Software upgrade for BrainScaleS-2 (D9.2.1 - SGA2)

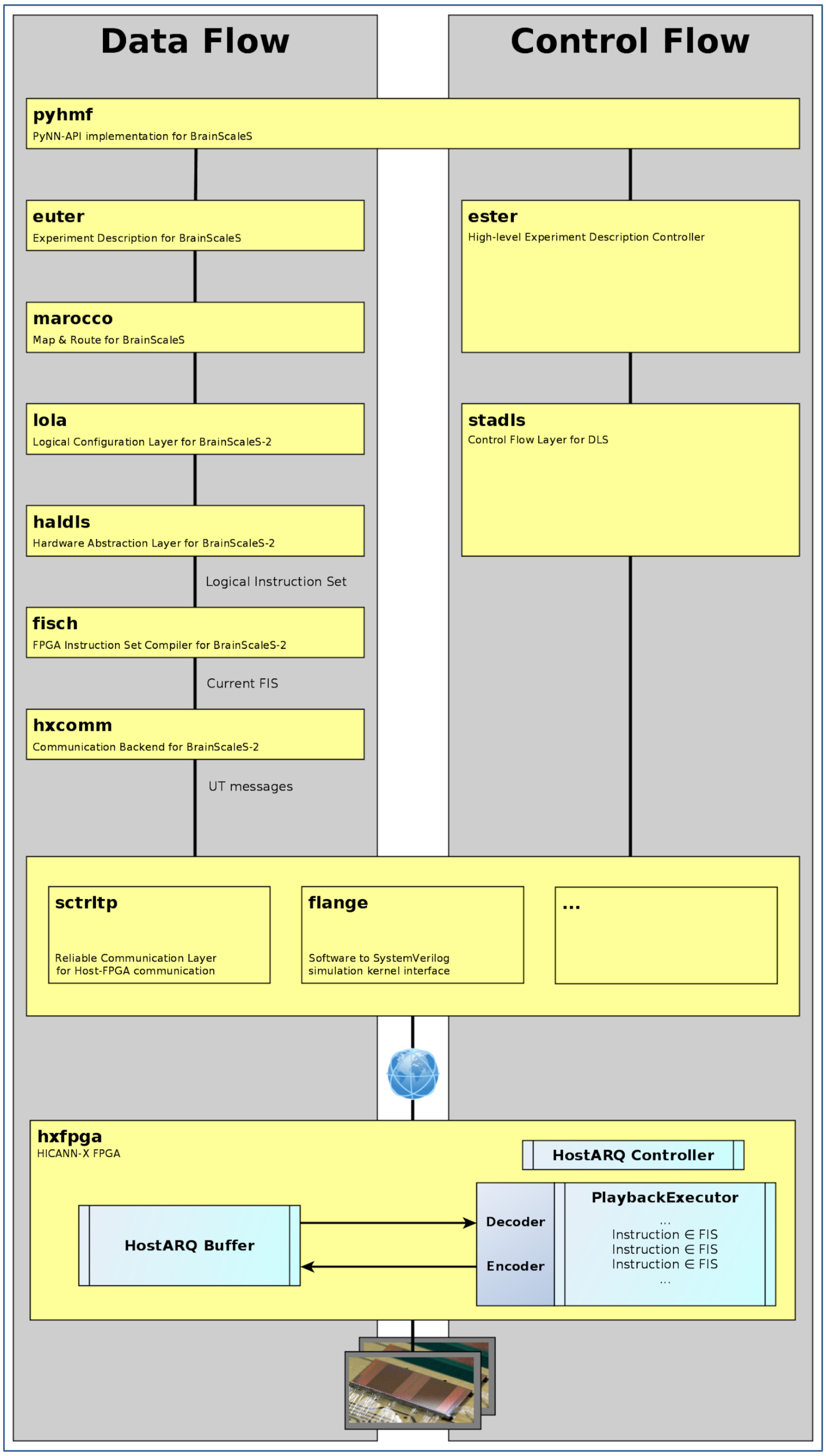

Data Flow
Control Flow
pyhmf
PyNN-API implementation for BrainScaleS
euter
Experiment Description for BrainScaleS
ester
High-level Experiment Description Controller
marocco
Map & Route for BrainScaleS
lola
Logical Configuration Layer for BrainScaleS-2
stadls
Control Flow Layer for DLS
haldls
Hardware Abstraction Layer for BrainScaleS-2
Logical Instruction Set
fisch
FPGA Instruction Set Compiler for BrainScaleS-2
Current FIS
hxcomm
Communication Backend for BrainScaleS-2
UT messages
sctrltp
Reliable Communication Layer for Host-FPGA communication
flange
Software to SystemVerilog simulation kernel interface
...
hxfpga
HICANN-X FPGA
HostARQ Controller
HostARQ Buffer
Decoder
Encoder
PlaybackExecutor
Instruction ∈ FIS
Instruction ∈ FIS
Instruction ∈ FIS

Figure 1: BrainScaleS-2 Data and Control Flow Diagram

Project Number:	785907	Project Title:	Human Brain Project SGA2

Document Title:	Software upgrade for BrainScaleS-2
Document Filename:	D9.2.1 (D58.1 D91) SGA2 M12 SUBMITTED 190325.docx
Deliverable Number:	SGA2 D9.2.1 (D58.1, D91)
Deliverable Type:	Report
Work Package(s):	WP9.2
Dissemination Level:	PU = Public
Planned Delivery Date:	SGA2 M12 / 31 Mar 2019
Actual Delivery Date:	SGA2 M12 / 25 Mar 2019; Accepted 23 Jul 2019
Authors:	Eric MÜLLER, Christian MAUCH, Philipp SPILGER - all UHEI (P47)
Compiling Editors:	
Contributors:	UHEI (P47) group members
SciTechCoord Review:	Yannick MOREL, TUM (P56)
Editorial Review:	Guy WILLIS, EPFL (P1)
Description in GA:	Software upgrade for BrainScaleS-2: Initial software upgrade of the BrainScaleS-1 software to support the BrainScaleS-2 ASIC, thereby enabling hardware-software co-simulation for the verification of the BrainScaleS-2 ASIC (Task T9.2.2)
Abstract:	The text describes the software components which have been developed to support the BrainScaleS-2 ASIC. Additional software components have been developed which allow the software stack to be linked to the hardware simulator. Hence, the ASIC's behaviour can be tested and verified in software before the hardware is produced and commissioned.
Keywords:	BrainScaleS-2, Neuromorphic Computing, Test and verification
Target Users/Readers:	Computational Neuroscientists, Neuromorphic Computing Community, Platform Users, Software developers

Table of Contents

Table of Tables

Table of Figures

1. Introduction

Compared to spiking neuronal network simulators, the operation of large-scale, accelerated analogue neuromorphic hardware systems poses additional challenges. Typical neuronal simulators process a network description and create distributed hierarchical data structures which represent the user-defined topology. Libraries such as the Connection-Set Algebra (CSA) allow for efficient parallelized construction of such data structures. During experiment runtime, a message passing mechanism communicates spikes between the interconnected neurons within the distributed compute infrastructure. However, the underlying hardware infrastructure —processors and data exchange networks— are typically abstracted away. On the other hand, large-scale neuromorphic hardware has to handle such constraints as the neurons and synapses are emulated by physical entities. User-defined neurons and synapses have to be mapped to the neuromorphic hardware substrate, the parameters have to be translated from biological to the hardware domain. In addition to the initial configuration comprised of neurons, synapses and their parameters, the experiment protocol introduces experiment-runtime dependencies. For example, spike sources have to be enabled or disabled at specific times, and firing rates have to be modulated. In the BrainScaleS systems, this experiment "protocol" handling is performed by a real-time capable controller running on the FPGA or, in the case of the BrainScaleS-2 system, also by the embedded processor.

In the following sections we describe the software parts which have been developed to support the BrainScaleS-2 ASIC. Additional software components have been developed which allow for linking the software stack to the hardware simulator. Hence, the ASIC's behaviour can be tested and verified in software before the hardware is produced and commissioned. The cover image (Figure 1) shows the main components of the complete BrainScaleS-2 Software Stack; the high-level layers are shared with the BrainScaleS-1 system.

2. BrainScaleS-2 Software Interface

The BrainScaleS-2 (BSS-2) software ecosystem builds upon the existing BrainScaleS-1 (BSS-1) platform. As a result of the advances in the hardware architecture, software components have to be adapted to the new system. When comparing BSS-2 to BSS-1 from a user's point of view, two new features stand out: programmable plasticity and structured neurons. From an algorithmic point of view, these also present the main challenges: user-defined structured neurons have to be mapped to the hardware substrate, and the description of local plasticity has to be translated into code for the embedded processors (Plasticity Processor Unit or PPU). However, for chip testing and behavioural verification in simulation and during the commissioning phase, the lower software layers are the highest priority. In particular, the chip configuration and experiment execution capabilities are essential. Hence, the initial BSS-2 software release focuses on the low-level software layers. It covers chip components, such as the configuration of neuron circuits, synapses, their parameters and the embedded processor, as well as the experiment execution protocol, i.e. the timed experiment control, as well as recording of input and output data streams, mostly spikes.

2.1 Component Overview

At the time of writing, the BSS-2 software stack already provides a complete hardware configuration interface and an experiment execution protocol abstraction layer for the earlier, small prototype chips. It has been successfully employed for peer-reviewed experiments- (currently awaiting publication; a pre-print can be found on arXiv: https://arxiv.org/abs/1811.03618). While the latest HICANN-X is not yet in the commissioning phase, it can already be accessed in simulation.

The individual components, denoted by 'single inverted commas' in this section, are described in detail in a subsequent section. The simulated hardware components, FPGA as well as the digital components of the BSS-2 ASIC, are interfaced and linked to the software stack via the 'hxcomm' and 'flange' layers. The chip configuration is described using a dedicated coordinate system, 'halco', and associated configuration containers (in ‘haldls’ and ‘fisch’). Coordinates provide a type-safe addressing scheme for the hardware components (e.g. the 5th neuron circuit from the left on the lower chip hemisphere); containers describe individual hardware units, either configurable or readable (e.g. the on-chip memory of the embedded processors, the PLL-based clock configuration or neuron spike rate counters). The configuration is stored in so-called “playback programmes”; they comprise timed write and read instructions (e.g. write X to Y at time Z), as well as timed spike events. The playback programme data structures representing the chip configuration, as well as the experiment protocol, are translated into an efficient FPGA bitstream representation and sent to the FPGA. During playback, all gathered results and recorded spike outputs are stored into a dedicated “trace” memory. The memory is read by the software, parsed and sorted into individual data streams (e.g. spikes and read answers). In the case of the earlier prototype chips, a pre-existing USB-based interface is used for host-FPGA communication. With the latest BrainScaleS-2 ASIC, the same function can be assured by reusing the communication layer of the BSS-1 system and the required software component (‘sctrltp’) has been integrated into the BSS-2 software stack.

2.2 Components

The higher software layers are listed for the sake of completeness. As these components have already been developed for the BSS-1 system, the main BSS-2 development effort therefore focused on the lower software layers specific for the BSS-2 ASIC (everything below ‘marocco’).

PyNN for BrainScaleS (‘pyhmf’): A PyNN API implementation for the BrainScaleS systems. It is an adapter between the upstream PyNN API and the C++ experiment representation layer. The PyNN API is currently being extended to support structured neurons and flexible, programmable and structural plasticity which constitutes a defining feature of the BSS-2 ASIC.

Spiking Neural Network Topology and Experiment Protocol Description (‘euter’ + ‘ester’): A C++ representation of the user-defined neural network topology, consisting of neurons, synapses, spike sources and the experiment protocol. All entities are parametrised in biological model units (e.g. a neuron’s tau_mem in milliseconds).

Map & Route (‘marocco’): A software layer containing algorithms to place neurons and synapses, route connections, and translate model parameters based on hardware constraints. It also provides an extensive API to parametrise the processes as well as individual algorithm behaviour. The result is a data structure comprised of neurons, synapses and routing information. All elements are parametrized in the hardware parameter domain. Subsequent software layers perform a 1-to-1 translation of this data into a hardware configuration.

Hardware Coordinates (‘halco’): This element provides abstract coordinates for the various hardware components. Many components are present in large numbers due to the parallel nature of neuromorphic hardware. The myriad components have varying dimensions and sizes. To ease software design, a type-safe indexing framework was developed for the BSS-1 system. This framework was reused to provide coordinates for the new BSS-2 systems.

Hardware Logical Configuration Containers (‘lola’): A planned wrapper API for BSS-2-specific ‘haldls’ configuration containers, to provide more abstract “logical” units, e.g. multiple neuron circuits are combined into structured neurons. The containers will be filled by the map & route layer.

Hardware Configuration Containers (‘haldls’): This encapsulates the complete hardware configuration in abstract units, hereinafter called containers. Each container represents the configuration for each hardware component that can be accessed independently, e.g. synapses, analogue neuron parameters or memory words on the embedded processor. Containers also provide decode and encode functions to and from the low level bit configuration. These containers can then be written or read back at specific times, via a so called playback programme. This playback programme is a timed sequence of configuration write/read instructions and spike events, which are to be executed on the FPGA. The programme builder provided in ‘haldls’ is mainly a wrapper around the programme builder implemented in the ‘fisch’ layer. All configuration containers for the earlier

BSS-2 prototype chips have been completed (see Section 2.3 for a code example). At the time of writing, not all containers for the latest ASIC (HICANN-X) have been completed. However, using the chip simulator-software interface described below, we project a significant increase in implemented containers between now and the end of SGA2 Year 1; a proof-of-principle API proposal has been developed for describing structured neurons:

```
neuron = MCNeuron()
soma = neuron.add_node(1, "soma")
ca = neuron.add_node(1, "ca")
# NOTE: Mapping this node to a wire segment
#       is not supported in this example
dummy = neuron.add_node(1, "dummy")
distal1 = neuron.add_node(1, "d1")
distal2 = neuron.add_node(1, "d2")
proximal1 = neuron.add_node(1, "p1")
proximal2 = neuron.add_node(1, "p2")
neuron.add_resistor(soma, ca)
neuron.add_resistor(ca, dummy)
neuron.add_resistor(dummy, distal1)
neuron.add_resistor(dummy, distal2)
neuron.add_resistor(soma, proximal1)
neuron.add_resistor(soma, proximal2)
```

Hardware Experiment Control ('stadls'): This layer provides run-time control for an experiment that runs on a BSS-2 prototype setup. It transfers a filled playback programme to the FPGA, triggers execution and reads back results after the hardware run. For the existing prototype setups, there are two different modes, local and "quiggeldy". The first mode is intended for use when the setup is directly connected to the host on which the user code is executed. The "quiggeldy" mode uses the SLURM resource manager to dispatch only the hardware execution part to a remote shared host, which allows for denser hardware utilisation in multi-user environments.

FPGA Instruction Set Architecture Abstraction ('fisch'): This provides a stable virtual FPGA instruction set architecture interface to the higher software layer, 'haldls'. It consists of an abstraction of the implemented FPGA instruction set, an in-software compensation for yet-to-be-implemented FPGA features and a playback programme builder pattern. Each instruction is represented by a container which stores instruction properties. An instruction can offer read and/or write access to container properties. A container's associated location is specified by a 'halco' coordinate. The playback programme builder pattern allows pre-compilation of a linear sequence of to-be-executed instructions. Instructions generating data responses, called "read instructions" offer access to (yet-to-be) acquired data via tickets, providing an interface similar to std::future. Response data is not guaranteed to stay in the original order across different origins. The playback programme is supposed to provide in-order response data access transparently for the individual origins, e.g. spike data or read responses.

FPGA (HICANN-X) Communication ('hxcomm'): This provides encoding of abstract FPGA instructions to FPGA-specific messages. A message is a variably-sized object storing the message type, called "header", and corresponding data, called "payload". An ensemble of instruction types is called a "dictionary". Different instruction dictionaries for host-to-FPGA and FPGA-to-host communication allow different data types to be communicated, depending on the orientation. The supported message instruction set can be extended by altering the corresponding dictionary, which makes it possible to support new FPGA features. In addition, 'hxcomm' handles the formatting of a FPGA message stream for fixed-width, transport-layer communication via HostARQ to hardware or similarly to a simulation backend. A control flow for each backend provides an “add”, “commit” and “receive” interface. Messages are added to a to-be-sent queue. Queue content is committed to the backend, i.e. hardware or simulation via “commit” and messages are received via “receive”. Interchangeable control flow interfaces allow simultaneous verification on hardware and in simulation.

Transport Layer (‘rw_api/vmodule’): A USB-based communication layer for block-based data exchange with FPGAs on early BSS-2 prototype setups.

Transport Layer (‘sctrltp’): An Ethernet-based custom transport protocol, similar to TCP. Originally developed for pre-BSS-1 systems, this was considerably modified and enhanced for the BSS-1 system. It has now been reused for the latest BSS-2 Ethernet-based setups. The protocol provides a reliable communication channel between the host computer and the FPGA. It implements a simple and FPGA-resource-efficient packet-based sliding-window protocol.

Hardware Simulator to Software Interface (‘flange’): This implements a remote procedure, call-based interface (based on the RCF C++ library) to the simulator, with tight coupling using SystemVerilog’s DPI mechanism. Every simulator clock cycle, data can be exchanged with the ‘flange’ software: instructions or spike data can be sent to the simulated FPGA/chip, or answers can be received from the FPGA. A concurrently running thread buffers input and output data in queues and manages the data exchange with the ‘hxcomm’ layer.

Hardware Top Level Simulation: The hardware simulator provides a cycle-accurate simulation of the digital FPGA as well as chip components. The main communication partner of the software stack is a streaming processor which builds upon the work discussed in [7]. It is used on the FPGA of the latest BSS-2 setup to send pre-buffered data to the BSS-2 ASIC with timed release. An improvement over the previous solution (employed in previous BSS-2 prototype setups) is the introduction of a parameterisable tokeniser that is also used on the latest BSS-2 ASIC (HICANN-X). It allows the definition of an instruction set that is independent from the underlying encoding, which greatly simplifies development. Event- and configuration data received from HICANN-X are annotated with timestamps and automatically encoded into a bitstream that is then processed by the host. Preliminary studies indicate a sustained event rate of up to 250MHz real time (250kHz bio) full-duplex to the HICANN-X.

PPU Software Environment: Programs written in C or C++ are cross-compiled for the PPU’s PowerPC™ architecture by an extended gcc —‘gcc-nux’— with support for the custom vector unit [1] and [4]. Recent test-driving of gcc 8.2 enables new C++17 features. The C++ standard library, libstdc++, is available via ‘newlib’ as libc. While some parts of C++'s STL work well, usage of advanced libstdc++ features is mainly constrained by the limited 16kB code size on the first prototype chips (HICANN-X allows for FPGA-backed DRAM access which relaxes this limitation). Partial interactive remote debugging using the GNU debugger has been implemented in [1] and [2]. Time-critical programme sections, e.g. the inner loop of a plasticity algorithm, are written in inline assembler. Currently, a limited number of helper functions and classes geared towards real-time application and plasticity algorithms are available, e.g., an earliest-deadline-first scheduler, stochastic synaptic weight container, getter and setter for spike-rate counters or per-synapse execution masking. For seamless migration from host to PPU code, future developments aim for direct cross-compilation of the ‘haldls’ container-based hardware abstraction layer. For larger plasticity experiments, code generation will become an essential part of the software. The first steps have been made: parametrised plasticity code templates have been developed and can be mapped to hardware.

2.3 Code Example for ‘haldls’ & ‘stadls’

cf. https://github.com/electronicvisions/haldls/blob/master/tests/hw/stadls/v2/test-hello-world.cpp

```
// configure off-chip parameters (reference currents, etc.)
Board board;
```

```
board.set_parameter(Board::Parameter::capmem_i_ref, DAC::Value(3906)); // ...

// configure 17th neuron and its synapse in the 3rd row on BSS-2
Chip chip;
auto& capmem = chip.get_capmem();
NeuronOnDLS const neuron{17};
capmem.set(neuron, NeuronParameter::v_leak, CapMemCell::Value(400));

SynapseDriverOnDLS const synapse_driver(3);
auto& syndrv_config = chip.get_synapse_drivers();
syndrv_config.set_mode(synapse_driver, SynapseDriverBlock::Mode::excitatory);

SynapseOnDLS const synapse(neuron.toSynapseColumnOnDLS(),
                        synapse_driver.toSynapseRowOnDLS());
auto& synapse_config = chip.get_synapse(synapse);

// max weight, some listening address
synapse_config.set_weight(63);
synapse_config.set_address(42);

// enable neuron's spike output
auto& neuron_config = chip.get_neuron_digital_config(neuron);
neuron_config.set_fire_out_mode(NeuronDigitalConfig::FireOutMode::enabled);

// create experiment protocol (→ regular spike train input)
PlaybackProgramBuilder builder;
size_t const offset = 1000, isi = 2000;
for (size_t ii = 0; ii < num_spikes; ++ii) {
        builder.wait_until(offset + ii * isi);
        builder.fire(synapse_driver, address);
}
builder.wait_for(offset);
builder.halt();
auto program = builder.done();

// acquire access to hardware, run experiment and read back spike data
ExperimentControl ctrl;
ctrl.run_experiment(board, chip, program);
```

auto const& spikes = program->get_spikes();

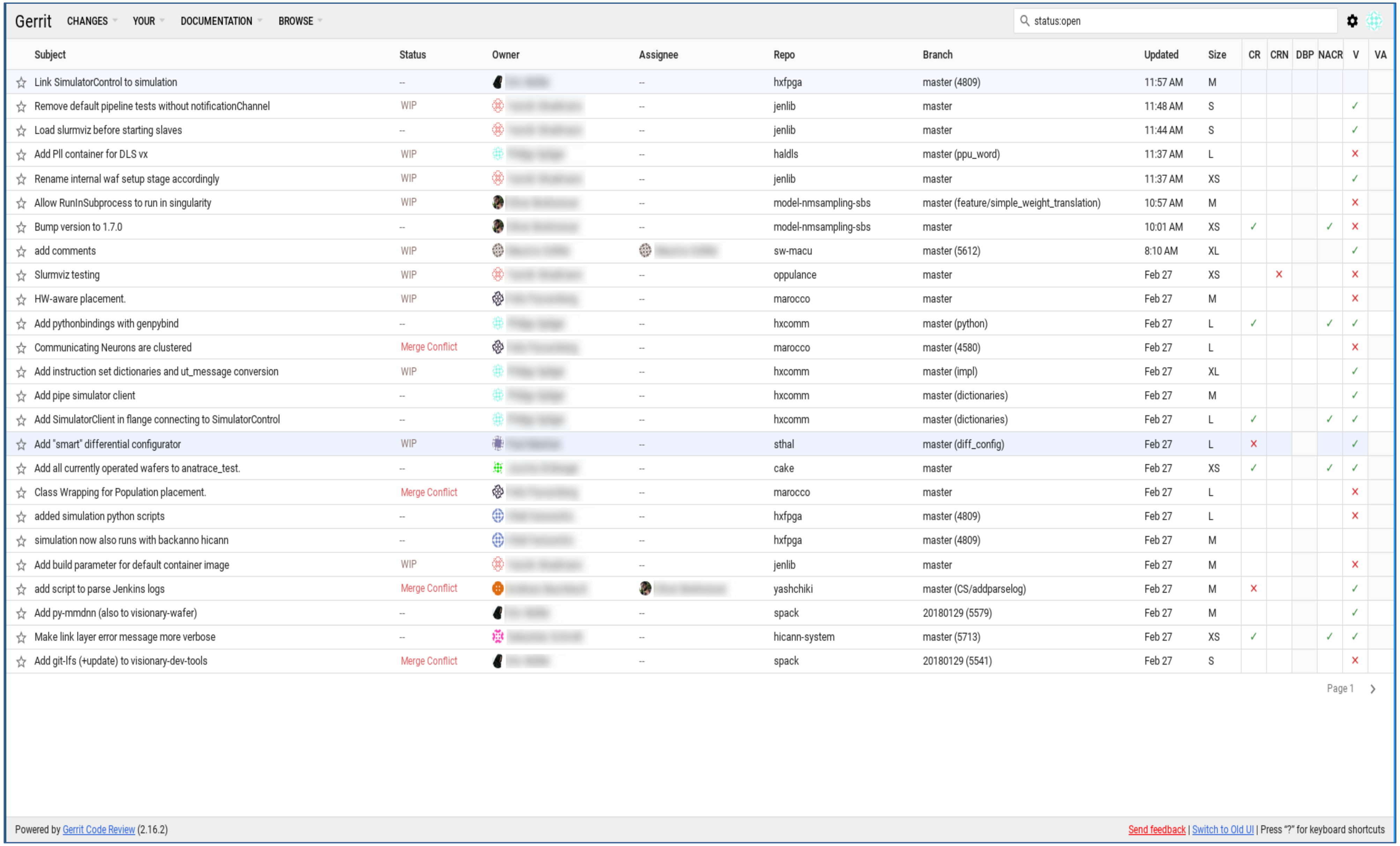
2.4 Development Process

2.4.1 Code-Review

Software development for the BrainScaleS systems started the transition to a changeset-based review process in 2012 and major parts of software development switched to the system in 2015. Over the past 7 years, all software repositories, as well as the repositories containing code for the FPGAs and the digital part of the latest BSS-2 ASIC, have migrated to the Gerrit [1]-based system. The review process requires at least one other expert to review each change and, after potentially iterating over multiple request/patch, a final approval of the other expert. In addition to the code review, a vote from the continuous integration Jenkins [2]-based system provides build and test results on a changeset-patchlevel basis. All software developments for the BSS-2 software stack have been using this code-review system. Local platform users and software designers met on a regular basis to ensure a sustainable software design.

[1] https://www.gerritcodereview.com/

[2] https://jenkins.io/



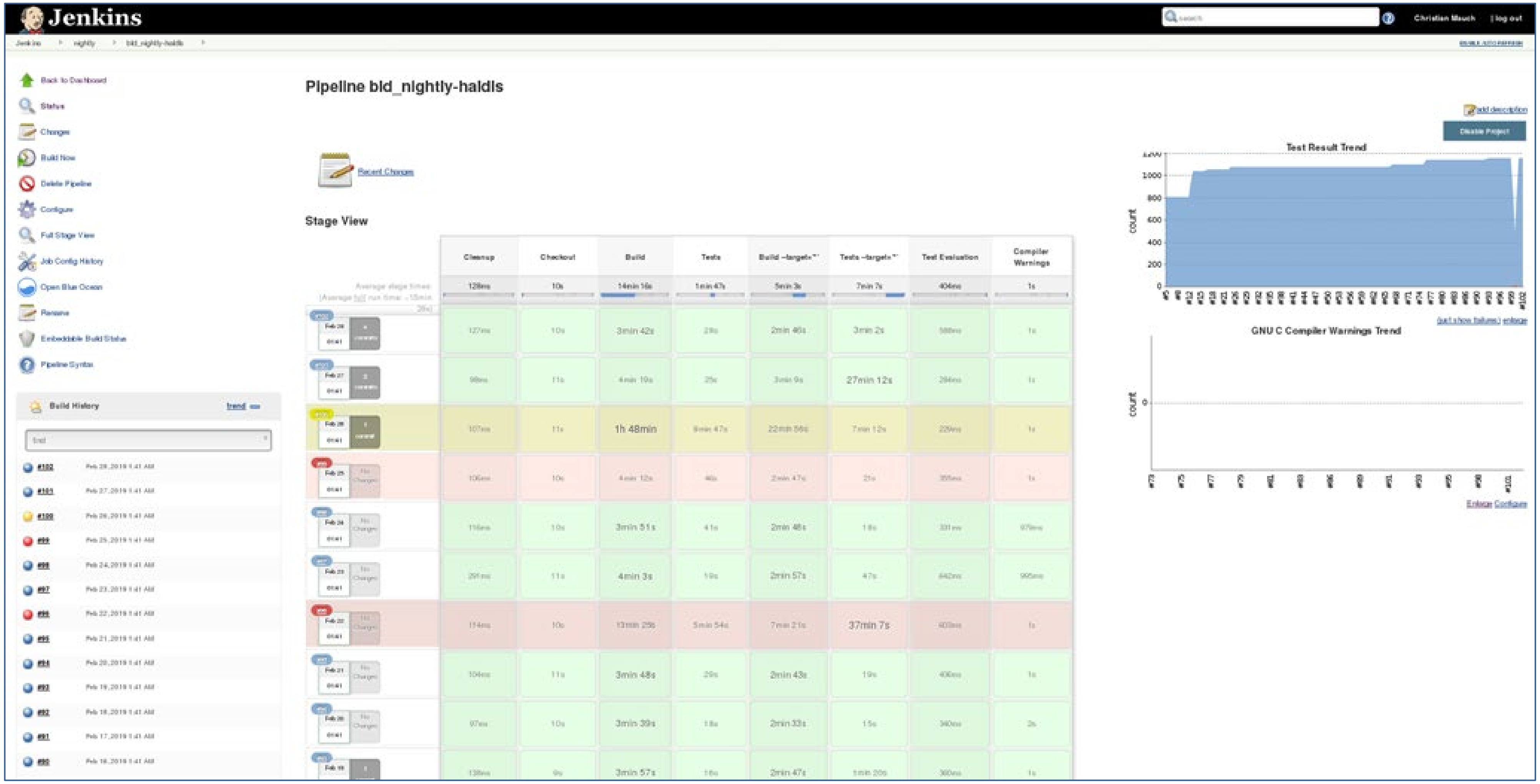
Gerrit CHANGES YOUR DOCUMENTATION BROWSE | status:open

Subject	Status	Owner	Assignee	Repo	Branch	Updated	Size	CR	CRN	DBP	NACR	V	VA
Link SimulatorControl to simulation	--	[redacted]	--	hxfpga	master (4809)	11:57 AM	M						
Remove default pipeline tests without notificationChannel	WIP	[redacted]	--	jenlib	master	11:48 AM	S					✓	
Load slurmviz before starting slaves	--	[redacted]	--	jenlib	master	11:44 AM	S					✓	
Add Pll container for DLS vx	WIP	[redacted]	--	haldls	master (ppu_word)	11:37 AM	L					✗	
Rename internal waf setup stage accordingly	WIP	[redacted]	--	jenlib	master	11:37 AM	XS					✓	
Allow RunInSubprocess to run in singularity	WIP	[redacted]	--	model-nmsampling-sbs	master (feature/simple_weight_translation)	10:57 AM	M					✗	
Bump version to 1.7.0	--	[redacted]	--	model-nmsampling-sbs	master	10:01 AM	XS	✓			✓	✗	
add comments	WIP	[redacted]	[redacted]	sw-macu	master (5612)	8:10 AM	XL					✓	
Slurmviz testing	WIP	[redacted]	--	oppulance	master	Feb 27	XS		✗			✗	
HW-aware placement.	WIP	[redacted]	--	marocco	master	Feb 27	M					✗	
Add pythonbindings with genpybind	--	[redacted]	--	hxcomm	master (python)	Feb 27	L	✓			✓	✓	
Communicating Neurons are clustered	Merge Conflict	[redacted]	--	marocco	master (4580)	Feb 27	L					✗	
Add instruction set dictionaries and ut_message conversion	WIP	[redacted]	--	hxcomm	master (impl)	Feb 27	XL					✓	
Add pipe simulator client	--	[redacted]	--	hxcomm	master (dictionaries)	Feb 27	M					✓	
Add SimulatorClient in flange connecting to SimulatorControl	--	[redacted]	--	hxcomm	master (dictionaries)	Feb 27	L	✓			✓	✓	
Add "smart" differential configurator	WIP	[redacted]	--	sthal	master (diff_config)	Feb 27	L	✗				✓	
Add all currently operated wafers to anatrace_test.	--	[redacted]	--	cake	master	Feb 27	XS	✓			✓	✓	
Class Wrapping for Population placement.	Merge Conflict	[redacted]	--	marocco	master	Feb 27	L					✗	
added simulation python scripts	--	[redacted]	--	hxfpga	master (4809)	Feb 27	L					✗	
simulation now also runs with backanno hicann	--	[redacted]	--	hxfpga	master (4809)	Feb 27	M						
Add build parameter for default container image	WIP	[redacted]	--	jenlib	master	Feb 27	M					✗	
add script to parse Jenkins logs	Merge Conflict	[redacted]	[redacted]	yashchiki	master (CS/addparselog)	Feb 27	M	✗				✓	
Add py-mmdnn (also to visionary-wafer)	--	[redacted]	--	spack	20180129 (5579)	Feb 27	M					✓	
Make link layer error message more verbose	--	[redacted]	--	hicann-system	master (5713)	Feb 27	XS	✓			✓	✓	
Add git-lfs (+update) to visionary-dev-tools	Merge Conflict	[redacted]	--	spack	20180129 (5541)	Feb 27	S					✗	

Page 1 >

Powered by Gerrit Code Review (2.16.2) | Send feedback | Switch to Old UI | Press "?" for keyboard shortcuts

Figure 2: Screenshot of BrainScaleS' software code review system (based on Gerrit)



Pipeline bld_nightly-haldls

Stage View

	Cleanup	Checkout	Build	Tests	Build --target=*'	Tests --target=*'	Test Evaluation	Compiler Warnings
Average stage times: (Average full run time: ~15min 26s)	128ms	10s	14min 16s	1min 47s	5min 3s	7min 7s	404ms	1s
#102 Feb 28 01:41	127ms	10s	3min 42s	29s	2min 46s	3min 2s	588ms	1s
#101 Feb 27 01:41	98ms	11s	4min 19s	25s	3min 9s	27min 12s	284ms	1s
#100 Feb 26 01:41	107ms	11s	1h 48min	9min 47s	22min 56s	7min 12s	229ms	1s
#99 Feb 25 01:41 No Changes	106ms	10s	4min 12s	46s	2min 47s	21s	355ms	1s
#98 Feb 24 01:41 No Changes	116ms	10s	3min 51s	41s	2min 48s	18s	331ms	979ms
#97 Feb 23 01:41 No Changes	291ms	11s	4min 3s	19s	2min 57s	47s	842ms	995ms
#96 Feb 22 01:41 No Changes	114ms	10s	13min 25s	5min 54s	7min 21s	37min 7s	403ms	1s
#95 Feb 21 01:41 No Changes	104ms	11s	3min 48s	29s	2min 43s	19s	406ms	1s
#94 Feb 20 01:41 No Changes	97ms	10s	3min 39s	18s	2min 33s	15s	340ms	2s
#93 Feb 19	138ms	9s	3min 57s	16s	2min 47s	1min 20s	360ms	1s

Build History trend

#102	Feb 28, 2019 1:41 AM
#101	Feb 27, 2019 1:41 AM
#100	Feb 26, 2019 1:41 AM
#99	Feb 25, 2019 1:41 AM
#98	Feb 24, 2019 1:41 AM
#97	Feb 23, 2019 1:41 AM
#96	Feb 22, 2019 1:41 AM
#95	Feb 21, 2019 1:41 AM
#94	Feb 20, 2019 1:41 AM
#93	Feb 19, 2019 1:41 AM
#92	Feb 18, 2019 1:41 AM
#91	Feb 17, 2019 1:41 AM
#90	Feb 16, 2019 1:41 AM

Figure 3: Screenshot of BrainScaleS-2's nightly CI and deployment job

2.4.2 Continuous Integration (CI)

Automated testing is essential to manage changes to a complex software code base. The BrainScaleS developers use Jenkins for test automation in a multitude of projects, including software for the BSS-2 system, as well as hardware-related repositories. A sample screenshot of a Jenkins dashboard can be seen in Figure 3 above.

After building and testing individual changesets in both software and hardware, the CI system deploys pre-built packages of all software elements to a distributed filesystem; the release strategy is a rolling release based on stable HEADs. These packages are identified by deployment timestamps and provided to local as well as HBP platform users. All software repository HEADs are replicated to public read-only clones on GitHub (https://github.com/electronicvisions).

2.4.3 Developer Software Environment

The Neuromorphic Platform within the HBP Collaboratory provides a means to execute experiments without requiring any additional software installation on the experimenter or user side. However, when developing the software components, a substantial software environment has to be deployed. A robust and simple solution can be provided using containerised software installations. The BrainScaleS developers track all external software dependencies in a state-of-the-art build-from-source-based package manager (https://github.com/spack/spack). Neuromorphic Platform-specific meta packages describe all individual dependencies, including possible version constraints (e.g. boost C++ library version at least 1.69.0). All BrainScaleS meta packages are installed into a container image and deployed to a central location which is also provided to external expert users. The process uses the same Continuous Response (CR) and CI mechanisms as all other BrainScaleS software components. Developers can upload changesets that will be test-built upon request, tested using the BrainScaleS software stack and, after successful verification, deployed as new latest container image.

3. Conclusion and Outlook

The BSS-2 software stack has been already successfully utilised in published and unpublished experiments [1], [3], [5], [6]. The Neuromorphic Platform provides access to a set of BSS-2 prototype setups via the usual REST API (Python library: https://github.com/HumanBrainProject/hbp-neuromorphic-client) which is also used for the BrainScaleS-1 and SpiNNaker architectures. Interactive, web UI-based demonstrators have been implemented; for example, an experiment demonstrating reinforcement learning on the second version of the BSS-2 prototype systems has been recorded: https://www.youtube.com/watch?v=LW0Y5SSIQU4.

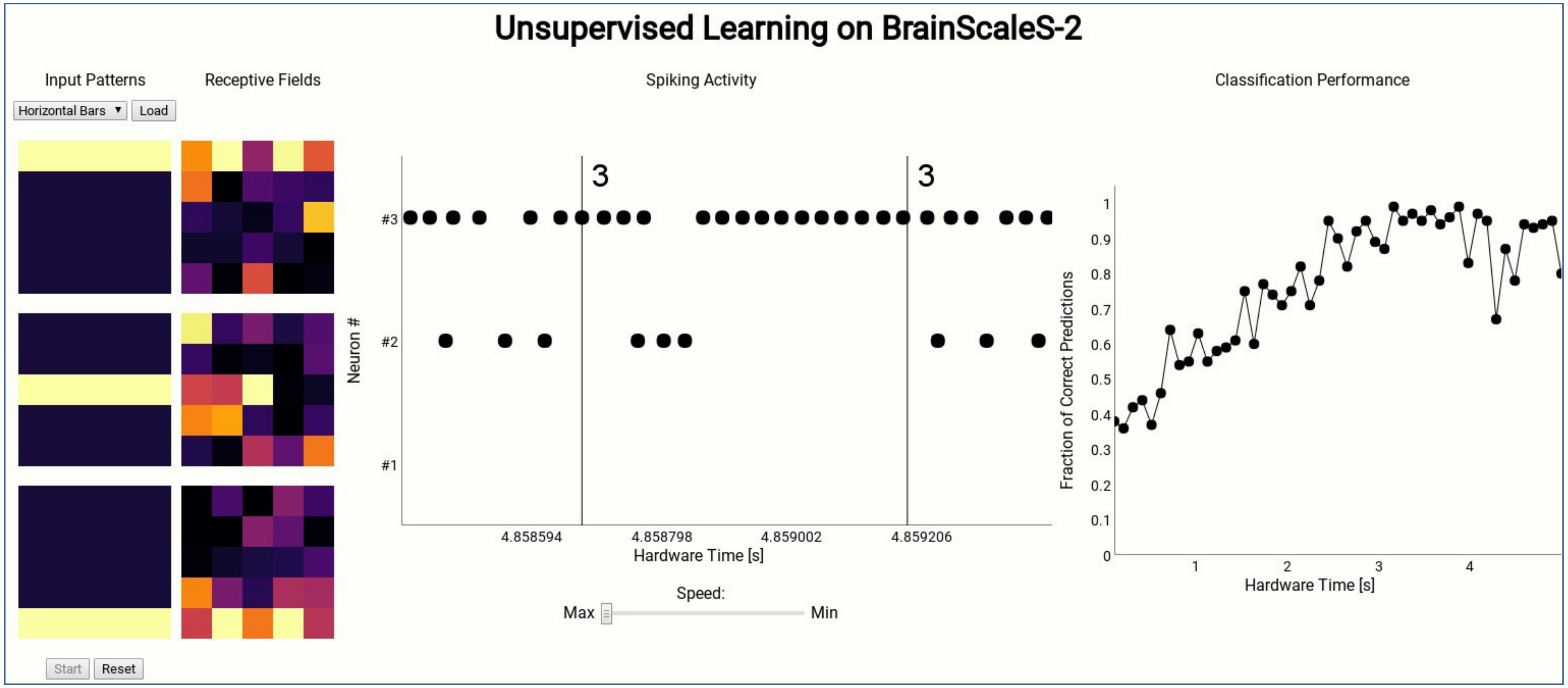


Figure 4: Web interface for an unsupervised learning experiment demonstrator running on the BrainScaleS-2 version 2 prototype chip

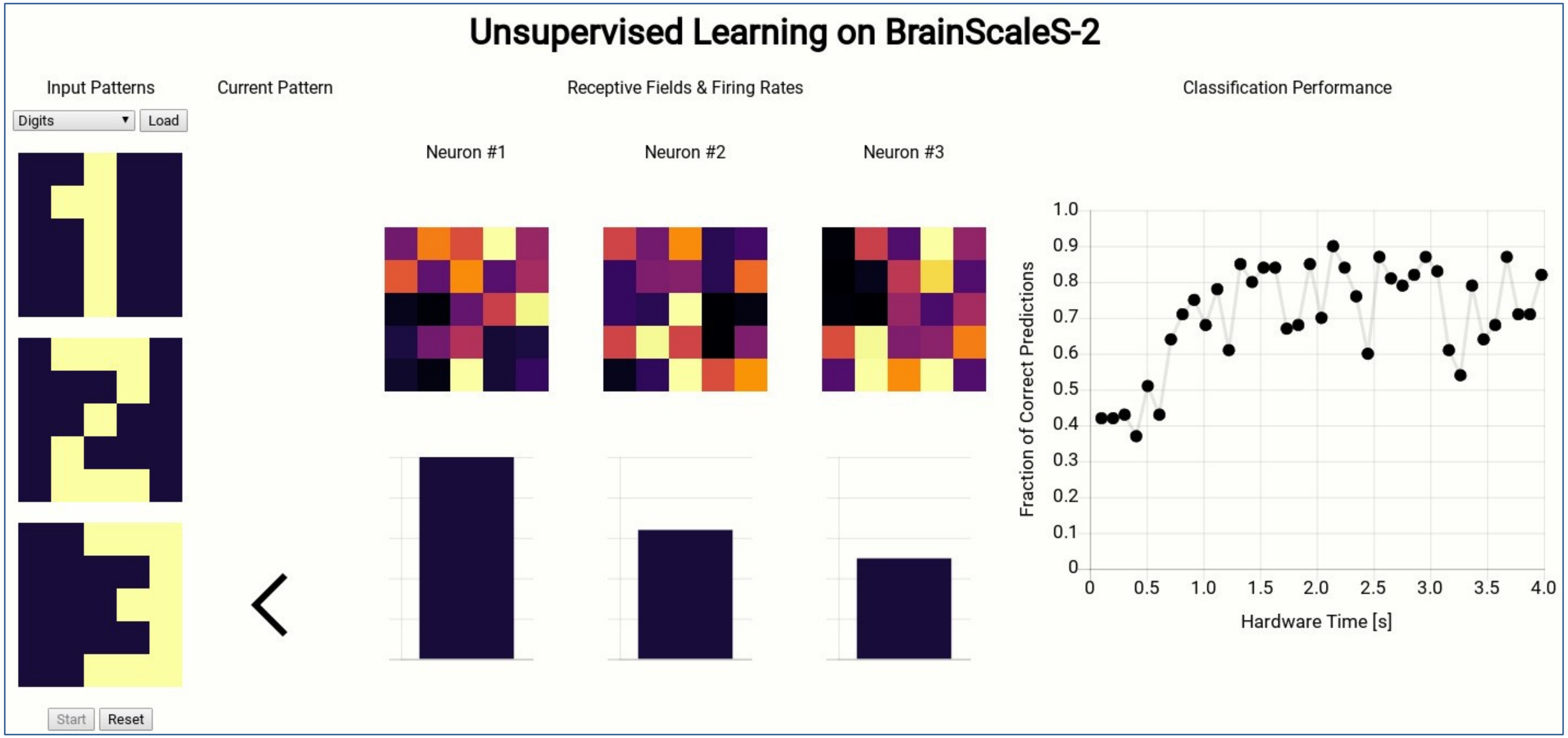


Figure 5: Web interface for an unsupervised learning experiment demonstrator running on the BrainScaleS-2 version 2 prototype chip

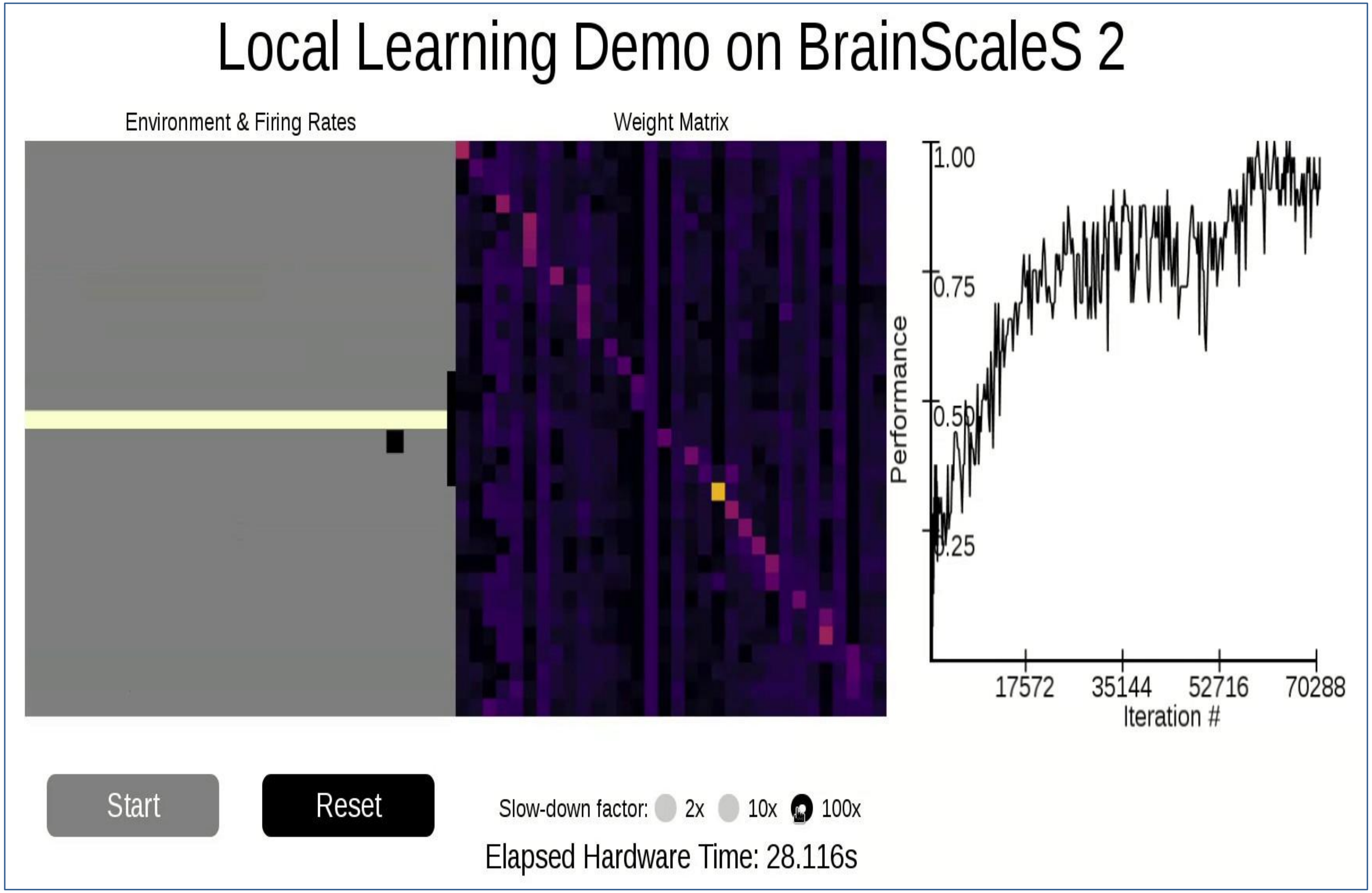

Local Learning Demo on BrainScaleS 2
Environment & Firing Rates
Weight Matrix
Performance
1.00
0.75
0.50
0.25
17572
35144
52716
70288
Iteration #
Start
Reset
Slow-down factor: 2x 10x 100x
Elapsed Hardware Time: 28.116s

Figure 6: Web interface for a local learning experiment demonstrator; the experiment combines spiking neural network operation on the BrainScaleS-2 version 2 prototype chip to a simulated environment running on the embedded processor

As the experiments suggest, the software stack provides a complete hardware configuration interface and an experiment execution protocol abstraction layer for the prototypes of the BSS-2 system. For the latest BSS-2 ASIC (HICANN-X), a bridge between the FPGA/chip simulation and the software stack has been developed. It enables early verification of the ASIC using the same software stack as in real operation.

The next major software development efforts will focus the higher software layers. The link between hardware configuration and the more abstract map & route layer will be established. This requires major changes to the 'marocco' package itself: chip-specific assumptions will be replaced by a generic approach which tracks constraints for different hardware backends (i.e. BrainScaleS-1 and BrainScaleS-2 architectures). Additionally, new features require new resource requirement calculations as, for example, structured neurons and programmable plasticity allow interesting new experiments, but also introduce constraints on the placement of neurons. The embedded processor, and programmable plasticity in particular, also require mechanisms for code generation, as every analogue chip is different and the placement will have to cope with blacklisted components, placement differences, as well as non-homogeneous parametrisation. The first steps towards code generation have been made by providing parametrised plasticity code templates. Further steps will be aligned to the PyNN API development.

4. References and Literature

Table 1: Public Software Repositories for BrainScaleS

Repository	URL	Short Description
‘pyhmf’	https://github.com/electronicvisions/pyhmf	PyNN implementation for BrainScaleS
‘euter’	https://github.com/electronicvisions/euter	C++ library for handling user-defined neural network topologies and experiment protocols
‘marocco’	https://github.com/electronicvisions/marocco	Routing and mapping of biological network description to hardware constraints
‘haldls’	https://github.com/electronicvisions/haldls	Low-level abstraction layer for HICANN-DLS-based chips
‘stadls’ (stored in ‘haldls’ repo)	https://github.com/electronicvisions/haldls	Experiment control flow
‘fisch’	https://github.com/electronicvisions/fisch	FPGA instruction interface API
‘hxcomm’	https://github.com/electronicvisions/hxcomm	FPGA instruction bit formatting
‘flange’ (stored in ‘hxcomm’ repo)	https://github.com/electronicvisions/hxcomm	Interface between ‘hxcomm’ and FPGA simulation
‘sctrltp’	https://github.com/electronicvisions/sctrltp	Ethernet-based reliable transport layer for communication between host and FPGA
‘vmodule’	https://github.com/electronicvisions/vmodule	USB-based library for data exchange between host and FPGA
‘gcc-nux’	https://github.com/electronicvisions/gcc	Compiler and linker for PPU programs
‘libnux’	https://github.com/electronicvisions/libnux	Helpers library for PPU programs
‘binutils-gdb’	https://github.com/electronicvisions/binutils-gdb	Binary utility programs of the PPU toolchain
‘newlib’	https://github.com/electronicvisions/newlib	Libc for PPU programs
Software dependencies	https://github.com/electronicvisions/spack	Tracks all external software dependencies of the BSS software stack
Container Images	https://openproject.bioai.eu/containers	Pre-built container images (for singularity) providing all external software dependencies of the BSS software environments

Table 2: Literature and References

#	URL	Short Description
[1]	http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3838	Philipp Spilger, Bachelor thesis, Spike-based Expectation Maximization on the HICANN-DLSv2 Neuromorphic Chip, 2018.
[2]	https://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3854	Timo Wunderlich, Master thesis, Demonstrating advantages of neuromorphic computation, 2019.
[3]	https://arxiv.org/abs/1811.03618	Timo Wunderlich *et al.*, Demonstrating Advantages of Neuromorphic Computation: A Pilot Study, Frontiers in Neuroscience (Neuromorphic Engineering), 2019 (to be published).
[4]	http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3448	Arthur Heimbrecht, Bachelor thesis, Compiler Support for the BrainScaleS Plasticity Processor, 2017.
[5]	https://online.tugraz.at/tug_online/wbAbs.showThesis?pThesisNr=63900&pOrgNr=2369	Thomas Bohnstingl, Master thesis, Development of an agent for solving Markov Decision Processes embedded in Spiking Neural Networks, 2018.
[6]	https://online.tugraz.at/tug_online/wbAbs.showThesis?pThesisNr=63901&pOrgNr=2369	Franz Scherr, Master thesis, Spike-Based Agents for Multi-Armed Bandits, 2018.
[7]	https://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=2952	Simon Friedmann, PhD thesis, A new approach to learning in neuromorphic hardware, 2013.