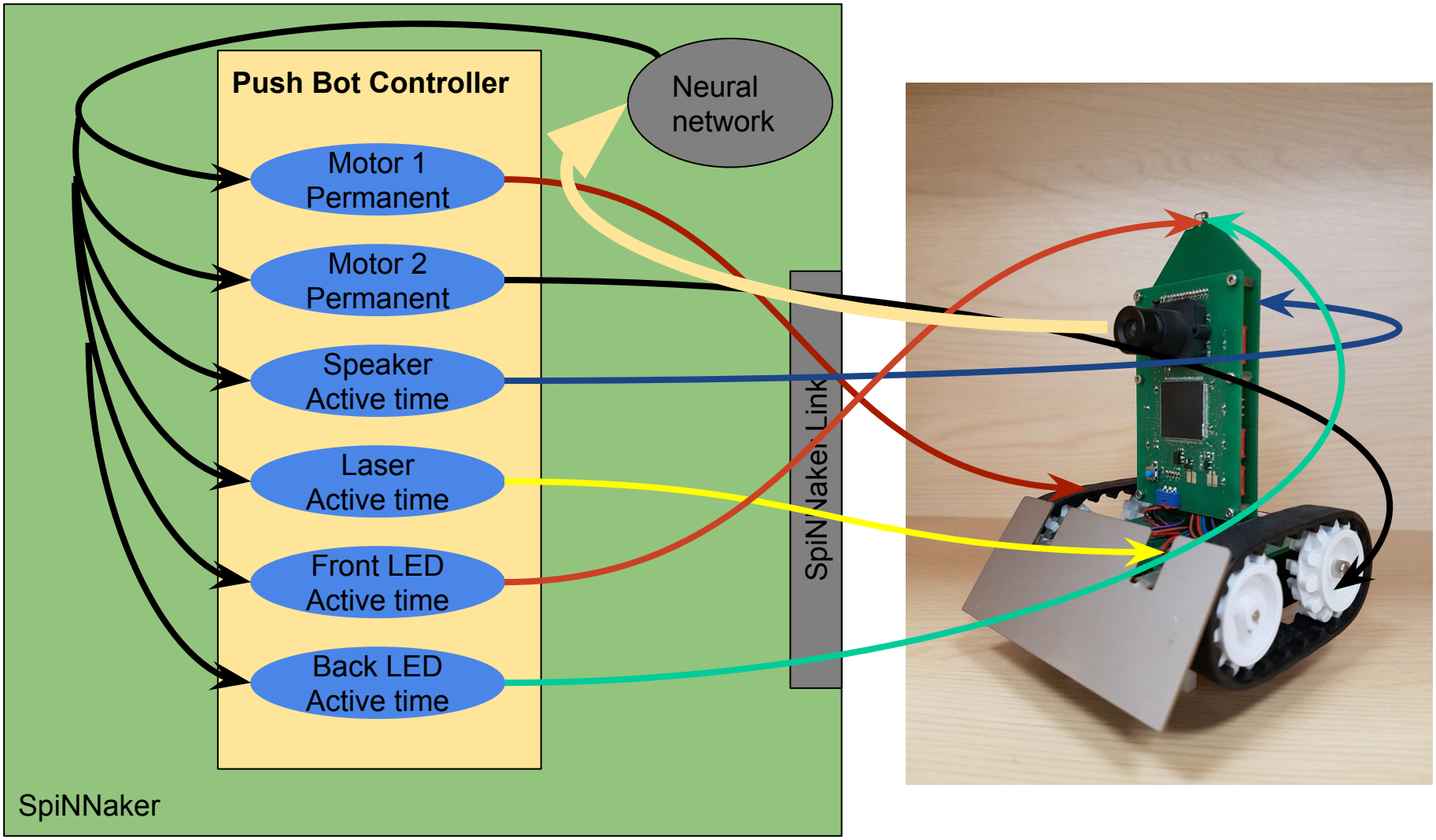
```
devices = [motor_0, motor_1, speaker, laser, led_front, led_back]

# Set up the PushBot control
pushbot = p.Population(
    len(devices), p.external_devices.PushBotLifSpinnakerLink(
        protocol=pushbot_protocol,
        devices=devices,
        tau_syn_E=500.0),
    label="PushBot"
)

# Send in some spikes
stimulation = p.Population(
    len(devices), p.SpikeSourceArray(
        spike_times=[[i * 1000] for i in range(len(devices))]),
    label="input"
)

connections = [
    (i, i, weights[device], 1) for i, device in enumerate(devices)
]
p.Projection(stimulation, pushbot, p.FromListConnector(connections))

retina_resolution = \
    p.external_devices.PushBotRetinaResolution.DOWNSAMPLE_64_X_64
pushbot_retina = p.Population(
    retina_resolution.value.n_neurons,
    p.external_devices.PushBotSpiNNakerLinkRetinaDevice(
        spinnaker_link_id=spinnaker_link,
        board_address=board_address,
        protocol=pushbot_protocol,
        resolution=retina_resolution))

viewer = p.external_devices.PushBotRetinaViewer(
    retina_resolution.value, port=17895)
p.external_devices.activate_live_output_for(
    pushbot_retina, port=viewer.local_port, notify=False)

viewer.start()
p.run(len(devices) * 1000)
p.end()
```

Push Bot: SpiNNaker link 2 of 10 Control module

```
Devices = [motor_0, motor_1, speaker_device, laser, led_front, led_back]

control = p.Population(
    size=len(devices),
    cell_class=p.external_devices.PushBotLifSpinnakerLink(
        protocol=pushbot_protocol, devices=devices, ......))
```

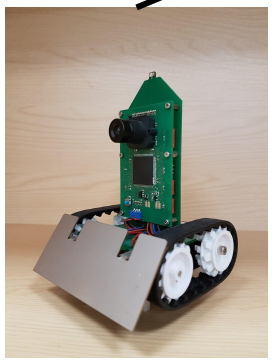
PushBotLifSpinnakerLink also has most of the params from a LIF neuron. They all have defaults to avoid you needing to fill them in.

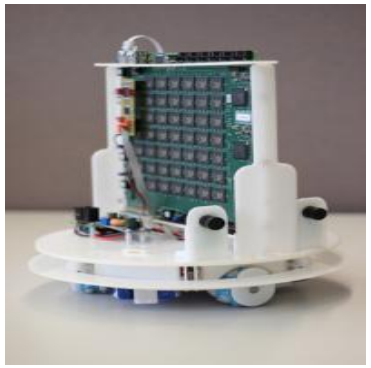
Push Bot: SpiNNaker link 3 of 10
Control module



Push Bot: SpiNNaker link 4 of 10 Protocol

```
pushbot_protocol = p.external_devices.MunichIoSpiNNakerLinkProtocol(
    mode=p.external_devices.MunichIoSpiNNakerLinkProtocol.MODES.PUSH_BOT,
    uart_id=0)
```





Ball balancer,
Oro_botics
others.....

Push Bot: SpiNNaker link 5 of 10 Retina

```
retina = p.Population(
    size=None,
    cell_class=p.external_devices.PushBotSpiNNakerLinkRetinaDevice(
        spinnaker_link_id=0, board_address=None,
        protocol=pushbot_protocol,
        resolution=p.external_devices.PushBotRetinaResolution.
                    DOWNSAMPLE_64_X_64))
```

Resolution could be: **NATIVE_128_X_128**
DOWNSAMPLE_64_X_64
DOWNSAMPLE_32_X_32
DOWNSAMPLE_16_X_16

Push Bot: SpiNNaker link 6 of 10 Retina viewer

```
viewer = p.external_devices.PushBotRetinaViewer(
        resolution=retina_resolution.value, port=17895)

p.external_devices.activate_live_output_for(
    population=pushbot_retina, port=viewer.local_port, notify=False)

viewer.start()
```

Push Bot: SpiNNaker link 7 of 10 Motors

```
motor_0 = p.external_devices.PushBotSpiNNakerLinkMotorDevice(
    motor=p.external_devices.PushBotMotor.MOTOR_0_PERMANENT,
    protocol=pushbot_protocol,
    spinnaker_link=0, board_address=None)
```

Motor could be: **MOTOR_0_PERMANENT,**
MOTOR_0_LEAKY,
MOTOR_1_PERMANENT,
MOTOR_1_LEAKY

Push Bot: SpiNNaker link 8 of 10 Speaker

```
speaker_device = p.external_devices.PushBotSpiNNakerLinkSpeakerDevice(
    speaker=p.external_devices.PushBotSpeaker.SPEAKER_TONE,
    protocol=pushbot_protocol,

    start_active_time=50, start_total_period=None,
    start_frequency=None, start_melody=None, spinnaker_link=0,
    board_address=None)
```

Speaker could be:
- **SPEAKER_TOTAL_PERIOD**
- **SPEAKER_ACTIVE_TIME**

} **Pulse Width Modulation**

- **SPEAKER_TONE**
- **SPEAKER_MELODY**

Push Bot: SpiNNaker link 9 of 10 Laser

```
laser = p.external_devices.PushBotSpiNNakerLinkLaserDevice(
    laser=p.external_devices.PushBotLaser.LASER_ACTIVE_TIME,
    protocol=pushbot_protocol,

    start_active_time=50, start_total_period=None,
    start_frequency=None, spinnaker_link=0, board_address=None,
    start_total_period=1000)
```

Laser could be: **LASER_TOTAL_PERIOD**
LASER_ACTIVE_TIME
LASER_FREQUENCY

} **Pulse Width Modulation**

Push Bot: SpiNNaker link 10 of 10 LED

```
led_front = p.external_devices.PushBotSpiNNakerLinkLEDDevice(
    led=p.external_devices.PushBotLED.LED_FRONT_ACTIVE_TIME,
    protocol=pushbot_protocol,
    spinnaker_link=0, board_address=None,

    start_total_period=1000, start_active_time_front=None,
    start_active_time_back=None, start_total_period=None, start_frequency=None)
```

Led could be: **LED_FRONT_ACTIVE_TIME**
LED_BACK_ACTIVE_TIME
LED_FREQUENCY
LED_TOTAL_PERIOD

} **Pulse Width Modulation**

Push bot: Ethernet? 1 of 4

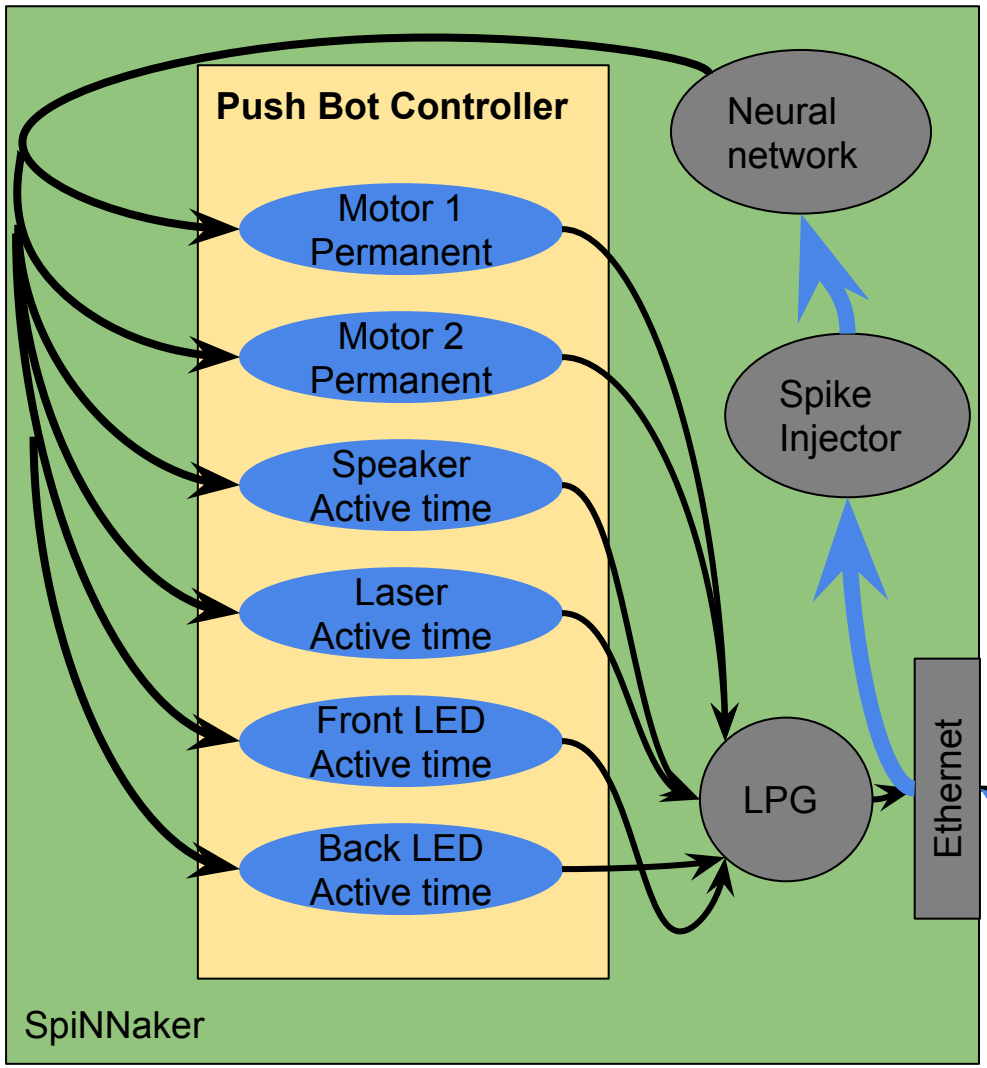
Full example found here:
https://github.com/SpiNNakerManchester/PyNN8Examples/blob/master/examples/external_devices_examples/pushbot_ethernet_example.py

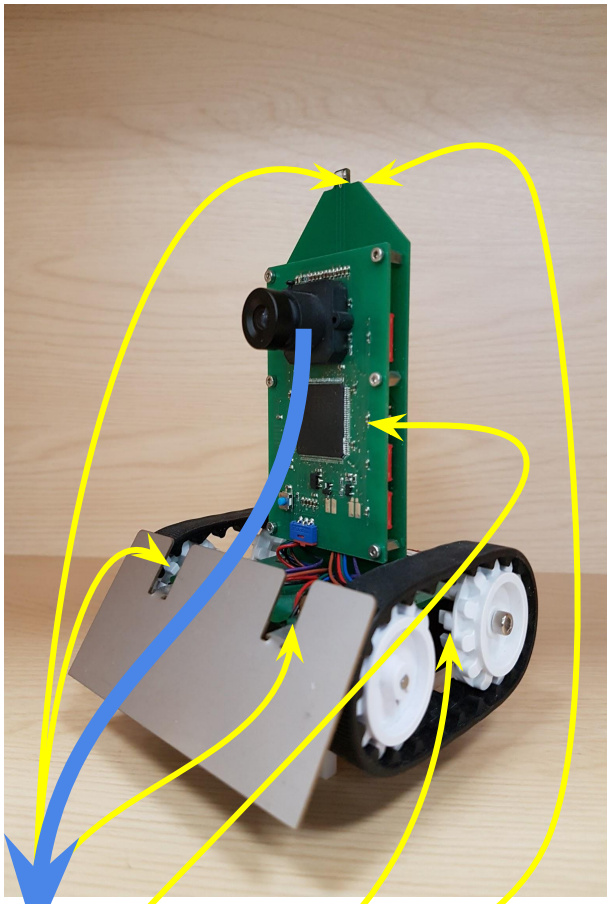
1. Each device has a ethernet counterpart:

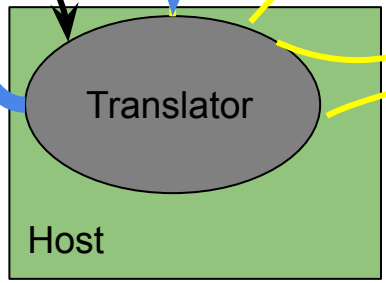
SpiNNaker Link device	**Ethernet Device counterpart**
PushBotSpiNNakerLinkMotorDevice	PushBotEthernetMotorDevice
PushBotSpiNNakerLinkSpeakerDevice	PushBotEthernetSpeakerDevice
PushBotSpiNNakerLinkLaserDevice	PushBotEthernetLaserDevice
PushBotSpiNNakerLinkLEDDevice	PushBotEthernetLEDDevice
PushBotSpiNNakerLinkRetinaDevice	PushBotEthernetRetinaDevice
PushBotLifSpinnakerLink	PushBotLifEthernet

All ethernet vertices have the same parameters as the spinnaker link versions. Except the PushBotLifEthernet

Push Bot: Ethernet 2 of 4



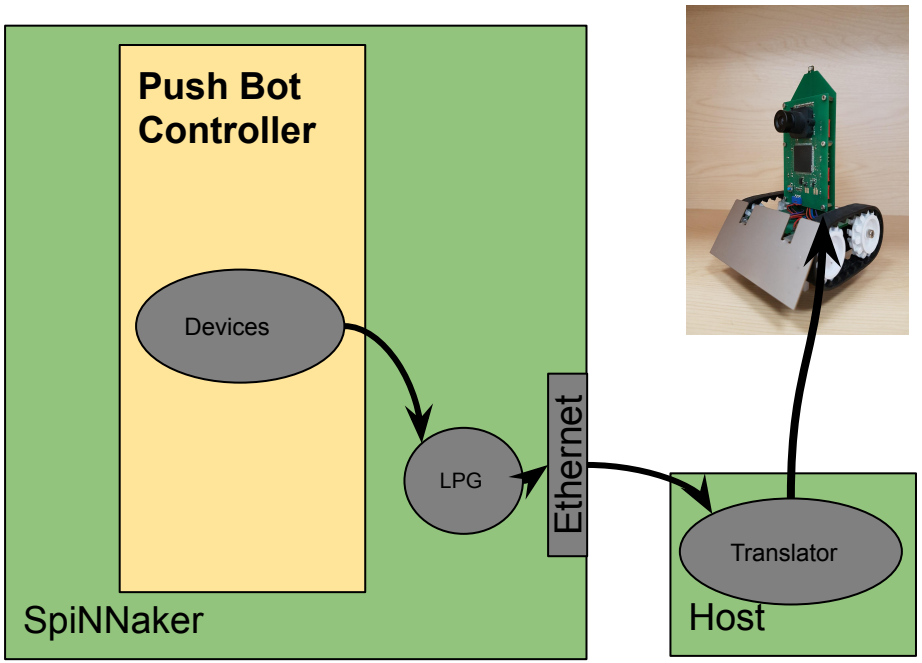




Push bot: Ethernet? 3 of 4

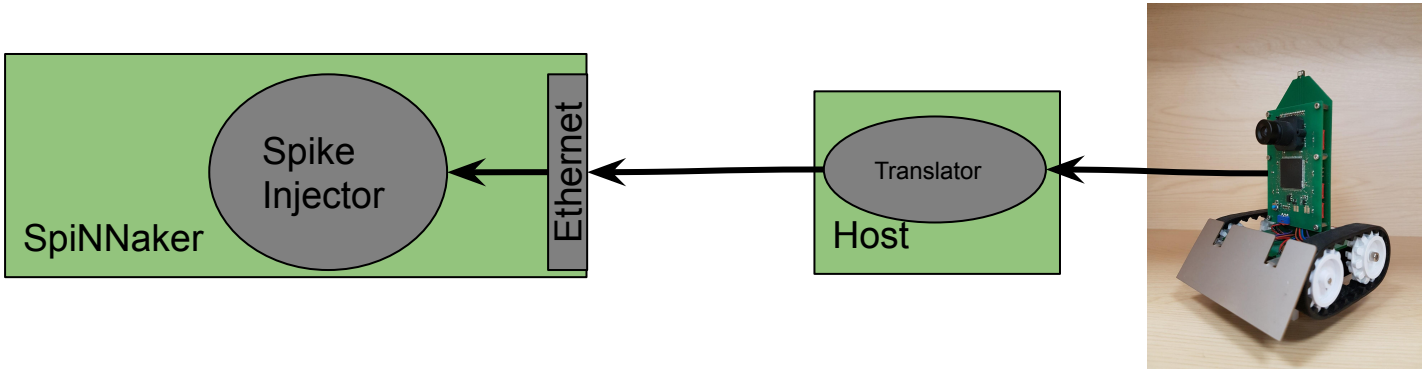
Creating the PushBotLifEthernet population.

```
p.external_devices.EthernetControlPopulation(
   len(devices), p.external_devices.PushBotLifEthernet(
       protocol=pushbot_protocol,
       devices=devices,
       pushbot_ip_address="10.162.177.57",.....))
```



Push bot: Ethernet? 4 of 4

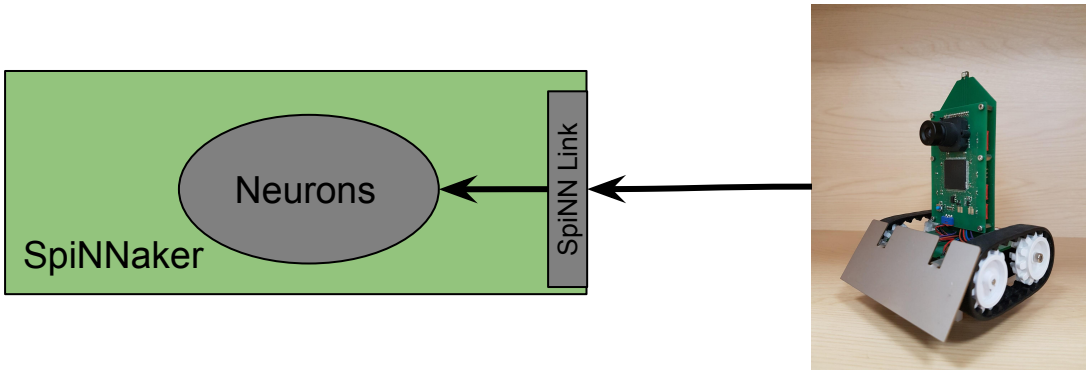
Adding the ethernet retina population to the PyNN script



p.external_devices.**EthernetSensorPopulation**(**PushBotEthernetRetinaDevice**(...))

Instead of

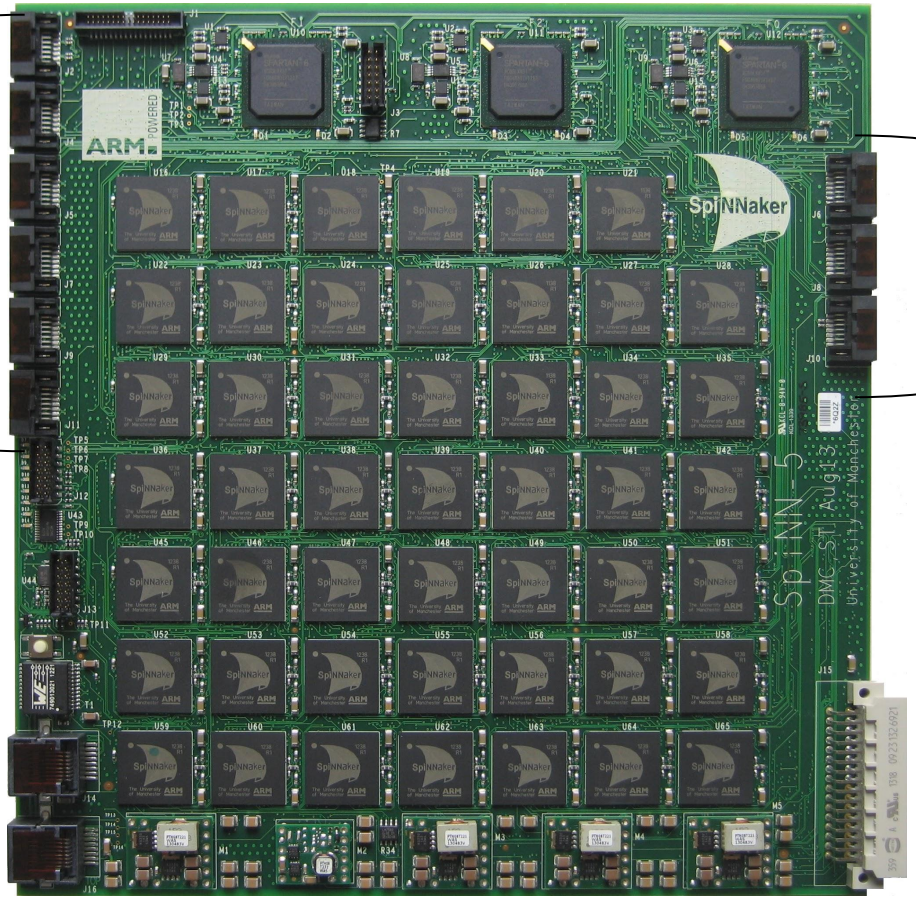
p.**Population**(retina_resolution.value.n_neurons,
p.external_devices.PushBotSpiNNakerLinkRetinaDevice(....))



MANCHESTER 1824

SATA Link connected devices!

SATA Link connected devices!



torus
SATA links

SATA
links

People who have devices

1. Bernabe Linares-Barranco
2. Jorg Conradt

How to represent this in your PyNN scripts.

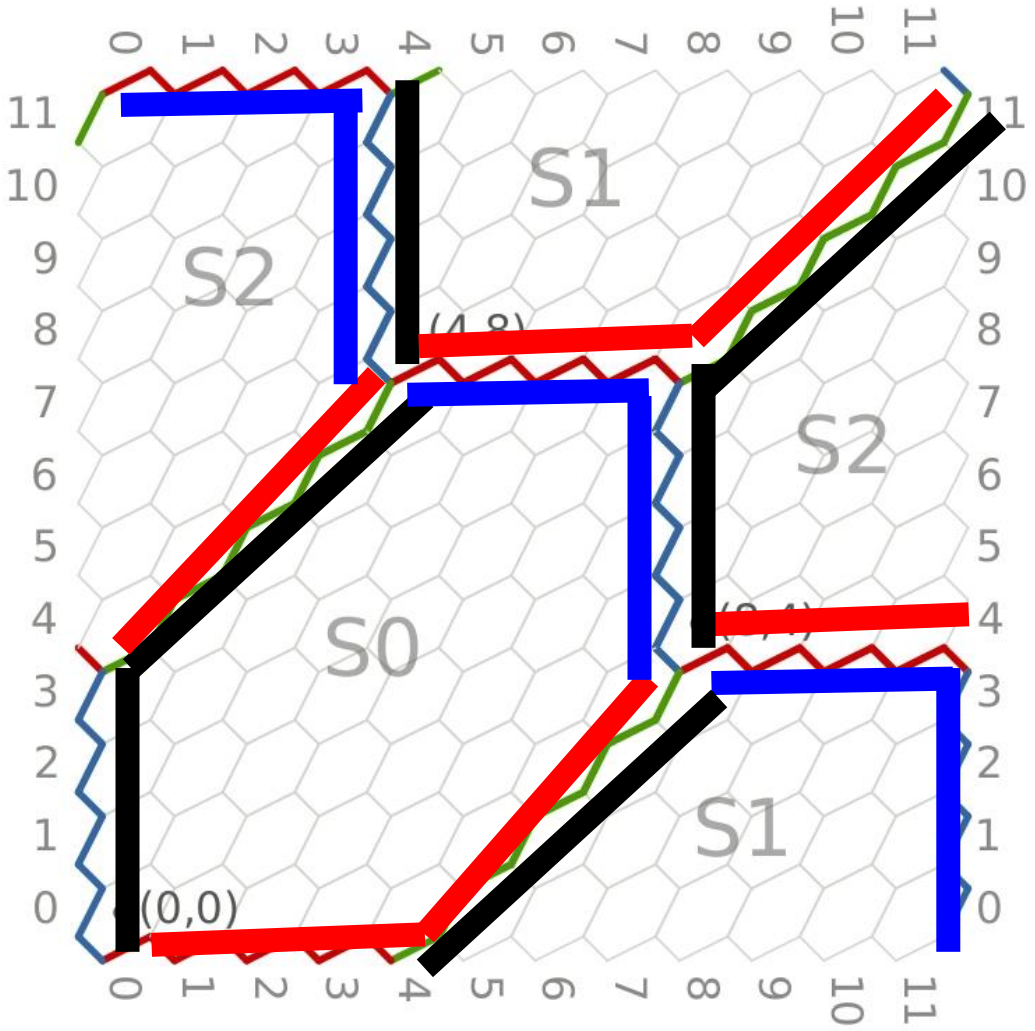
```
p.Population(
    size=2000, cell_class=p.external_devices.ArbitraryFPGADevice(
        fpga_link_id=12,
        fpga_id=1,
        board_address=None OR "192.168.0.1"))
```

How do FPGA's work in Multi-board machines?

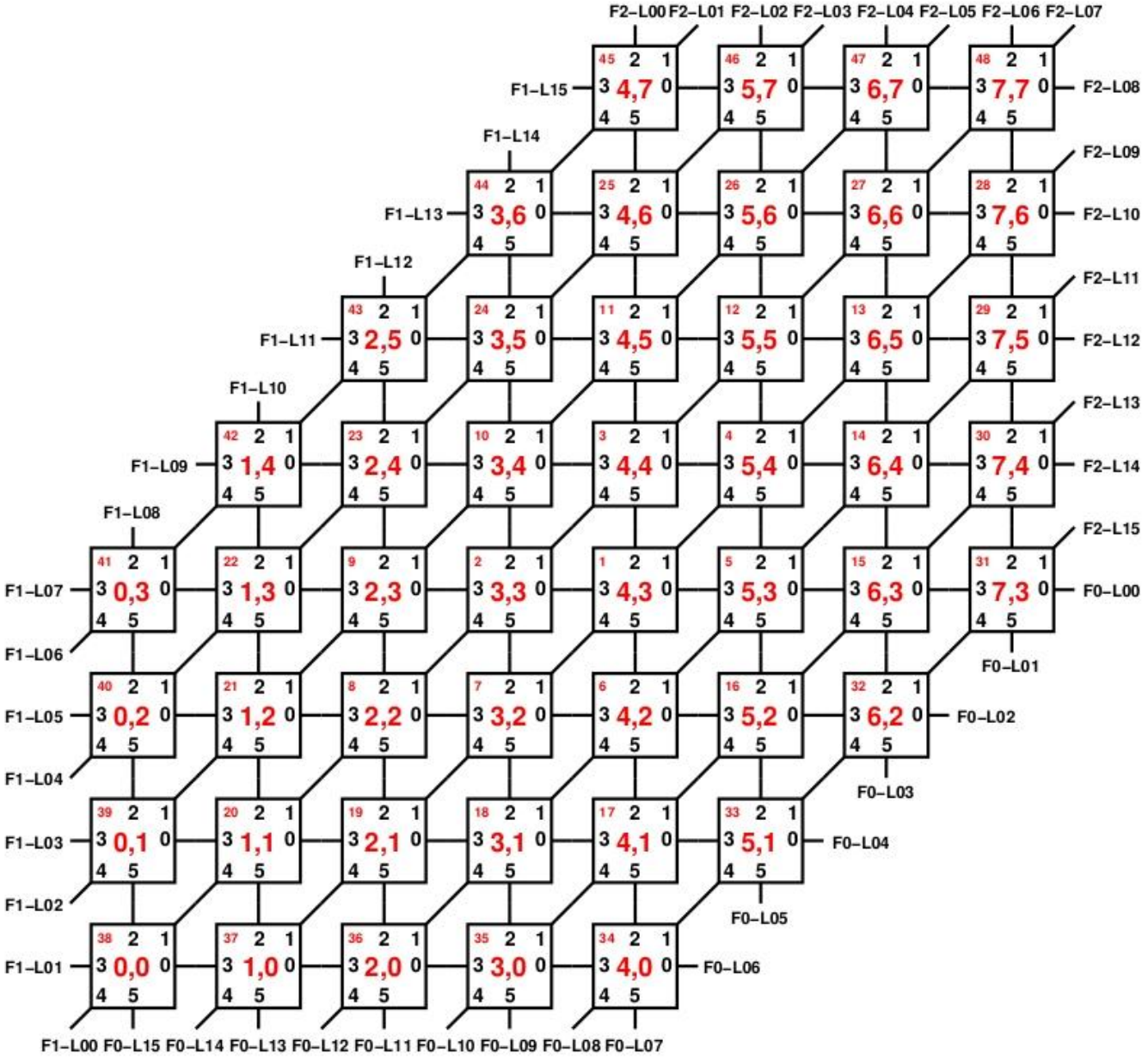
FPGA 0

FPGA 1

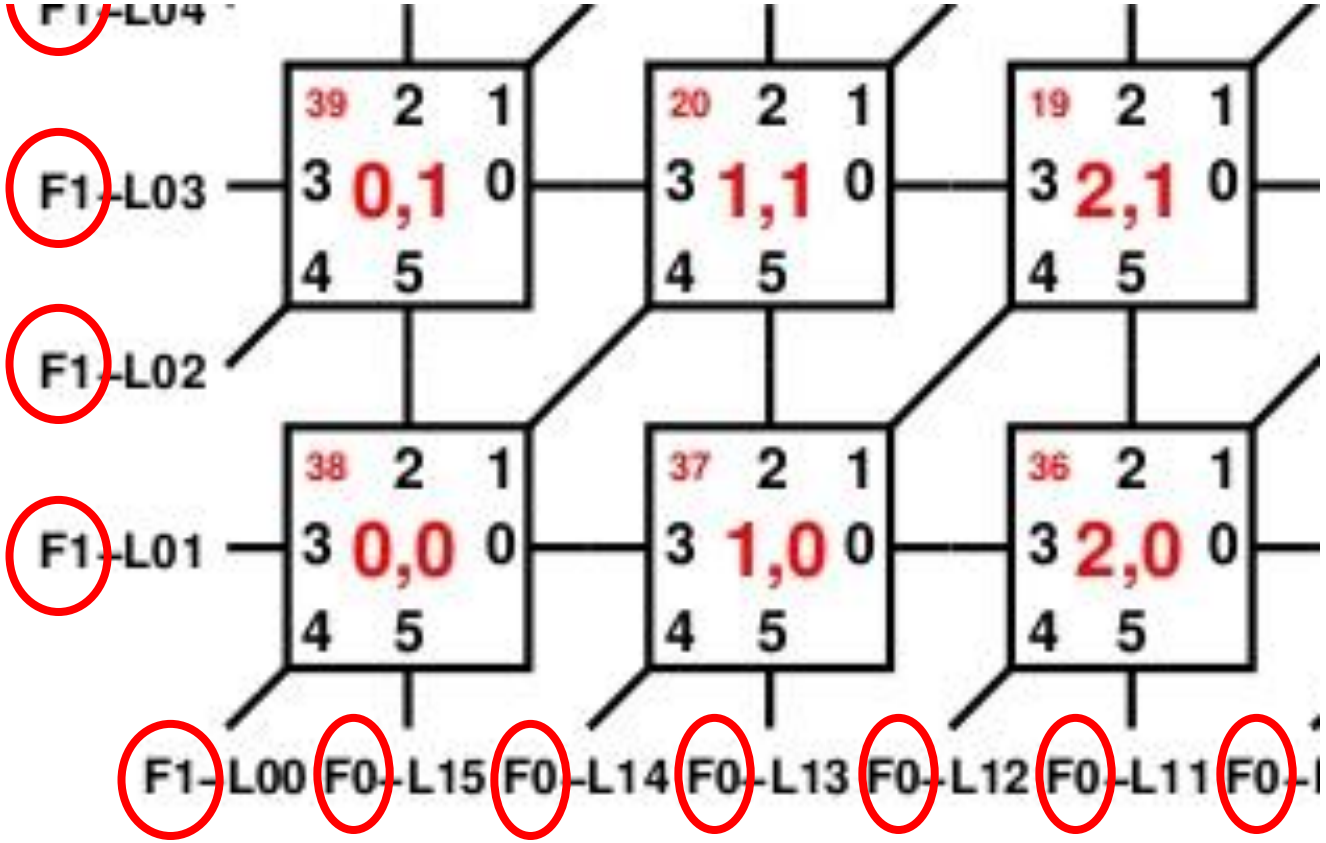
FPGA 2



What the input parameters mean?

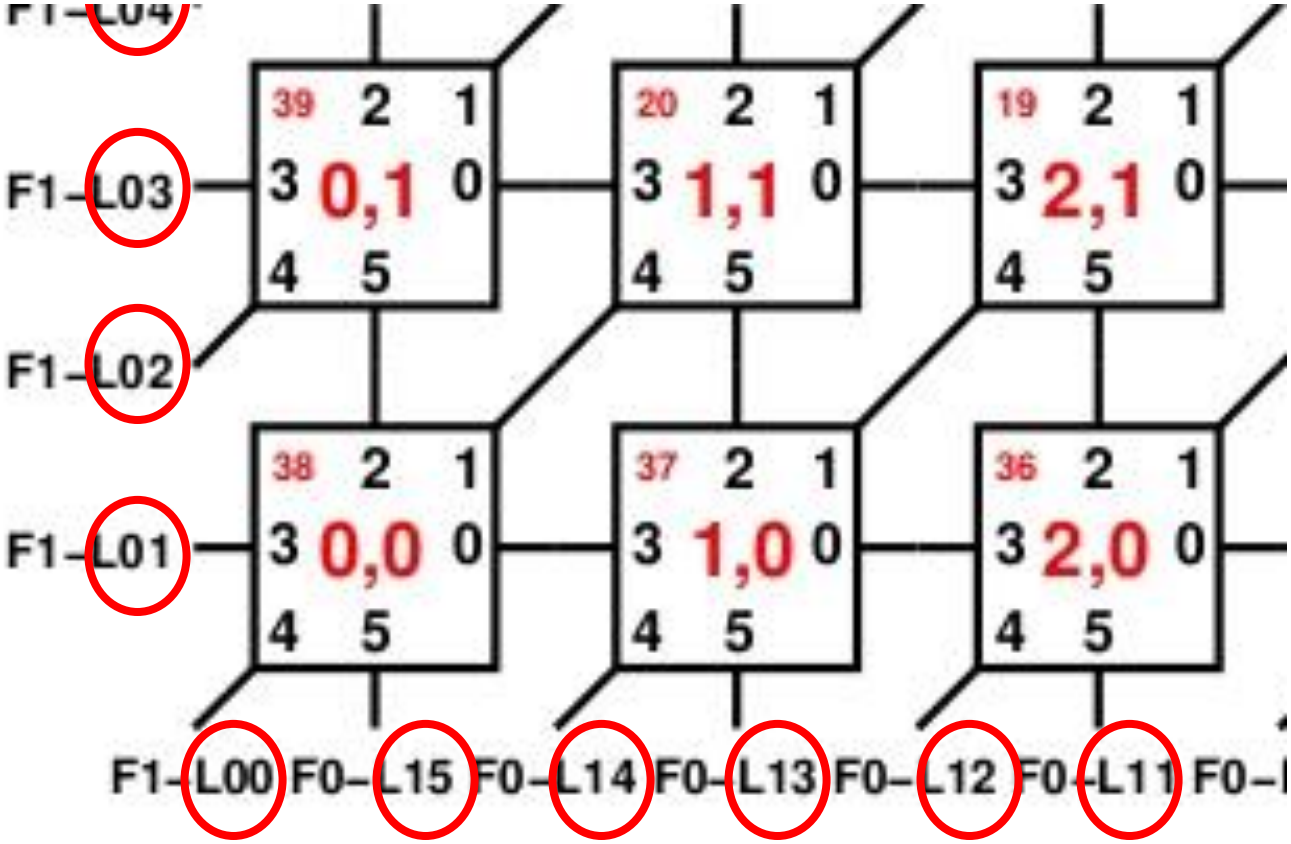


What the input parameters mean?



FPGA_ID =

What the input parameters mean?



FPGA_link_ID =

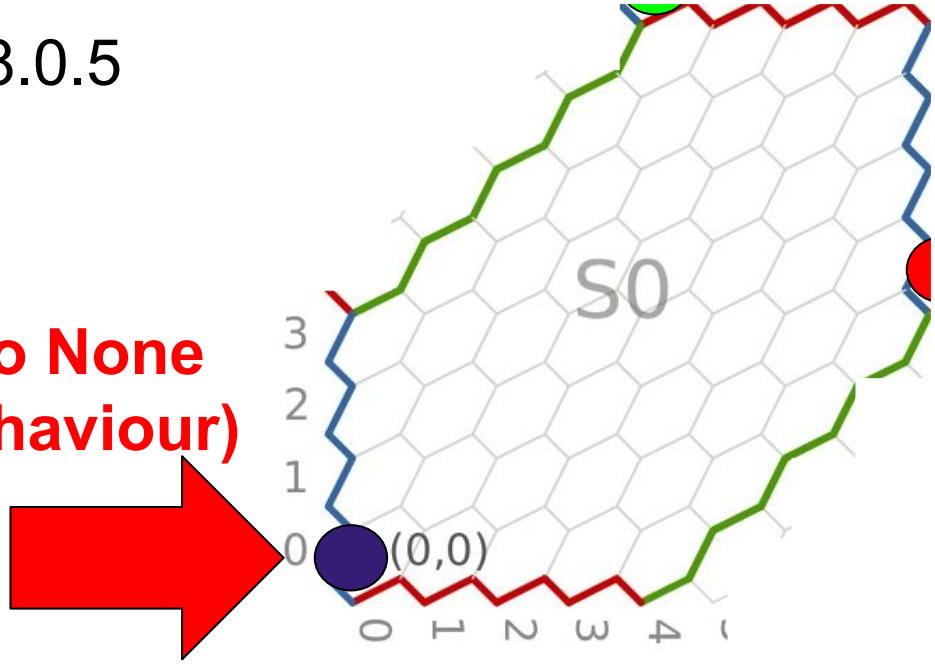
Board Address again

192.168.0.1

192.168.0.3

192.168.0.5

When set to None (default behaviour)



What you need to do to get SATA links working for your device.

1. Reprogram the FPGA's to support the communication between device and PyNN related models.

2. **The reprogramming needs to result in a disconnected edge between two chips who's communication is done through the FPGA.**

3. Extend or use the ArbitaryFPGADevice vertex to represent any extra constraints you need.

Underlying protocol 1 of 3

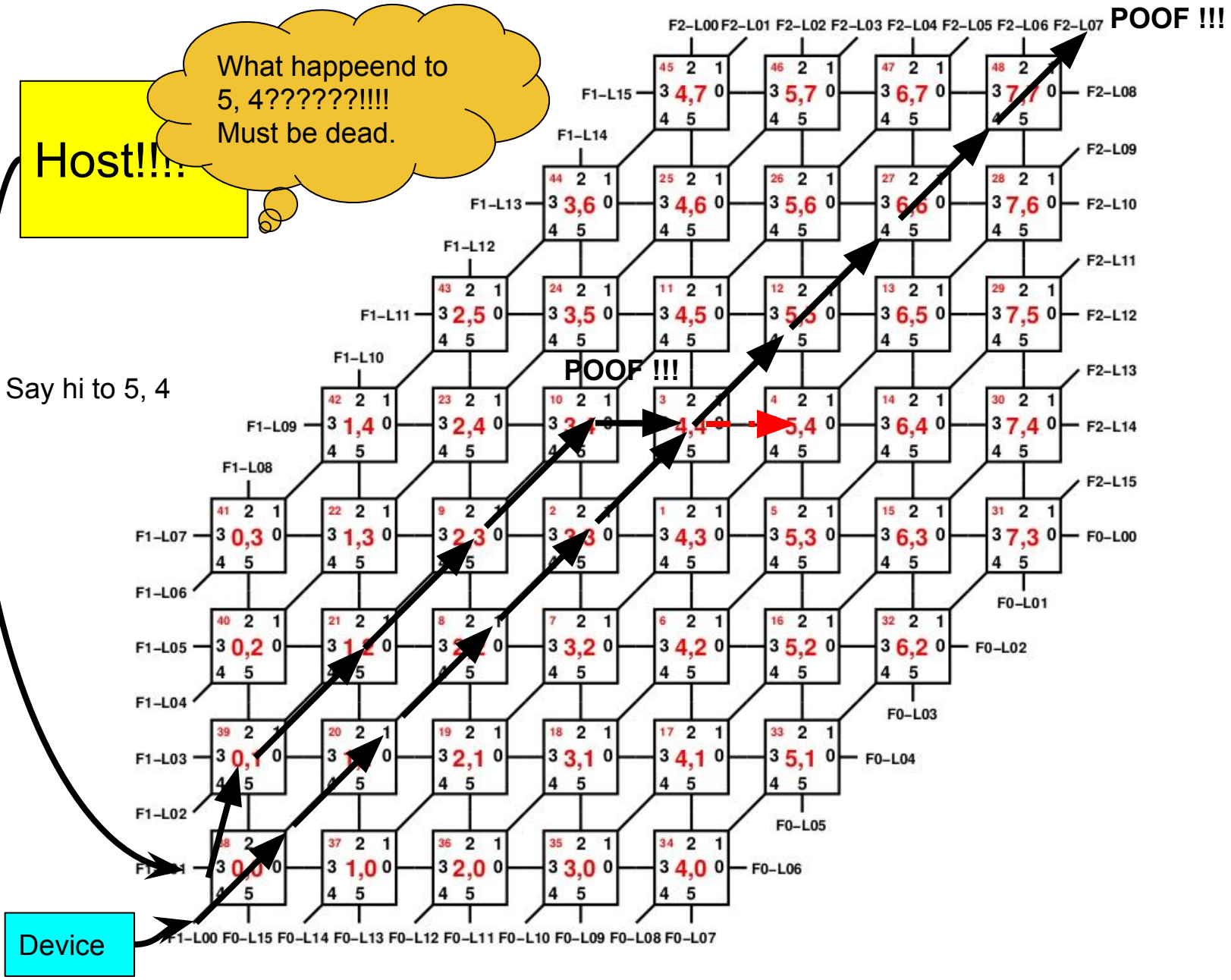
Issues:

1. How to ensure your device only sends when the simulation is running.
2. How to ensure your device stops sending when the simulation has finished running/is paused.

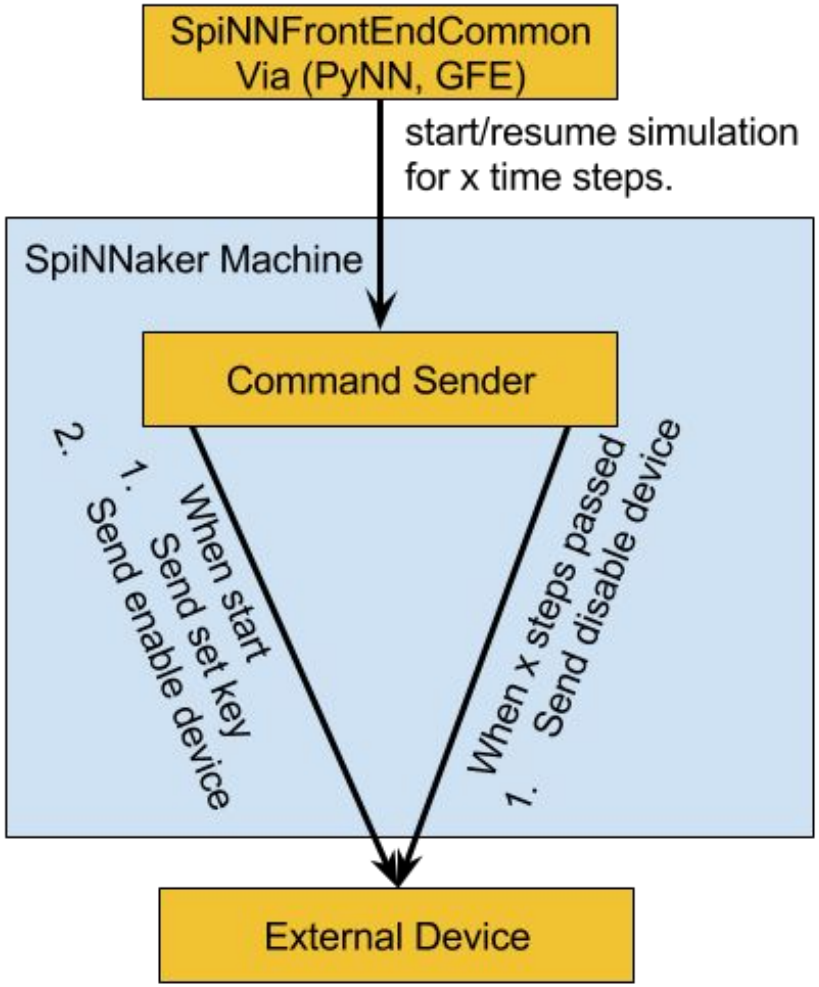
Effects from getting this wrong.

1. Boot fails to complete correctly.
2. loading fails to complete correctly.
3. Data extraction fails to complete correctly.
4. Memory management functions fail to complete correctly.

Underlying protocol 2 of 3



Underlying protocol 3 of 3



Roll your own!

Creating your own external device class

```
Class ExternalDeviceName(object)

    def __init__(self):
        pass
```

Creating your own external device class: SpiNNaker Link device

```
Class ExternalDeviceName(
        ApplicationSpiNNakerLinkVertex):

    def __init__(self, spinnaker_link_id, board_address, n_neurons, label,
                 constraints, max_atoms_per_core):
        ApplicationSpiNNakerLinkVertex.__init__(
            self, n_neurons, spinnaker_link_id, board_address, label, constraints,
            max_atoms_per_core)
```

Creating your own external device class: SpiNNaker Link device

```
Class ExternalDeviceName(
      ApplicationSpiNNakerLinkVertex, AbstractSendMeMulticastCommandsVertex):

    def __init__(spinnaker_link_id, board_address, n_neurons, label,
                 constraints, max_atoms_per_core):
        ApplicationSpiNNakerLinkVertex.__init__(
            self, n_neurons, spinnaker_link_id, board_address, label, constraints,
            max_atoms_per_core)
        AbstractSendMeMulticastCommandsVertex.__init__(self)

    def start_resume_commands(self):
        return [MultiCastCommand(key, payload, repeat, delay_between_repeats),
                MultiCastCommand(key, payload, repeat, delay_between_repeats)]
    def pause_stop_commands(self):
        return [MultiCastCommand(key, payload, repeat, delay_between_repeats)]
    def timed_commands(self):
        return []
```

Set key command

Enable command

disable command

Creating your own external device class: SpiNNaker Link device

```
Class ExternalDeviceName(
        ApplicationSpiNNakerLinkVertex, AbstractSendMeMulticastCommandsVertex,
        AbstractProvidesOutgoingPartitionConstraints):

    def __init__(......):
        ApplicationSpiNNakerLinkVertex.__init__(....)
        AbstractSendMeMulticastCommandsVertex.__init__(self)
        AbstractProvidesOutgoingPartitionConstraints.__init__(self)

    def start_resume_commands(self): .....
    .....

    def get_outgoing_partition_constraints(self, partition):
        return [FixedKeyAndMaskConstraint(
                    [BaseKeyAndMask(self._fixed_key, self._fixed_mask)])]
```

Creating your own external device class: Sata Connected device

```
Class ExternalDeviceName(object)

    def __init__(self):
        pass
```

Creating your own external device class: Sata Connected device

```
Class ExternalDeviceName(
        ApplicationFPGAVertex)

    def __init__(self, n_neurons, fpga_link_id, fpga_id, board_address, label,
                 constraints, max_atoms_per_core):
        ApplicationFPGAVertex.__init__(
            self, n_neurons, fpga_link_id, fpga_id, board_address, label,
            constraints, max_atoms_per_core)
```

Creating your own external device class: Sata Connected device

```
Class ExternalDeviceName(
        ApplicationFPGAVertex, AbstractSendMeMulticastCommandsVertex)

    def __init__(self, n_neurons, fpga_link_id, fpga_id, board_address, label,
                 constraints, max_atoms_per_core):
        ApplicationFPGAVertex.__init__(
            self, n_neurons, fpga_link_id, fpga_id, board_address, label,
            constraints, max_atoms_per_core)
        AbstractSendMeMulticastCommandsVertex.__init__(self)


    def start_resume_commands(self):
        return [MultiCastCommand(key, payload, repeat, delay_between_repeats),
                MultiCastCommand(key, payload, repeat, delay_between_repeats)]
    def pause_stop_commands(self):
        return [MultiCastCommand(key, payload, repeat, delay_between_repeats)]
    def timed_commands(self):
        return []
```

Creating your own external device class: Sata Connected device

```
Class ExternalDeviceName(
        ApplicationFPGAVertex, AbstractSendMeMulticastCommandsVertex,
        AbstractProvidesOutgoingPartitionConstraints)

    def __init__(.....):
        ApplicationFPGAVertex.__init__(....)
        AbstractSendMeMulticastCommandsVertex.__init__(self)
        AbstractProvidesOutgoingPartitionConstraints.__init__(self)

     def start_resume_commands(self):.....
     ....

     def get_outgoing_partition_constraints(self, partition):
          return [FixedKeyAndMaskConstraint(
                        [BaseKeyAndMask(self._fixed_key, self._fixed_mask)])]
```

Summary

1. Discussed External devices plugged in through the SpiNNaker Link.

2. Discussed External devices plugged in through the FPGA / SATA connector.

3. Discussed How the FPGA's interact in the communication fabric.

4. Discussed how to connect the push bot via the spinnaker link and ethernet connection.

5. Discussed underlying protocols for external devices.

6. Discussed how to build your own external device with spynnaker interfaces.