The PyNN module for the FACETS waferscale neuromorphic hardware system
pyNN.hardware.stage2 (& stage1)

Eric Müller

http://www.kip.uni-heidelberg.de/vision/
Kirchhoff Institute for Physics,
University of Heidelberg

2009-10-09
FACETS Hardware Stage 1

- 384 gLIF neurons, ~ 98k synapses
- highly accelerated operation (speed-up $10^4 - 10^5$)
- long- and short-term plasticity
- mapping biological $\leftrightarrow$ hardware value domain
- remote access, configurable neuron and synapse parameters
- measurable: spike output, membrane potential, weights

Schemmel et al.: Modeling Synaptic Plasticity within Networks of Highly Accelerated I&F Neurons, ISCAS 2007
Software status

- pyNN.hardware.stage1:
  - procedural PyNN interface
  - object-oriented PyNN interface
  - simple check of hardware constraints

- up & running

- experiments ...
Recurrent network dynamics

- Daniel Brüderle\textsuperscript{a} & Jens Kremkow\textsuperscript{b}
- compare the Stage 1 hardware system to a simulator
- systematical parameter sweep to generate different states of network activity
- e.g. this recurrent network architecture

\textsuperscript{a}University of Heidelberg
\textsuperscript{b}CNRS, Marseille, France
Firing rate

(a) Hardware: Firing Rate $\nu_{\text{net}}$

(b) NEST: Firing Rate $\nu_{\text{net}}$

Stage 1

Negst
Irregularity

Stage 1

(c) Hardware: Irregularity $CV^2$

(d) NEST: Irregularity $CV^2$
Recurrence network dynamics

Synchrony

Stage 1

(e) Hardware: Synchrony CC<sub>Sync</sub>

(f) NEST: Synchrony CC<sub>Sync</sub>
Self-Stabilizing Network Architectures

Johannes Bill\textsuperscript{a} & Klaus Schuch\textsuperscript{b}
inspired by Sussillo et al., 2007
excitatory population $P_e$
comprising $N_e = 144$ neurons
inhibitory population $P_i$
consisting of $N_i = 48$ neurons
interconnected via depressing and facilitating synapses – see schematic

\textsuperscript{a}University of Heidelberg/TU Graz
\textsuperscript{b}TU Graz
Self-Stabilizing Network Architectures

**HW**

(a) Static synapses.

(b) Dynamic synapses.

(c) Case C: Internal synapses static and strong.

(d) Case D: Internal synapses dynamic and strong.

**PC-SIM**

**Experiments**

- [ ]

**Goals**

- [ ]

- [ ]

- []

- [ ]

- [ ]

- [ ]

- [ ]

- [ ]

- [ ]

- [ ]

- [ ]
Goals

- comparing simulators and real hardware implementation
  - chip specification, debugging
- cooperation with external users
  - non-expert usability \(\Rightarrow\) PyNN
FACETS Hardware Stage 2

- first HICANN\(^a\) chip prototype produced
  - \(\sim 110\,k\) hw-synapses,
  - \(\leq 512\) AdEx neurons
  - \#inputs \(\times\) \#neurons = \#hwsynapses
- long- and short-term plasticity
- accelerated operation (speed-up \(10^4\))

- wafer system (2010?): \(\sim 350\) HICANNs
- basic functionality test in preparation
- development of low level control software

\(^a\)High Input Count Analog Neural Network
**Status - pyNN.hardware.stage2**

- **first use case**: FACETS Demonstrator
  - an emulation of the real hardware system
  - low-level software connecting the hardware system and the existing simulation software stack is still missing
- **procedural PyNN-API supported, except for**
  - `record_gsyn`, `_gsyn` - hardware constraint, $g_{syn}$ not observable
  - `record_v` - hardware solution unknown (ADCs?) so far, recording via oscilloscope possible
- **basic translation from object-oriented API to low-level API working**
Bio model
Hardware model
Mapping between models
Routing

- 64 horizontal busses per HICANN row ⇒ approx. 1k per wafer
- 256 vertical busses per HICANN column ⇒ approx. 9k per wafer
- sparse switch matrices connect vertical, horizontal busses & HICANNs
- complex routing
A test of the placing and mapping steps

- dummy network example
- populations connected in a ring structure (e.g. synfire chain)
Routing after random placing
intelligent mapping before routing

NForceCluster Algorithm

- clusters neurons according to similarity criterions
- e.g. identical input sources

Figure by TUD
route after intelligent placing – NForceCluster Algorithm
Pre- & postmap analysis

- debugging mapping and routing
- measure for mapping quality

1. create representation of bio model in a flat format (PyNN script, procedural API, every connection, every parameter in a single row)
2. run mapping and routing steps (synaptic loss due to connectivity constrains)
3. create representation of the mapped hardware model
4. differences between scripts due to
   - neuron loss, synaptic loss (limited bandwidth and routing resources)
   - parameter distortion (limited parameter range, flexibility, delays)
Runtime control – event flow

Operating Interface

Mapping Software

10GBit/s FD

Digital (Layer 2)

Analog (Layer 1)
after setup, the *Configurator* controls event flow

- input- and output spikes
- runtime chip control (recording, read-out of parameters, setting parameters)
Incremental configuration in PyNN

- main problem: clustering of parameter space takes a lot of time
- placing and routing steps are slow (1 min – 1 h, depends on model)
- but parameter changes can be typically done at zero cost ⇒ parameter sweeps still possible
- some topological changes do not require rerunning the mapping steps
- we will need support for incremental configuration: in our interface:
  setup(), connect(), run(), reconnect(), run(), ...
- PyNN support for disconnect(), setting and resetting parameters of high-level objects?
Backend-specific pre-processing of data

- multi-wafer systems will generate too many spikes to analyze (or store) in real-time
- even a single wafer can’t be fully recorded
  - on-wafer bandwidth is in the range of several TB/s
  - off-wafer (FPGA) bandwidth: \(~ 50\,\text{GB/s}\)
  - host-FPGA bandwidth: 2\,\text{GB/s}
- reduce data rates – FPGA can do a lot of preprocessing
- statistical and averaged measures evaluated online
  - avg. firing rates, ISI, measures for synchrony, irreg., etc.
  - low-pass filtering of analog (e.g. sub-threshold membrane trace) data, avg. weight over time
Conclusion: Open issues

- hardware specific unit tests
- fragile make flow needs to be replaced by a better build system
  - we have to link to the GraphModel:
    - C++ and dependencies on other libraries
- create dummy PyNN module for virtual hardware (using the FACETS Demonstrator simulation code)
- lots of control software missing
- waiting for the real wafer system