1 Main Page

GeNN is a software package to enable neuronal network simulations on NVIDIA GPUs by code generation. Models are defined in a simple C-style API and the code for running them on either GPU or CPU hardware is generated by GeNN. GeNN can also be used through external interfaces. Currently there are prototype interfaces for Spine↩ Creator and SpineML and for Brian2.

GeNN is currently developed and maintained by

James Turner (contact James)
Dr. Esin Yavuz (contact Esin)
Prof. Thomas Nowotny (contact Thomas)

The development of GeNN is partially supported by the EPSRC (grant number EP/J019690/1 - Green Brain Project).

Note

This documentation is under construction. If you cannot find what you are looking for, please contact the project developers.

Next

2 Installation

You can download GeNN either as a zip file of a stable release or a snapshot of the most recent stable version or the unstable development version using the Git version control system.

2.1 Downloading a release

Point your browser to https://github.com/genn-team/genn/releases and download a release from the list by clicking the relevant source code button. Note that GeNN is only distributed in the form of source code due to its code generation design. Binary distributions would not make sense in this framework and are not provided. After downloading continue to install GeNN as described in the Installing GeNN section below.

2.2 Obtaining a Git snapshot

If it is not yet installed on your system, download and install Git (http://git-scm.com/). Then clone the GeNN repository from Github

```
git clone https://github.com/genn-team/genn.git
```

The github url of GeNN in the command above can be copied from the HTTPS clone URL displayed on the GeNN Github page (https://github.com/genn-team/genn).

This will clone the entire repository, including all open branches. By default git will check out the master branch which contains the source version upon which the latest release is based. If you want the most recent (but unstable) development version (which may or may not be fully functional at any given time), checkout the development branch

```
git checkout development
```

There are other branches in the repository that are used for specific development purposes and are opened and closed without warning.

As an alternative to using git you can also download the full content of GeNN sources clicking on the "Download ZIP" button on the bottom right of the GeNN Github page (https://github.com/genn-team/genn).

2.3 Installing GeNN

Installing GeNN comprises a few simple steps to create the GeNN development environment.

(i) If you have downloaded a zip file, unpack GeNN.zip in a convenient location. Otherwise enter the directory where you downloaded the Git repository.

(ii) Define the environment variable "GENN_PATH" to point to the main GeNN directory, e.g. if you extracted/downloaded GeNN to /usr/local/GeNN, then you can add "export GENN_PATH=/usr/local/GeNN" to your login script (e.g. `.profile` or `.bashrc`. If you are using WINDOWS, the path should be a windows path as it will be interpreted by the Visual C++ compiler `cl`, and enciront variables are best set using `setx` in a Windows cmd window. To do so, open a Windows cmd window byt typing `cmd` in the search field of the start menu, followed by the `enter` key. In the `cmd` window type

```
setx GENN_PATH "C:\Users\me\GeNN"
```

where `C:\Users\me\GeNN` is the path to your GeNN directory.

(iii) Add $GENN_PATH/lib/bin to your PATH variable, e.g.

```
export PATH=$PATH:$GENN_PATH/lib/bin
```

in your login script, or in windows,

```
setx PATH=%PATH%;%GENN_PATH%/lib/bin
```

(iv) If you haven't installed CUDA on your machine, obtain a fresh installation of the NVIDIA CUDA toolkit from https://developer.nvidia.com/cuda-downloads

(v) Set the `CUDA_PATH` variable if it is not already set by the system, by putting

```
export CUDA_PATH=/usr/local/cuda
```

in your login script (or, if CUDA is installed in a non-standard location, the appropriate path to the main CUDA directory). For most people, this will be done by the CUDA install script and the default value of /usr/local/cuda is fine. In Windows, use `setx` to set this variable,

```
setx CUDA_PATH
```

This normally completes the installation.

Depending on the needs of your own projects, e.g., depencies on other libraries or non-standard installation paths of libraries used by GeNN, you may want to modify Makefile examples under `$GENN_PATH/userproject/xxx_project` and `$GENN_PATH/userproject/xxx_project/model` to add extra linker-, include- and compiler-flags on a per-project basis, or modify global default flags in $GENN_PATH/lib/include/makefile_common.mk.

For all makefiles there are separate makefiles for Unix-style operating systems (GNUmakefile) such as Linux or MacOS and for Windows (WINmakefile).

If you are using GeNN in Windows, you can use make.bat to build examples which will attempt to setup your development environment by executing `vcvarsall.bat` which is part of every Visual Studio distribution. For this to work properly, GeNN must be able to locate the Visual Studio install directory, which should be contained in the `VS_PATH` environment variable. You can set this variable by hand if it is not already set by the Visual C++ installer by typing

```
setx VS_PATH "C:\Program Files (x86)\Microsoft Visual Studio 10.0"
```

Note

- The exact path and name of Visual C++ installations will vary between systems.
- Double quotation marks like in the above example are necessary whenever a path contains spaces.

GeNN also has experimental CYGWIN support. However, with the introduction of native Windows support in GeNN 1.1.3, this is not being developed further and should be considered as deprecated.

2.4 Testing Your Installation

To test your installation, follow the example in the Quickstart section.

Top | Next

3 Quickstart

GeNN is based on the idea of code generation for the involved GPU or CPU simulation code for neuronal network models but leaves a lot of freedom how to use the generated code in the final application. To facilitate the use of GeNN on the background of this philosophy, it comes with a number of complete examples containing both the model description code that is used by GeNN for code generation and the "user side code" to run the generated model and safe the results. Running these complete examples should be achievable in a few minutes. The necessary steps are described below.

3.1 Running an Example Model in Unix

In order to get a quick start and run a provided model, open a shell, navigate to `GeNN/tools` and type

```
make
```

This will compile additional tools for creating and running example projects. For a first complete test, the system is best used with a full driver program such as in the Insect olfaction model example:

```
tools/generate_run <0(CPU)/1(GPU)> <#AL> <#KC> <#LH> <#DN> <gscale> <OUTNAME> <MODEL> <DEBUG> <FTYPE>
       <REUSE>.
```

To compile `generate_run.cc`, navigate to the `userproject/MBody1_project` directory and type

```
make all
```

This will generate an executable that you can invoke with, e.g.,

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody1 0 FLOAT 0
```

which would generate and simulate a model of the locust olfactory system with 100 projection neurons, 1000 Kenyon cells, 20 lateral horn interneurons and 100 output neurons n the mushroom body lobes.

The tool generate_run will generate connectivity matrices for the model `MBody1` and store them into files, compile and run the model on the GPU using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument `1` with the device number of the desired GPU plus 2, e.g., `2` for GPU 0. All input and output files will be prefixed with `test1` and will be created in a sub-directory with the name `test1_output`. The third to last parameter `0` will switch the debugging mode off, `1` would switch it on. More about debugging in the debugging section . The parameter `FLOAT` will run the model in float (single precision floating point), using `DOUBLE` would use double precision. The last parameter regulates whether previously generated files for connectivity and input should be reused (1) or files should be generated anew (0).

The MBody1 example is already a highly integrated example that showcases many of the features of GeNN and how to program the user-side code for a GeNN application. More details in the User Manual

3.2 Running an Example Model in Windows

All interaction with GeNN programs are command-line based and hence are executed within a `cmd` window. Open a `cmd` window and navigate to the `userprojects\tools` directory.

```
cd %GENN_PATH%\userprojects\tools
```

Then type

```
make.bat all
```

to compile a number of tools that are used by the example projects to generate connectivity and inputs to model networks.

The navigate to the `MBody1_project` directory.

```
cd ..\MBody1_project
```

By typing

```
make.bat all
```

you can compile the `generate_run` engine that allows to run a Insect olfaction model model of the insect mushroom body:

```
tools/generate_run <0(CPU)/1(GPU)> <#AL> <#KC> <#LH> <#DN> <gscale> <OUTNAME> <MODEL> <DEBUG> <FTYPE>
       <REUSE>.
```

To invoke `generate_run.exe` type, e.g.,

```
generate_run.exe 1 100 1000 20 100 0.0025 test1 MBody1 0 FLOAT 0
```

which would generate and simulate a model of the locust olfactory system with 100 projection neurons, 1000 Kenyon cells, 20 lateral horn interneurons and 100 output neurons n the mushroom body lobes.

The tool `generate_run.exe` will generate connectivity matrices for the model `MBody1` and store them into files, compile and run the model on an automatically chosen GPU using these files as inputs and output the resulting spiking activity. To fix the GPU used, replace the first argument `1` with the device number of the desired GPU plus 2, e.g., `2` for GPU 0. All input and output files will be prefixed with `test1` and will be created in a sub-directory with the name `test1_output`. The last parameter `0` will switch the debugging mode off, `1` would switch it on. More about debugging in the debugging section .

The MBody1 example is already a highly integrated example that showcases many of the features of GeNN and how to program the user-side code for a GeNN application. More details in the User Manual

3.3 How to use GeNN for New Projects

Creating and running projects in GeNN involves a few steps ranging from defining the fundamentals of the model, inputs to the model, details of the model like specific connectivity matrices or initial values, running the model and analyzing or saving the data.

The most common way to use GeNN is to create or modify a program such as `userproject/MBody1_project/generate_run.cc` that wraps around other programs that are used for each of the necessary steps listed above. In more detail, what `generate_run` and similar programs do is:

1. To use other tools (programs) to generate connectivity matrices and store them into files.
2. To build the source code of a model simulation using GeNN. In the example of the MBody1_project this entails writing neuron numbers into `userproject/include/sizes.h`, and executing

   ```
   buildmodel.sh MBody1 [DEBUG OFF/ON]
   ```

   The `buildmodel.sh` script compiles the installed GeNN code generator in conjunction with the user-provided model description (see below), in this example `model/MBody1.cc`. It then executes the GeNN code generator to generate the complete model simulation code for the MBody1 model.

3. To compile the generated model code, that can be found in `model/MBody1_CODE/` by invoking `make clean && make` in the `model` directory. It is at this stage that GeNN generated model simulation code is combined with user-side run-time code, in this example `classol_sim.cu` (classify-olfaction-simulation) which uses the `map_classol` (map-neuron-based-classifier-olfaction) class.
4. To finally run the resulting stand-alone simulator executable, in the MBody1 example `classol_sim` in the `model` directory.

The `generate_run` tool is only a suggested usage scenario of GeNN. Alternatively, users can manually execute the four steps above or integrate GeNN with development environments of their own choice.

Note

The usage scenario described was made explicit for Unix environments. In Windows the setup is essentially the same except for the usual operating system dependent syntax differences, e.g. the build script is named buildmodel.bat, compilation of the generated model simulator would be `make.bat clean && make.bat all`, or, `nmake /f WINmakefile clean && nmake /f WINmakefile all`, and the resulting executable would be named `classol_sim.exe`.

GeNN comes with several example projects which showcase how to use its features. The MBody1 example discussed above is one of the many provided examples that are described in more detail in Example projects.

3.4 Defining a New Model in GeNN

According to the work flow outlined above, there are several steps to be completed to define a neuronal network model.

1. The neuronal network of interest is defined in a model definition file, e.g. `Example1.cc`.

   Note

   GeNN follows a convention in which C/C++ files end with `.cc` and the model definition file will only be recognized by the `buildmodel.sh` or `buildmodel.bat` scripts if it follows this convention and ends on `.cc`.

2. Within the the model definition file `Example1.cc`, the following tasks need to be completed:

   a) The time step `DT` needs to be defined, e.g.

   ```
   #define DT 0.1
   ```

   Note

   All provided examples and pre-defined model elements in GeNN work with units of mV, ms, nF and muS. However, the choice of units is entirely left to the user if custom model elements are used.

   b) The GeNN files `modelSpec.h` and `modelSpec.cc` need to be included,

   ```
   #include "modelSpec.h"
   #include "modelSpec.cc"
   ```

   c) The values for initial variables and parameters for neuron and synapse populations need to be defined, e.g.

   ```
   float myPOI_p[4]= {
     0.1,        // 0 - firing rate
     2.5,        // 1 - refractory period
     20.0,       // 2 - Vspike
     -60.0       // 3 - Vrest
   };
   ```

   would define the (homogeneous) parameters for a population of Poisson neurons.

Note

The number of required parameters and their meaning is defined by the neuron or synapse type. Refer to the User Manual for details.
We recommend, however, to use comments like in the above example to achieve maximal clarity of each parameter's meaning.

If heterogeneous parameter values are needed for any particular population of neurons (synapses), a new neuron (synapse) type needs to be defined in which these parameters are defined as "variables" rather than parameters. See the User Manual for how to define new neuron (synapse) types.

d) the actual network needs to be defined in the form of a function `modelDefinition`, i.e.

```
void modelDefinition(NNmodel &model);
```

Note

The name `modelDefinition` and its parameter of type `NNmodel&` are fixed and cannot be changed if GeNN is to recognize it as a model definition.

`MBody1.cc` shows a typical example of a model definition function. In its core it contains calls to `model.addNeuronPopulation` and `model.addSynapsePopulation` to build up the network. For a full range of options for defining a network, refer to the User Manual.

1. The programmer defines her own "user-side" modeling code similar to the code in `userproject/MBody1_project/model/map_classol.*` and `userproject/MBody1_project/model/classol_sim.*`. In this code,

   a) she defines the connectivity matrices between neuron groups. (In the MBody1 example those are read from files). Refer to the User Manual for the required format of connectivity matrices for dense or sparse connectivities.

   b) she defines input patterns (e.g. for Poisson neurons like in the MBody1 example) or individual initial values for neuron and/or synapse variables.

   Note

   The initial values given in the `modelDefinition` are automatically applied homogeneously to every individual neuron or synapse in each of the neuron or synapse groups.

   c) she uses `stepTimeGPU(...);` to run one time step on the GPU or `stepTimeCPU(...);` to run one on the CPU. (both GPU and CPU versions are always compiled).

   Note

   However, mixing CPU and GPU execution does not make too much sense. Among other things, The CPU version uses the same host side memory where to results from the GPU version are copied, which would lead to collisions between what is calculated on the CPU and on the GPU (see next point). However, in certain circumstances, expert users may want to split the calculation and calculate parts (e.g. neurons) on the GPU and parts (e.g. synapses) on the CPU. In such cases the fundamental kernel and function calls contained in `stepTimeXXX` need to be used and appropriate copies of the data from the CPU to the GPU and vice versa need to be performed.

   d) she uses functions like `copyStateFromDevice()` etc to transfer the results from GPU calculations to the main memory of the host computer for further processing.

   e) she analyzes the results. In the most simple case this could just be writing the relevant data to output files.

4 Examples

GeNN comes with a number of complete examples. At the moment, there are seven such example projects provided with GeNN.

4.1 Single compartment Izhikevich neuron(s)

This is a minimal example, with only one neuron population (with more or less neurons depending on the command line, but without any synapses). The neurons are Izhikevich neurons [1] with homogeneous parameters across the neuron population. The model can be used by navigating to the `userproject/OneComp_project` directory and entering a command line

```
./generate_run <CPU/GPU> <n> <OUTNAME> <MODEL> <DEBUG> <FTYPE> <REUSE>.
```

All parameters are mandatory and signify:

- `CPU/GPU`: Choose whether to run the simulation on CPU (`0`) or GPU (`1`).
- `n`: Number of neurons
- `OUTNAME`: The base name of the output location and output files
- `MODEL`: The name of the model to execute, as provided this would be `OneComp`
- `DEBUG`: Whether to start in debug mode (1) or normally (0)
- `FTYPE`: Floating point type `FLOAT` or `DOUBLE`
- `REUSE`: whether to re-use input files

This would create `n` tonically spiking Izhikevich neuron(s) with no connectivity, receiving a constant, identical 4 nA input current.

For example, navigate to the `userproject/OneComp_project` directory and type

```
make all
./generate_run 1 1 Outdir OneComp_sim OneComp 0 FLOAT 0
```

to model a single neuron which output will be saved in the `Outdir_output` directory.

4.2 Izhikevich neurons driven by Poisson input spike trains:

In this example project there is again a pool of non-connected Izhikevich model neurons [1] that are connected to a pool of Poisson input neurons with a fixed probability.

The model can be compiled by navigating to the `userproject\PoissonIzh_project` directory and typing

```
make all
```

Subsequently it can be invoked using the following command line

```
./generate_run <CPU/GPU> <nPOISSON> <nIZHIKEVICH> <PCONN> <GSCALE> <OUTNAME> <MODEL> <DEBUG OFF/ON> <
      FTYPE> <REUSE>
```

All parameters are mandatory and signify:

- CPU/GPU: Choose whether to run the simulation on CPU (0) or GPU (1).
- \nPOISSON: Number of Poisson input neurons
- \nIZHIKEVICH: Number of Izhikevich neurons
- PCONN: Probability of connections
- OUTNAME: The base name of the output location and output files
- MODEL: The name of the model to execute, as provided this would be PoissonIzh
- DEBUG: Whether to start in debug mode (1) or normally (0)
- FTYPE: Floating point type FLOAT or DOUBLE
- REUSE: whether to re-use input files

For example, navigate to the userproject/PoissonIzh_project directory and type

```
./generate_run 1 100 10 0.5 2 Outdir PoissonIzh 0 DOUBLE 0
```

This will generate a network of 100 Poisson neurons connected to 10 Izhikevich neurons with a probability of 0.5. The same example network can be used with sparse connectivity (i.e. sparse matrix representations for the connectivity within GeNN) by using the keyword SPARSE in the addSynapsePopulation instead of DENSE in PoissonIzh.cc, uncommenting the lines following the comment //SPARSE CONNECTIVITY in the file PoissonIzh_sim.cu and commenting the lines following //DENSE CONNECTIVITY. In this example the model would be simulated with double precision variables and input files and connectivity would not be reused from earlier runs.

4.3 Pulse-coupled Izhikevich network

This example model is inspired by simple thalamo-cortical network of Izhikevich [1] with an excitatory and an inhibitory population of spiking neurons that are randomly connected.

The model can be built by navigating to the userproject/Izh_Sparse_project directory and typing

```
make all
```

It can then be used as

```
./generate_run <CPU/GPU> <nN> <nCONN> <GSCALE> <OUTNAME> <MODEL> <DEBUG OFF/ON>
 <FTYPE> <REUSE>
```

All parameters are mandatory and signify:

- CPU/GPU: Choose whether to run the simulation on CPU (0) or GPU (1).
- \nN: Number of neurons
- \nCONN: Number of connections per neuron
- 'GSCALE': General scaling of synaptic conductances
- OUTNAME: The base name of the output location and output files
- MODEL: The name of the model to execute, as provided this would be Izh_sparse
- DEBUG: Whether to start in debug mode (1) or normally (0)
- FTYPE: Floating point type FLOAT or DOUBLE

- `REUSE`: whether to re-use input files

The model creates a pulse-coupled network [1] with 80% excitatory 20% inhibitory neurons, each connecting to nCONN neurons using the sparse matrix connectivity methods of GeNN.

For example, typing

```
./generate_run 1 10000 1000 1 Outdir Izh_sparse 0 FLOAT 0
```

generates a random network of 8000 excitatory and 2000 inhibitory neurons which each have 1000 outgoing synapses to randomly chosen post-synaptic target neurons. The synapses are of a simple pulse-coupling type. The results of the simulation are saved in the directory `Outdir_output`, debugging is switched off, and the connectivity is generated afresh (rather than being read from existing files).

Note

If connectivity were to be read from files, the connectivity files would have to be in the `inputfiles` subdirectory and be named according to the names of the synapse populations involved, e.g., `gIzh_sparse_ee` (<variable name>=`g`<model name>=`Izh_sparse`<synapse population>=`_ee`). These name conventions are not part of the core GeNN definitions and it is the privilege (or burden) of the user to find their own in their own versions of `generate_run`.

4.4 Izhikevich network with delayed synapses

This example project demonstrates the feature of synaptic delays in GeNN. It creates a network of three Izhikevich neuron groups, connected all-to-all with short, medium and long delay synapse groups. Synaptic weights are chosen such that neurons in the output group only spike if they are simultaneously excited by the input neurons, via synapses with long delay, and the interneurons, via synapses with shorter delay.

To run this example project, navigate to `userproject/SynDelay_project` and type, e.g.,

```
buildmodel SynDelay
make clean && make
./bin/release/syn_delay 1 output
```

4.5 Insect olfaction model

This project implements the insect olfaction model by Nowotny et al. [2] that demonstrates self-organized clustering of odours in a simulation of the insect antennal lobe and mushroom body. As provided the model works with conductance based Hodgkin-Huxley neurons [4] and several different synapse types, conductance based (but pulse-coupled) excitatory synapses, graded inhibitory synapses and synapses with a simplified STDP rule.

To explore the model navigate to `userproject/MBody1_project/` and type

```
make all
./generate_run
```

This will show you the command line parameters that are needed,

```
generate_run <CPU/GPU> <nAL> <nKC> <nLHI> <nDN> <GSCALE> <OUTNAME> <MODEL> <DEBUG> <FTYPE> <REUSE>
```

All parameters are mandatory and signify:

- `CPU/GPU`: Choose whether to run the simulation on CPU (`0`) or GPU (`1`).
- `nAL`: Number of neurons in the antennal lobe (AL), the input neurons to this model
- `nKC`: Number of Kenyon cells (KC) in the "hidden layer"
- `nLH`: Number of lateral horn interneurons, implementing gain control
- `nDN`: Number of decision neurons (DN) in the output layer
- `GSCALE`: A general rescaling factor for snaptic strength
- `OUTNAME`: The base name of the output location and output files
- `MODEL`: The name of the model to execute, as provided this would be `MBody1`
- `DEBUG`: Whether to start in debug mode (1) or normally (0)
- `FTYPE`: Floating point type `FLOAT` or `DOUBLE`
- `REUSE`: whether to re-use input files

The tool generate_run will generate connectivity files for the model `MBody1`, compile this model for the CPU and GPU and execute it. The command line parameters are the numbers of neurons in the different neuropils of the model and an overall synaptic strength scaling factor. A typical call would be, e.g.,

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody1 0 FLOAT 0
```

which would generate a model, and run it on the GPU (first parameter), with 100 antennal lobe neurons, 1000 mushroom body Kenyon cells, 20 lateral horn interneurons and 100 mushroom body output neurons. All output files will be prefixed with `test1` and stored in `test1_output`. The model that is run is defined in `model/↩ MBody1.cc`, debugging is switched off, the model would be simulated using float (single precision floating point) variables and parameters and the connectivity and input would be generated afresh for this run.

As provided, the model outputs a file `test1.out.st` that contains the spiking activity observed in the simulation, There are two columns in this ASCII file, the first one containing the time of a spike and the second one the ID of the neuron that spiked. Users of matlab can use the scripts in the `matlab` directory to plot the results of a simulation. For more about the model itself and the scientific insights gained from it see [2].

4.6 Insect olfaction model with user-defined neuron and synapse models

This examples recapitulates the exact same model as Insect olfaction model above but with user-defined model types for neurons and synapses. It is run the same way as Insect olfaction model, e.g.

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody_userdef 0 FLOAT 0
```

But the way user-defined types are used should be very instructive to advanced users wishing to do the same with their models.

4.7 Insect Olfaction Model using INDIVIDUALID connectivity scheme

This example is very similar to the Insect olfaction model example. The only difference is that PN to KC connections are defined with the INDIVIDUALID mechanism.

It is run the same way as Insect olfaction model, e.g.

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody_individualID 0 FLOAT 0
```

4.8 Insect Olfaction Model using INDIVIDUALID connectivity scheme

A variation of the Insect olfaction model example using synaptic delays. In this example, the Kenyon Cell-Decision Neuron synapses are delayed by (5 ∗ DT) ms, and the Decision Neuron-Decision Neuron synapses are delayed by (3 ∗ DT) ms. The example is intended to test the operation of synapses which have a combination of delayed spike propagation and STDP (plasticity).

It is run the same way as Insect olfaction model, e.g.

```
./generate_run 1 100 1000 20 100 0.0025 test1 MBody_delayedSyn 0 FLOAT 0
```

4.9 Voltage clamp simulation to estimate Hodgkin-Huxley parameters

This example simulates a population of Hodgkin-Huxley neuron models on the GPU and evolves them with a simple guided random search (simple GA) to mimic the dynamics of a separate Hodgkin-Huxley neuron that is simulated on the CPU. The parameters of the CPU simulated "true cell" are drifting according to a user-chosen protocol↩ : Either one of the parameters gNa, ENa, gKd, EKd, gleak, Eleak, Cmem are modified by a sinusoidal addition (voltage parameters) or factor (conductance or capacitance) - protocol 0-6. For protocol 7 all 7 parameters undergo a random walk concurrently.

The syntax of the generate_run program is

```
generate_run <CPU=0, GPU=1> <protocol> <nPop> <totalT> <outdir> <debug mode? (0/1)> <
      GPU choice>
```

The parameters are

- GPU/CPU: Whether to use the GPU (1) or CPU (0) for the model neuron population
- protocol: Which changes to apply during the run to the parameters of the "true cell"
- nPop: Number of neurons in the tracking population
- totalT: Time in ms how long to run the simulation
- outdir: The directory in which to save results
- debug mode: whether to start in debugging mode (1) or not (0) GPU choice: explicit choice of the GPU to be used.

An example invocation is:

```
./generate_run 1 2 5000 1000 test1 0 0
```

This will simulate nPop= 5000 Hodgkin-Huxley neurons on the GPU which will for 1000 ms be matched to a Hodgkin-Huxley neuron where the parameter gKd is sinusoidally modulated. The output files will be written into a directory of the name test1_output, which will be created if it does not yet exist.

Previous | Top | Next

5 Release Notes for GeNN v2.0

Version 2.0 of GeNN comes with a lot of improvements and added features, some of which have necessitated some changes to the structure of parameter arrays among others.

User Side Changes

1. Users are now required to call `initGeNN()` in the model definition function before adding any populations to the neuronal network model.
2. glbscnt is now call glbSpkCnt for consistency with glbSpkEvntCnt.
3. There is no longer a privileged parameter `Epre`. Spike type events are now defined by a code string `spk↩ EvntThreshold`, the same way proper spikes are. The only difference is that Spike type events are specific to a synapse type rather than a neuron type.
4. The function setSynapseG has been deprecated. In a `GLOBALG` scenario, the variables of a synapse group are set to the initial values provided in the `modeldefinition` function.
5. Due to the split of synaptic models into weightUpdateModel and postSynModel, the parameter arrays used during model definition need to be carefully split as well so that each side gets the right parameters. For example, previously

"C++ float myPNKC_p[3]= { 0.0, // 0 - Erev: Reversal potential -20.0, // 1 - Epre: Presynaptic threshold potential 1.0 // 2 - tau_S: decay time constant for S [ms] }; "would define the parameter array of three parameters,Erev,Epre, andtau_Sfor a synapse of typeNSYNAPS↩ E. This now needs to be "split" into "C++ float *myPNKC_p= NULL; float postExpPNK↩ C[2]={ 1.0, // 0 - tau_S: decay time constant for S [ms] 0.0 // 1 - Erev: Reversal potential }; "i.e. parametersErevandtau_Sare moved to the post-synaptic model and its parameter array of two parameters.Epreis discontinued as a parameter forNSYNAPSE. As a consequence the weightupdate model ofNSYNAPSEhas no parameters and one can passNULLfor the parameter array inaddSynapsePopulation".

The correct parameter lists for all defined neuron and synapse model types are listed in the User Manual.

Note:* If the parameters are not redefined appropriately this will lead to uncontrolled behaviour of models and likely to sgementation faults and crashes.

1. Advanced users can now define variables as type `scalar` when introducing new neuron or synapse types. This will at the code generation stage be translated to the model's floating point type (ftype), `float` or `double`. This works for defining variables as well as in all code snippets. Users can also use the expressions `SCALAR_MAX` and `SCALAR_MIN` for `FLT_MIN`, `FLT_MAX`, `DBL_MIN` and `DBL_MAX`, respectively. Corresponding definitions of `scalar`, `SCALAR_MIN` and `SCALAR_MAX` are also available for user-side code whenever the code-generated file `runner.cc` has been included.
2. The example projects have been re-organized so that wrapper scripts of the `generate_run` type are now all located together with the models they run instead of in a common `tools` directory. Generally the structure now is that each example project contains the wrapper script `generate_run` and a `model` subdirectory which contains the model description file and the user side code complete with Makefiles for Unix and Windows operating systems. The generated code will be deposited in the `model` subdirectory in its own `modelname_CODE` folder. Simulation results will always be deposited in a new sub-folder of the main project directory.
3. The `addSynapsePopulation(...)` function has now more mandatory parameters relating to the introduction of separate weightupdate models (pre-synaptic models) and postsynaptic models. The correct syntax for the `addSynapsePopulation(...)` can be found with detailed explanations in teh User Manual.
4. We have introduced a simple performance profiling method that users can employ to get an overview over the differential use of time by different kernels. To enable the timers in GeNN generated code, one needs to declare "C++ networkmodel.setTiming(TRUE); "This will make available and

operate GPU-side cudeEvent based timers whose cumulative value can be found in the double precision variablesneuron_tme, synapse_tmeandlearning_tme. They measure the accumulated time that has been spent calculating the neuron kernel, synapse kernel and learning kernel, respectively. CP↩ U-side timers for the simulation functions are also available and their cumulative values can be obtained through "C++ float x= sdkGetTimerValue(&neuron↩ _timer); float y= sdkGetTimerValue(&synapse_timer); float z= sdkGetTimerValue(&learning_timer); "' The Insect olfaction model example shows how these can be used in the user-side code. To enable timing profiling in this example, simply enable it for GeNN: "'C++ model.setTiming(TRUE); "inMBody1.cc<tt>'smodel↩ Definitionfunction and define the macroTIMINGinclassol_sim.h<tt> "C++ #define TIMING "This will have the effect that timing information is output intoOUTNA↩ ME_output/OUTNAME.timingprofile'.

Developer Side Changes

1. allocateSparseArrays() has been changed to take the number of connections, connN, as an argument rather than expecting it to have been set in the Connetion struct before the function is called as was the arrangement previously.

2. For the case of sparse connectivity, there is now a reverse mapping implemented with revers index arrays and a remap array that points to the original positions of variable values in teh forward array. By this mechanism, revers lookups from post to pre synaptic indices are possible but value changes in the sparse array values do only need to be done once.

3. SpkEvnt code is no longer generated whenever it is not actually used. That is also true on a somewhat finer granularity where variable queues for synapse delays are only maintained if the corresponding variables are used in synaptic code. True spikes on the other hand are always detected in case the user is interested in them.

Previous | Top | Next

/∗!

This release includes some new features and several bug fixes on GeNN v2.0.

User Side Changes

1. Block size debugging flag and the asGoodAsZero variables are moved into include/global.h.

2. NGRADSYNAPSES dynamics have changed (See Bug fix #4) and this change is applied to the example projects. If you are using this synapse model, you may want to consider changing model parameters.

3. The delay slots are now such that NO_DELAY is 0 delay slots (previously 1) and 1 means an actual delay of 1 time step.

4. The convenience function convertProbabilityToRandomNumberThreshold(float ∗, uint64_t ∗, int) was changed so that it actually converts firing probability/timestep into a threshold value for the GeNN random number generator (as its name always suggested). The previous functionality of converting a *rate* in kHz into a firing threshold number for the GeNN random number generator is now provided with the name convertRateToRandomNumberThreshold(float ∗, uint64_t ∗, int)

5. Every model definition function modelDefinition() now needs to end with calling NNmodel↩ ::finalize() for the defined network model. This will lock down the model and prevent any further changes to it by the supported methods. It also triggers necessary analysis of the model structure that should only be performed once. If the finalize() function is not called, GeNN will issue an error and exit before code generation.

6. To be more consistent in function naming the `pull<SYNAPSENAME>FromDevice` and `push<SYNAPSENAME>ToDevice` have been renamed to `pull<SYNAPSENAME>StateFromDevice` and `push<SYNAPSENAME>StateToDevice`. The old versions are still supported through macro definitions to make the transition easier.
7. New convenience macros are now provided to access the current spike numbers and identities of neurons that spiked. These are called spikeCount_XX and spike_XX where "XX" is the name of the neuron group. They access the values for the current time step even if there are synaptic delays and spikes are stored in circular queues.
8. There is now a pre-defined neuron type "SPIKECOURCE" which is empty and can be used to define PyNN style spike source arrays.
9. The macros FLOAT and DOUBLE were replaced with GENN_FLOAT and GENN_DOUBLE due to name clahes with typedefs in Windows that define FLOAT and DOUBLE.

Developer Side Changes

1. We introduced a file definitions.h, which is generated and filled with useful macros such as spkQuePtrShift which tells users where in the circular spike queue their spikes start.

Improvements

1. Improved debugging information for block size optimisation and device choice.
2. Changed the device selection logic so that device occupancy has larger priority than device capability version.
3. A new HH model called TRAUBMILES_PSTEP where we can set the number of inner loops as a parameter is introduced. It uses the TRAUBMILES_SAFE method.
4. An alternative method is added for the insect olfaction model in order to fix the number of connections to a maximum of 10K in order to avoid negative conductance tails.
5. We introduced a #define for an "int_" function that translates floating points to integers.

Bug fixes:

1. AtomicAdd replacement for old GPUs were used by mistake if the model runs in double precision.
2. Timing of individual kernels is fixed and improved.
3. More careful setting of maximum number of connections in sparse connectivity, covering mixed dense/sparse network scenarios.
4. NGRADSYNAPSES was not scaling correctly with varying time step.
5. Fixed a bug where learning kernel with sparse connectivity was going out of array.
6. Fixed synapse kernel name substitutions where the "dd_" prefix was omitted by mistake.

Previous | Top | Next

6 User Manual

6.1 Contents

Introduction
Defining a network model
Neuron models
Synapse models

6.2 Introduction

GeNN is a software library for facilitating the simulation of neuronal network models on NVIDIA CUDA enabled GPU hardware. It was designed with computational neuroscience models in mind rather than artificial neural networks. The main philosophy of GeNN is two-fold:

1. GeNN relies heavily on code generation to make it very flexible and to allow adjusting simulation code to the model of interest and the GPU hardware that is detected at compile time.
2. GeNN is lightweight in that it provides code for running models of neuronal networks on GPU hardware but it leaves it to the user to write a final simulation engine. It so allows maximal flexibility to the user who can use any of the provided code but can fully choose, inspect, extend or otherwise modify the generated code. She can also introduce her own optimisations and in particular control the data flow from and to the GPU in any desired granularity.

This manual gives an overview of how to use GeNN for a novice user and tries to lead the user to more expert use later on. With this we jump right in.

Previous | Top | Next

6.3 Defining a network model

A network model is defined by the user by providing the function

```
void modelDefinition(NNmodel &model)
```

in a separate file with name `name.cc`, where `name` is the name of the model network under consideration. In this function, the following tasks must be completed:

1. The name of the model must be defined:

   ```
   model.setName("MyModel");
   ```

2. Neuron populations (at least one) must be added (see Defining neuron populations). The user may add as many neuron populations as she wishes. If resources run out, there will not be a warning but GeNN will fail. However, before this breaking point is reached, GeNN will make all necessary efforts in terms of block size optimisation to accommodate the defined models. All populations must have a unique name.
3. Synapse populations (zero or more) can be added (see Defining synapse populations). Again, the number of synaptic connection populations is unlimited other than by resources.

Note

GeNN uses the convention where C/C++ files end in `.cc`. If this is not adhered to the build script `buildmodel.sh` will not recognise the model definition file.

6.3.1 Defining neuron populations

Neuron populations are added using the function

```
model.addNeuronPopulation(name, n, TYPE, para, ini);
```

where the arguments are:

- `const char* name`: Unique name of the neuron population
- `int n`: number of neurons in the population
- `int TYPE`: Type of the neurons, refers to either a standard type (see Neuron models) or user-defined type; this is an integer that indicates the position in the list of all neuron models where the model in question is stored.
- `double *para`: Parameters of this neuron type
- `double *ini`: Initial values for variables of this neuron type

The user may add as many neuron populations as the model necessitates. They must all have unique names. The possible values for the arguments, predefined models and their parameters and initial values are detailed Neuron models below.

6.3.2 Defining synapse populations

Synapse populations are added with the command

```
model.addSynapsePopulation("name", sType, sConn, gType, delay, postSyn, "preName", "
      postName", sIni, sParam, postSynIni, postSynParam);
```

where the arguments are

- `const char*` name: The name of the synapse population
- `int sType`: The type of synapse to be added. See Built-in Models below for the available predefined synapse types.
- `int sConn`: The type of synaptic connectivity. the options currently are "ALLTOALL", "DENSE", "SPARSE" (see Connectivity types)
- `int gType`: The way how the synaptic conductivity g will be defined. Options are "INDIVIDUALG", "GLOBALG", "INDIVIDUALID". For their meaning, see LEARN1SYNAPSE (Learning Synapse with a Primitive Role) below.
- `int delay`: Synaptic delay (in multiples of the simulation time step `DT`).
- `int postSyn`: Postsynaptic integration method. See Postsynaptic integration methods for predefined types.
- `char* preName`: Name of the (existing!) pre-synaptic neuron population.
- `char* postName`: Name of the (existing!) post-synaptic neuron population.
- `double *sIni`: A C-array of doubles containing initial values for the (pre-) synaptic variables.
- `double *sParam`: A C-array of double precision that contains parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored. For pre-defined synapse types the required parameters and their meaning are listed in NSYNAPSE (No Learning) below.

- `double * psIni`: A C-array of double precision numbers containing initial values for the post-synaptic model variables
- `double * psPara`: A C-array of double precision numbers containing parameters fo the post-snaptic model.

  Note

  If the synapse conductance definition type is "GLOBALG" then the global value of the synapse conductances is taken from the initial value provided in `sINI`. (The function setSynapseG() from earlier versions of GeNN has been deprecated).

  Synaptic updates can occur per "true" spike (i.e at one point per spike, e.g. after a threshold was crossed) or for all "spike type events" (e.g. all points above a given threshold). This is defined within each given synapse type.

Previous | Top | Next

6.4 Neuron models

There is a number of predefined models which can be chosen in the `addNeuronGroup(...)` function by their unique cardinal number, starting from 0. For convenience, C variables with readable names are predefined

- 0: MAPNEURON
- 1: POISSONNEURON
- 2: TRAUBMILES_FAST
- 3: TRAUBMILES_ALTERNATIVE
- 4: TRAUBMILES_SAFE
- 5: TRAUBMILES_PSTEP
- 6: IZHIKEVICH
- 7: IZHIKEVICH_V
- 8: SPIKESOURCE

Note

Ist is best practice to not depend on the unique cardinal numbers but use predefined names. While it is not intended that the numbers will change the unique names are guaranteed to work in all future versions of GeNN.

6.4.1 MAPNEURON (Map Neurons)

The MAPNEURON type is a map based neuron model based on [3] but in the 1-dimensional map form used in [2] :

$$V(t+\Delta t) \quad = \quad \begin{cases} V_{\text{spike}}\left(\frac{\alpha V_{\text{spike}}}{V_{\text{spike}}-V(t)\beta I_{\text{syn}}}+y\right) & V(t)\leq 0 \\ V_{\text{spike}}\left(\alpha+y\right) & V(t)\leq V_{\text{spike}}\left(\alpha+y\right)\ \&\ V(t-\Delta t)\leq 0 \\ -V_{\text{spike}} & \text{otherwise} \end{cases}$$

Note

The MAPNEURON type only works as intended for the single time step size of DT= 0.5.

The MAPNEURON type has 2 variables:

- V - the membrane potential
- preV - the membrane potential at the previous time step

and it has 4 parameters:

- Vspike - determines the amplitude of spikes, typically -60mV
- alpha - determines the shape of the iteration function, typically α= 3
- y - "shift / excitation" parameter, also determines the iteration function,originally, y= -2.468
- beta - roughly speaking equivalent to the input resistance, i.e. it regulates the scale of the input into the neuron, typically β= 2.64 $\mathrm{M}\Omega$.

Note

The initial values array for the MAPNEURON type needs two entries for V and Vpre and the parameter array needs four entries for Vspike, alpha, y and beta, *in that order*.

6.4.2 POISSONNEURON (Poisson Neurons)

Poisson neurons have constant membrane potential (Vrest) unless they are activated randomly to the Vspike value if (t- SpikeTime) $>$ trefract.

It has 3 variables:

- V - Membrane potential
- Seed - Seed for random number generation
- SpikeTime - Time at which the neuron spiked for the last time

and 4 parameters:

- therate - Firing rate
- trefract - Refractory period
- Vspike - Membrane potential at spike (mV)
- Vrest - Membrane potential at rest (mV)

Note

The initial values array for the POISSONNEURON type needs three entries for V, Seed and SpikeTime and the parameter array needs four entries for therate, trefract, Vspike and Vrest, *in that order*. Internally, GeNN uses a linear approximation for the probability of firing a spike in a given time step of size DT, i.e. the probability of firing is therate times DT: $p = \lambda \Delta t$. This approximation is usually very good, especially for typical, quite small time steps and moderate firing rates. However, it is worth noting that the approximation becomes poor for very high firing rates and large time steps. An unrelated problem may occur with very low firing rates and small time steps. In that case it can occur that the firing probability is so small that the granularity of the 64 bit integer based random number generator begins to show. The effect manifests itself in that small changes in the firing rate do not seem to have an effect on the behaviour of the Poisson neurons because the numbers are so small that only if the random number is identical 0 a spike will be triggered.

6.4.3 TRAUBMILES_FAST (Hodgkin-Huxley neurons with Traub & Miles algorithm)

This conductance based model has been taken from [4] and can be described by the equations:

$$\begin{aligned} C\frac{dV}{dt} &= -I_{\mathrm{Na}} - I_K - I_{\mathrm{leak}} - I_M - I_{i,DC} - I_{i,\mathrm{syn}} - I_i, \\ I_{\mathrm{Na}}(t) &= g_{\mathrm{Na}} m_i(t)^3 h_i(t)(V_i(t) - E_{\mathrm{Na}}) \\ I_{\mathrm{K}}(t) &= g_{\mathrm{K}} n_i(t)^4 (V_i(t) - E_{\mathrm{K}}) \\ \frac{dy(t)}{dt} &= \alpha_y(V(t))(1 - y(t)) - \beta_y(V(t))y(t), \end{aligned}$$

where $y_i = m, h, n$, and

$$\begin{aligned} \alpha_n &= 0.032(-50 - V)/\big(\exp((-50 - V)/5) - 1\big) \\ \beta_n &= 0.5\exp((-55 - V)/40) \\ \alpha_m &= 0.32(-52 - V)/\big(\exp((-52 - V)/4) - 1\big) \\ \beta_m &= 0.28(25 + V)/\big(\exp((25 + V)/5) - 1\big) \\ \alpha_h &= 0.128\exp((-48 - V)/18) \\ \beta_h &= 4/\big(\exp((-25 - V)/5) + 1\big). \end{aligned}$$

and typical parameters are $C = 0.143$ nF, $g_{\mathrm{leak}} = 0.02672$ μS, $E_{\mathrm{leak}} = -63.563$ mV, $g_{\mathrm{Na}} = 7.15$ μS, $E_{\mathrm{Na}} = 50$ mV, $g_{\mathrm{K}} = 1.43$ μS, $E_{\mathrm{K}} = -95$ mV.

It has 4 variables:

- `V` - membrane potential E
- `m` - probability for Na channel activation m
- `h` - probability for not Na channel blocking h
- `n` - probability for K channel activation n

and 7 parameters:

- `gNa` - Na conductance in 1/(mOhms $*$ cm^2)
- `ENa` - Na equi potential in mV
- `gK` - K conductance in 1/(mOhms $*$ cm^2)
- `EK` - K equi potential in mV
- `gl` - Leak conductance in 1/(mOhms $*$ cm^2)
- `El` - Leak equi potential in mV
- `Cmem` - Membrane capacity density in muF/cm^2

Note

Internally, the ordinary differential equations defining the model are integrated with a linear Euler algorithm and GeNN integrates 25 internal time steps for each neuron for each network time step. I.e., if the network is simulated at `DT= 0.1` ms, then the neurons are integrated with a linear Euler algorithm with `lDT= 0.004` ms.

Other variants of the same model are TRAUBMILES_ALTERNATIVE, TRAUBMILES_SAFE and TRAUBMILES_↩ PSTEP. The former two are adressing the problem of singularities in the original Traub & Miles model [4]. At V= -52 mV, -25 mV, and -50 mV, the equations for α have denominators with value 0. Mathematically this is not a problem because the nominator is 0 as well and the left and right limits towards these singular points coincide. Numerically,

however this does lead to nan (not-a-number) results through division by 0. The TRAUBMILES_ALTERNATIVE model adds SCALAR_MIN to the denominators at all times which typically is completely effect-free because it is truncated from teh mantissa, except when the denominator is very close to 0, in which case it avoids the singular value.

TRAUBMILES_SAFE takes a much more direct approach in which at the singular points, the correct value calculated offline with l'Hopital's rule in substituted. This is implemented with "if" statements.

Finally, the TRAUBMILES_PSTEP model allows users to control the number of internal loops, or sub-timesteps, that are used. This is enabled by making the number of time steps an explicit parameter of the model.

6.4.4 IZHIKEVICH (Izhikevich neurons with fixed parameters)

This is the Izhikevich model with fixed parameters [1]. It is usually described as

$$\begin{aligned} \frac{dV}{dt} &= 0.04V^2 + 5V + 140 - U + I, \\ \frac{dU}{dt} &= a(bV - U), \end{aligned}$$

I is an external input current and the voltage V is reset to parameter c and U incremented by parameter d, whenever V >= 30 mV. This is paired with a particular integration procedure of two 0.5 ms Euler time steps for the V equation followed by one 1 ms time step of the U equation. Because of its popularity we provide this model in this form here event though due to the details of the usual implementation it is strictly speaking inconsistent with the displayed equations.

Variables are:

- `V` - Membrane potential
- `U` - Membrane recovery variable

Parameters are:

- `a` - time scale of U
- `b` - sensitivity of U
- `c` - after-spike reset value of V
- `d` - after-spike reset value of U

6.4.5 IZHIKEVICH_V (Izhikevich neurons with variable parameters)

This is the same model as IZHIKEVICH (Izhikevich neurons with fixed parameters) IZHIKEVICH but parameters are defined as "variables" in order to allow users to provide individual values for each individual neuron instead of fixed values for all neurons across the population.

Accordingly, the model has the Variables:

- `V` - Membrane potential
- `U` - Membrane recovery variable
- `a` - time scale of U
- `b` - sensitivity of U
- `c` - after-spike reset value of V
- `d` - after-spike reset value of U

and no parameters.

6.4.6 SPIKESOURCE (empty neuron which allows setting spikes from external sources)

This model does not contain any update code and can be used to implement the equivalent of a SpikeGenerator↩ Group in Brian or a SpikeSourceArray in PyNN.

6.4.7 Defining your own neuron type

In order to define a new neuron type for use in a GeNN application, it is necessary to populate an object of class `neuronModel` and append it to the global vector `nModels`. This can be done conveniently within the model↩ Definition fundtion just before the model is needed. The `neuronModel` class has several data members that make up the full description of the neuron model:

- `simCode` of type `string`: This needs to be assigned a C++ (stl) string that contains the code for executing the integration of the model for one time step. Within this code string, variables need to be referred to by , where NAME is the name of the variable as defined in the vector varNames. The code may refer to the predefined primitives `DT` for the time step size and for the total incoming synaptic current. It can also refer to a unique ID (within the population) using .
  Example:

  ```
  neuronModel model;
  model.simCode=String("$(V)+= (-$(a)$(V)+$(Isyn))*DT;");
  ```

  would implement a leaky itegrator $\frac{dV}{dt} = -aV + I_{\text{syn}}$.

- `varNames` of type `vector<string>`: This vector needs to be filled with the names of variables that make up the neuron state. The variables defined here as `NAME` can then be used in the syntax in the code string.
  Example:

  ```
  model.varNames.push_back(String("V"));
  ```

  would add the variable V as needed by the code string in the example above.

- `varTypes` of type `vector<string>`: This vector needs to be filled with the variable type (e.g. "float", "double", etc) for the variables defined in `varNames`. Types and variables are matched to each other by position in the respective vectors, i.e. the 0th entry of `varNames` will have the type stored in the 0th entry of `varTypes` and so on.
  Example:

  ```
  model.varTypes.push_back(String("float"));
  ```

  would designate the variable V to be of type float.

  Note

- `pNames` of type `vector<string>`: This vector will contain the names of parameters relevant to the model. If defined as `NAME` here, they can then be referenced as in the code string. The length of this vector determines the expected number of parameters in the initialisation of the neuron model. Parameters are assumed to be always of type double.

  ```
  model.pNames.push_back(String("a"));
  ```

  stores the parameter `a` needed in the code example above.

- `dpNames` of type `vector<string>`: Names of "dependent parameters". Dependent parameters are a mechanism for enhanced efficiency when running neuron models. If parameters with model-side meaning, such as time constants or conductances always appear in a certain combination in the model, then it is more efficient to pre-compute this combination and define it as a dependent parameter. This vector contains the names of such dependent parameters.

- `thresholdConditionCode` of type `vector<string>` (if applicable): Condition for true spike detection.

For example, if in the example above the original model had been $\frac{dV}{dt} = -g/CV + I_{\text{syn}}$. Then one could define the code string and parameters as

```
model.simCode=String("$(V)+= (-$(a)$(V)+$(Isyn))*DT;");
model.varNames.push_back(String("V"));
model.varTypes.push_back(String("float"));
model.pNames.push_back(String("g"));
model.pNames.push_back(String("C"));
model.dpNames.push_back(String("a"));
```

- `dps` of type `dpclass*`: The dependent parameter class, i.e. an implementation of `dpclass` which would return the value for dependent parameters when queried for them. E.g. in the example above it would return `a` when queried for dependent parameter 0 through `dpclass::calculateDerivedParameter()`. Examples how this is done can be found in the pre-defined classes, e.g. expDecayDp, pwSTDP, rulkovdp etc.

Once the completed `neuronModel` object is appended to the `nModels` vector,

```
nModels.push_back(model);
```

it can be used in network descriptions by referring to its cardinal number in the nModels vector. I.e., if the model is added as the 4th entry, it would be model "3" (counting starts at 0 in usual C convention). The information of the cardinal number of a new model can be obtained by referring to `nModels.size()` right before appending the new model, i.e. a typical use case would be.

```
int myModel= nModels.size();
nModels.push_back(model);
```

. Then one can use the model as

```
networkModel.addNeuronPopulation(..., myModel, ...);
```

6.4.8 Explicit current input to neurons (will be removed in future versions)

The user can decide whether a neuron group receives external input current or not in addition to the synaptic input that it receives from the network. External input to a neuron group is activated by calling `activateDirect↩ Input` function. It receives two arguments: The first argument is the name of the neuron group to receive input current. The second parameter defines the type of input. Current options are:

- 0: NOINP: Neuron group receives no input. This is the value by default when the explicit input is not activated.
- 1: CONSTINP: All the neurons receive a constant input value. This value may also be defined as a variable, i.e. be time-dependent.
- 2: MATINP: The input is read from a file containing the input matrix.
- 3: INPRULE: The input is defined as a rule. It differs from CONSTINP in the way that the input is injected: CONSTINP creates a value which is modified in real time, hence complex instructions are limited. INPRULE creates an input array which will be copied into the device memory.

6.5 Synapse models

A synapse model is a weightUpdateModel object which consists of variables, parameters, and string objects which will be substituted in the code. The three strings that will be substituted in the code for synapse update are:

- `simCode`: Simulation code that is used when a true spike is detected. The update is performed only once, one time step after threshold condition detection, which is defined by `thresholdConditionCode` in the neuron model of the corresponding presynaptic neuron population.

Typically, spikes lead to update of synaptic variables that then lead to the activation of input into the post-synaptic neuron. Most of the ime these inputs add linearly at the post-synaptic neuron. This is assumed in GeNN and the term to be added to the activation of the post-synaptic neuron should be provided as the value of the by the $(addtoinsyn) snippet. For example

```
"\$(addtoinsyn) = $(inc);"
```

where "inc" is a parameter of the weightupdate model, would define a constant increment of the synaptic input of a post-synaptic neuron for each pre-synaptic spike.

When a spike is detected in the presynaptic neuron population, the simCode is executed. Within the code snippet the $(addtoinSyn) update just discussed should be be followed by the $(updatelinsyn) keyword to indicate that the summation of synaptic inputs can now occur. This can then be followed by updates on the internal synapse variables that may have contributed to addtoinSyn. For an example, see NSYNAPSE (No Learning) for a simple synapse update model and LEARN1SYNAPSE (Learning Synapse with a Primitive Role) for a more complicated model that uses STDP.

- `simCodeEvnt`: Simulation code that is used for spike like events, where updates are done for all instances in which the event condition defined by `evntThreshold` is met. `evntThreshold` is also be provided as a code string. For an example, see NGRADSYNAPSE (Graded Synapse).
- `simLearnPost`: Simulation code which is used in the learnSynapsesPost kernel/function, which performs updates to synapses that are triggered by post-synaptic spikes. This is rather unusual other than in learning rules like e.g. STDP. For an example that uses `simLearnPost`, see LEARN1SYNAPSE (Learning Synapse with a Primitive Role).
- `synapseDynamics`: Simulation code that applies for every time step, i.e.is unlike the others not gated with a condition. This can be used where synapses have internal variables and dynamics that are described in continuous time, e.g. by ODEs. Usng this mechnanism is typically computationally veruy costly because of the large number of synapses in a typical network.

All code snippets can be used to manipulate any synapse variable and so implement both synaptic dynamics and learning processes.

6.5.1 Built-in Models

Currently 3 predefined synapse models are available:

- NSYNAPSE
- NGRADSYNAPSE
- LEARN1SYNAPSE

These are defined in `lib.include/utils.h`. The `MBody_userdef` example also includes a modified version of these models as user-defined models.

6.5.2 NSYNAPSE (No Learning)

If this model is selected, no learning rule is applied to the synapse and for each pre-synaptic spikes the synaptic conductances are simply added to the postsynaptic input variable. The model has 1 variable:

- `g` - conductance of `scalar` type

and no other parameters.

`simCode` is:

```
" $(addtoinSyn) = $(g);\n\
  $(updatelinsyn);\n"
```

6.5.3 NGRADSYNAPSE (Graded Synapse)

In a graded synapse, the conductance is updated gradually with the rule:

$$gSyn = g * tanh((V - E_{pre})/V_{slope}$$

whenever the membrane potential V is larger than the threshold E_{pre}. The model has 1 variable:

- `g`: conductance of `scalar` type

The parameters are:

- `Epre`: Presynaptic threshold potential
- `Vslope`: Activation slope of graded release

`simCodeEvnt` is:

```
" $(addtoinSyn) = $(g)* tanh(($(V_pre)-($(Epre)))*DT*2/$(Vslope));\n\
  $(updatelinsyn);\n"
```

`evntThreshold` is:

```
" $(V_pre) > $(Epre)"
```

Note

The pre-synaptic variables are referenced with the suffix `_pre` in synapse related code such as an `evnt↩Threshold`. Users can also access post-synaptic neuron variables using the suffix `_post`.

6.5.4 LEARN1SYNAPSE (Learning Synapse with a Primitive Role)

This is a simple STDP rule including a time delay for the finite transmission speed of the synapse, defined as a piecewise function.

The model has 2 variables:

- `g`: conductance of `scalar` type
- `gRaw`: raw conductance of `scalar` type

Parameters are:

- `Epre`: Presynaptic threshold potential
- `tLrn`: Time scale of learning changes
- `tChng`: Width of learning window
- `tDecay`: Time scale of synaptic strength decay
- `tPunish10`: Time window of suppression in response to 1/0

- `tPunish01`: Time window of suppression in response to 0/1
- `gMax`: Maximal conductance achievable
- `gMid`: Midpoint of sigmoid g filter curve
- `gSlope`: Slope of sigmoid g filter curve
- `tauShift`: Shift of learning curve
- `gSyn0`: Value of syn conductance g decays to

For more details about these built-in synapse models, see [2].

6.5.5 LEARN1SYNAPSE (Learning Synapse with a Primitive Role)

If users want to define their own models, they can add a new `weightUpdateModel` that includes the variables, parameters, and update codes as desired, and then push this object in the `weightUpdateModels` vector. The model can be used by referring to its index in the `weightUpdateModels` vector when adding a new population by with a call to `addSynapsePopulation`.

6.5.6 Conductance definition methods

The available options work as follows:

- `INDIVIDUALG`: When this option is chosen in the `addSynapsePopulation` command, GeNN reserves an array of size n_pre x n_post float for individual conductance values for each combination of pre and postsynaptic neuron. The actual values of the conductances are passed at runtime from the user side code, using the `pushXXXXXToDevice` function, where XXXX is the name of the synapse population.
- `GLOBALG`: When this option is chosen, the value of the variables of the synapse model (including its conductance) is taken to be the initial value provided for the synapse model's variables. This option can only be sensibly combined with connectivity type ALLTOALL.
- `INDIVIDUALID`: When this option is chosen, GeNN expects to use the same maximal conductance for all existing synaptic connections but which synapses exist will be defined at runtime from the user side code, provided as a binary array (see Insect olfaction model).

6.5.7 Connectivity types

If `INDIVIDUALG` is used with `ALLTOALL` or `DENSE` connectivity (these are equivalent in this case), synapse variables are stored in an array of size npre $*$ npost .

If the connectivity is of SPARSE type, connectivity indices are stored in a struct named SparseProjection in order to minimize the memory equirements. The struct SparseProjection contains the following members: 1: unsigned int connN: number of connections in the population. This value is needed for allocation of arrays. The indices that correspond to these values are defined in a pre-to-post basis by the following arrays: 2: unsigned int ind, of size connN: Indices of corresponding postsynaptic neurons concatenated for each presynaptic neuron. 3: unsigned int $*$indInG, of size model.neuronN[model.synapseSource[synInd]]+1: This array defines from which index in the synapse variable array the indices in ind would correspond to the presynaptic neuron that corresponds to the index of the indInG array, with the number of connections being the size of ind. More specifically, indIng[n+1]-indIng[n] would give the number of postsynaptic connections for neuron n.

For example, consider a network of two presynaptic neurons connected to three postsynaptic neurons: 0th presynaptic neuron connected to 1st and 2nd postsynaptic neurons, the 1st presynaptic neuron connected to 0th and 2nd neurons. The struct SparseProjection should have these members, with indexing from 0:

```
ConnN = 4

ind= [1 2 0 2]

indIng= [0 2 4]
```

A synapse variable of a sparsely connected synapse will be kept in an array using this conductance for indexing. For example, a variable caled `g` will be kept in an array such as: g= [g_Pre0-Post1 g_pre0-post2 g_pre1-post0 g_pre1-post2] If there are no connections for a presynaptic neuron, then g[indIng[n]]=gp[indIng[n]+1].

See tools/gen_syns_sparse_IzhModel used in Izh_sparse project to see a working example.

6.5.8 Postsynaptic integration methods

The postSynModel defines how synaptic activation translates into an input current (or other input term for models that are not current based). It also can contain equations defining dynamics that are applied to the (summed) synaptic activation, e.g. an exponential decay over time.

A postSynModel object consists of variables, parameters, derived parameters and two strings that define the code for current generation and continuous dynamics.

- string `postSynDecay`: This code defines the continuous time dynamics of the summed presynaptic inputs at the postsynaptic neuron. This usually consists of some kind of decay function.
- string `postSyntoCurrent`: This code defines how the synaptic inputs lead to an input input current (Isyn) to the postsynaptic neuron.

There are currently 2 built-in postsynaptic integration methods:

EXPDECAY: Exponential decay. Decay time constant and reversal potential parameters are needed for this postsynaptic mechanism.

This model has no variables and two parameters:

- `tau` : Decay time constant
- `E` : Reversal potential

`tau` is used by the derived parameter `expdecay` which returns expf(-dt/tau).

IZHIKEVICH_PS: Empty postsynaptic rule to be used with Izhikevich neurons.

Previous | Top | Next

7 Tutorial 1

In this tutorial we will go through step by step instructions how to create and run a GeNN simulation starting from scratch. Normally, we recommend users to use one of the example projects as a starting point but it can be very instructive to go through the necessary steps one by one once to appreciate what parts make a GeNN simulation.

7.1 The Model Definition

In this tutorial we will use a pre-defined neuron model type (TRAUBMILES) and create a simulation of ten Hodgkin-Huxley neurons [4] without any synaptic connections. We will run this simulation on a GPU and save the results to stdout.

The first step is to write a model definition function in a model definition file. Create a new empty file `tenHH↩Model.cc` with your favourite editor, e.g.

```
>> emacs tenHHModel.cc &
```

Note

The ">>" in the example code snippets refers to a shell prompt in a unix shell, do not enter them as part of your shell commands.

The model definition file contains the definition of the network model we want to simulate. First, we need to define the simulation step size `DT` and include the GeNN model specification codes `modelSpec.h` and modelSpec.cc<tt>. Then the model definition takes the form of a function named`modelDefinition` that takes one argument, passed by reference, of type`NNmodel`. Type in your`tenHHModel.cc` file:

```
// Model definintion file tenHHModel.cc

#define DT 0.1
#include "modelSpec.h"
#include "modelSpec.cc"

void modelDefinition(NNmodel &model)
{
  // definition of tenHHModel
}
```

With this we have fixed the integration time step to `0.1` in the usual time units. The typical units in GeNN are `ms`, `mV`, `nF`, and `\form#10S`. Therefore, this defines `DT= 0.1 ms`. Now we need to fill the actual model definition.

Two standard elements to the 'modelDefinition function are initialising GeNN and setting the name of the model:

```
initGeNN();
model.setName("tenHHModel");
```

Note

The name of the model given in the setName method does not need to match the file name of the model definition file. However, we strongly recommend it and if conflicting, the file name of the model definition file will prevail.

Making the actual model definition makes use of the `addNeuronPopulation` and 'addSynapse↩ Population`member functions of the NNmodel object. The arguments to a call to`addNeuronPopulations are

- `string name`: the name of the population
- `int N`: The number of neurons in the population
- `int type`: The type of neurons in the population
- `double *p`: An array of parameter values for teh neurons in the population
- `double *ini`: An array of initial values for neuron variables

We first create the parameter and initial variable arrays,

```
// definition of tenHHModel
double p[7]= {
  7.15,          // 0 - gNa: Na conductance in muS
  50.0,          // 1 - ENa: Na equi potential in mV
  1.43,          // 2 - gK: K conductance in muS
 -95.0,          // 3 - EK: K equi potential in mV
  0.02672,       // 4 - gl: leak conductance in muS
  -63.563,       // 5 - El: leak equi potential in mV
  0.143          // 6 - Cmem: membr. capacity density in nF
};

double ini[4]= {
  -60.0,         // 0 - membrane potential V
  0.0529324,     // 1 - prob. for Na channel activation m
  0.3176767,     // 2 - prob. for not Na channel blocking h
  0.5961207      // 3 - prob. for K channel activation n
};
```

Note

The comments are obviously only for clarity, they can in principle be omitted. To avoid any confusion about the meaning of parameters and variables, however, we recommend strongly to always include comments of this type.

Having defined the parameter values and initial values we can now create the neuron population,

```
model.addNeuronPopulation("Pop1", 10, TRAUBMILES, p, ini);
```

Note

`TRAUBMILES` is a variable defined in the GeNN model specification that contains the index number of the pre-defined Traub & Miles model [4].

The model definition then needs to end on calling

```
model.finalize();
```

This completes the model definition in this example. The complete `tenHHModel.cc` file now should look like this:

```
// Model definintion file tenHHModel.cc

#define DT 0.1
#include "modelSpec.h"
#include "modelSpec.cc"

void modelDefinition(NNmodel &model)
{
  // definition of tenHHModel
  initGeNN();
  model.setName("tenHHModel");
  double p[7]= {
    7.15,          // 0 - gNa: Na conductance in muS
    50.0,          // 1 - ENa: Na equi potential in mV
    1.43,          // 2 - gK: K conductance in muS
   -95.0,          // 3 - EK: K equi potential in mV
    0.02672,       // 4 - gl: leak conductance in muS
    -63.563,       // 5 - El: leak equi potential in mV
    0.143          // 6 - Cmem: membr. capacity density in nF
  };

  double ini[4]= {
    -60.0,         // 0 - membrane potential V
    0.0529324,     // 1 - prob. for Na channel activation m
    0.3176767,     // 2 - prob. for not Na channel blocking h
    0.5961207      // 3 - prob. for K channel activation n
  };
  model.addNeuronPopulation("Pop1", 10, TRAUBMILES, p, ini);
  model.finalize();
}
```

This model definition suffices to generate code for simulating the ten Hodgkin-Huxley neurons on the a GPU or CPU. The second part of a GeNN simulation is the user code that sets up the simulation, does the data handling for input and output and generally defines the numerical experiment to be run.

7.2 User Code

For the purposes of this tutorial we will initially simply run the model for one simulated second and record the final neuron variables into a file. GeNN provides the code for simulating the model in a function called `stepTimeCP↩U()` (execution on CPU only) or `stepTimeGPU()` (execution on a GPU). To make use of this code we need to define a minimal C/C++ main function. Open a new empty file `tenHHSimulation.cc` in an editor and type

```
// tenHHModel simulation code
#include "tenHHModel.cc"
#include "tenHHModel_CODE/runner.cc"
```

```
int main()
{
  allocateMem();
  initialize();

  return 0;
}
```

This boiler plate code includes the relevant model definition file we completed earlier and the entry point to the generated code `runner.cc` in the subdirectory `tenHHModel_CODE` where GeNN deposits all generated code.

Calling `allocateMem()` allocates the memory structures for all neuron variables and `initialize()` sets the initial values and copies values to the GPU.

Now we can use the generated code to execute the integration of the neuron equations provided by GeNN. To do so, we add after `initialize();`

```
stepTimeGPU(1000.0);
```

and we need to copy the result, and output it to stdout,

```
pullPop1fromDevice();
for (int i= 0; i < 10; i++) {
  cout << VPop1[i] << " ";
  cout << mPop1[i] << " ";
  cout << hPop1[i] << " ";
  cout << nPop1[i] << endl;
}
```

`pullPop1fromDevice()` copies all relevant state variables of the `Pop1` neuron group from the GPU to the CPU main memory. Then we can output the results to stdout by looping through all 10 neurons and outputting the state variables VPop1, mPop1, hPop1, nPop1.

Note

The naming convention for variables in GeNN is the variable name defined by the neuron type, here TRAU↩ BMILES defining V, m, h, and n, followed by the population name, here `Pop1`.

This completes the user code. The complete `tenHHSimulation.cu` file shoul now look like

```
// tenHHModel simulation code
#include "tenHHModel.cc"
#include "tenHHModel_CODE/runner.cc"

int main()
{
  allocateMem();
  initialize();
  stepTimeGPU(1000.0);
  pullPop1fromDevice();
  for (int i= 0; i < 10; i++) {
    cout << VPop1[i] << " ";
    cout << mPop1[i] << " ";
    cout << hPop1[i] << " ";
    cout << nPop1[i] << endl;
  }
  return 0;
}
```

7.3 Makefile

A GeNN simulation is built with a simple Makefile. On Unix systems we typically name it `GNUmakefile`. Create this file and enter

```
EXECUTABLE      :=tenHHSimulation
SOURCES         :=tenHHSimulation.cu

include $(GENN_PATH)/userproject/include/makefile_common_gnu.mk
```

This defines that the final executable of this simulation is named tenHHSimulation and the simulation code is given in the file `tenHHSimulation.cu` that we completed above.

Now we are ready to compile and run the simulation

7.4 Making and Running the Simulation

To build the model and generate the GeNN code, type in a terminal where you are in the directory containing your `tenHHModel.cc` file,

```
>> buildmodel.sh tenHHModel
```

If your environment variables `GENN_PATH` and `CUDA_PATH` are correctly configured, you should see some compile output ending in `Model build complete ...`. Now type

```
make
```

This should compile your `tenHHSimulation` executable and you can execute it with

```
./tenHHSimulation
```

The output you obtain should look like

```
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
-63.7838 0.0350042 0.336314 0.563243
```

This completes this tutorial. You have created a GeNN model and simulated it successfully!

7.5 Adding External Input

In the example we have created so far, the neurons are not connected and do not receive input. As the TRAUB↩ MILES model is silent in such conditions, the ten neurons simply will simply rest at their resting potential. To make things more interesting, let us add a constant input to all neurons, add to the end of the `modelDefinition` function

```
model.activateDirectInput("Pop1", CONSTINP);
model.setConstInp("Pop1", 0.1);
```

This will add a constant input of 0.1 nA to all ten neurons. When run with this addition you should observe the output

```
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
-63.1468 0.0211871 0.987233 0.0423695
```

This is still not particularly interesting as we are just observing the final value of the membrane potentials. To see what is going on in the meantime, we need to copy intermediate values from the device and best save them into a file. This can be done in many ways but one sensible way of doing this is to replace the line

```
stepTimeGPU(1000.0);
```

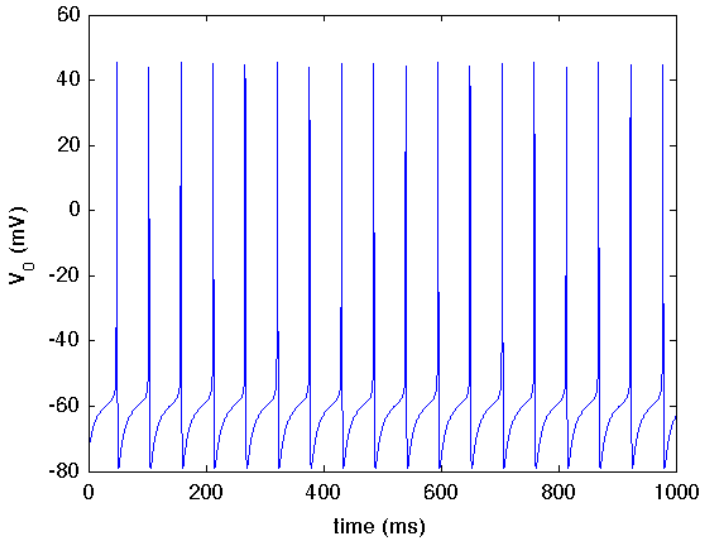
in `tenHHSimulation.cu` to something like this:

```
ofstream os("tenHH_output.V.dat");
double t= 0.0;
for (int i= 0; i < 5000; i++) {
  stepTimeGPU(0.2);
  pullPop1fromDevice();
  os << t << " ";
  for (int j= 0; j < 10; j++) {
    os << VPop1[j] << " ";
  }
  os << endl;
  t+= 0.2;
}
os.close();
```

After building, making and executing,

```
builmodel.sh tenHHModel
make clean all
./tenHHSimulation
```

there should be a file `tenHH_output.V.dat` in the same directory. If you plot column one (time) against column two (voltage of neuron 0), you should observe dynamics like this:



The completed files from this tutorial can be found in `userproject/tenHH_project`.

8 Tutorial 2

In this tutorial we will learn to add synapsePopulations to connect neurons in neuron groups to each other with synatic models. As an example we will connect the ten Hodgkin-Huxley neurons from tutorial 1 in a ring of excitatory synapses.

First, copy the files from Tutorial 1 into a new directory and rename them to new names, e.g.

```
>> cp -r tenHH_project tenHHRing_project
>> cd tenHHRing_project
>> mv tenHHModel.cc tenHHRingModel.cc
>> mv tenHHSimulation.cu tenHHRingSimulation.cu
```

Now, we need to add a synapse group to the model that allows to connect neurons from the Pop1 group to connect to other neurons of this group. Open `tenHHRingModel.cc`, change the model name inside,

```
model.setName("tenHHRing");
```

8.1 Adding Synaptic connections

Now we need additional initial values and parameters for the synapse and post-synaptic models. We will use the standard `NSYNAPSE` weightupdate model and `EXPDECAY` post-synaptic model. They need intial variables and parameters as follows:

```
double s_ini[1] = {
  0.0      // 0 - g: the synaptic conductance value
};
double *s_p= NULL;
double *ps_ini= NULL;
double ps_p[2]= {
  1.0,      // 0 - tau_S: decay time constant for S [ms]
  -80.0     // 1 - Erev: Reversal potential
};
```

If an array is not needed we set it to the NULL pointer. Here there are for example no synaptic parameters and no initial values for the post-synaptic mechanism. We can then add a synapse population at the end of the `model↩Definition(...)` function,

```
model.addSynapsePopulation("Pop1self", NSYNAPSE,
      DENSE, INDIVIDUALG, NO_DELAY, EXPDECAY, "Pop1", "Pop1", s_ini, s_p, ps_ini,
       ps_p);
```

The addSynapsePopulation parameters are

- `const char *name`: The name of the synapse population, here "Pop1self"
- `int sType`: The type of synapse to be added, we here use the predefined typse `NSYNAPSE`. See Built-in Models for all available predefined synapse types.
- `int sConn`: The type of synaptic connectivity, here `DENSE` which means we will provide a full connectivity mtrix later.
- `int gType`: The way how the synaptic conductivity g will be defined. With `GLOBALG` we indicate that all conductance are of the same conductance value, which will be the value given in `sPara`.
- `int delay`: `NO_DELAY` means that there wil be no delays for synaptic signal propagation.
- `int postSyn`: Postsynaptic integration method, we are here using the standard model of an exponential decay of synaptic excitation.
- `char *preName`: Name of the pre-synaptic neuron population, here the `Pop1` population.
- `char *postName`: Name of the post-synaptic neuron population, here also `Pop1`.
- `double *sIni`: A C-array of doubles containing initial values for the synaptic variables.
- `double *sParam`: A C-array of double precision that contains parameter values (common to all synapses of the population)
- `double *psIni`: A C-array of double precision numbers containing initial values for the post-synaptic model variables
- `double *psPara`: A C-array of double precision numbers containing parameters fo the post-snaptic model.

Adding the addSynapsePopulation command to the model definition informs GeNN that there will be synapses between the named neuron populations, here between population `Pop1` and itself. The detailed connectivity as defined by the variables `g`, we have still to define in the setup of our simulation. As always, the `modelDefinition` function ends on

```
model.finalize();
```

At this point our model definition file `tenHHRingModel.cc` should look like this

```
// Model definintion file tenHHModel.cc

#define DT 0.1
#include "modelSpec.h"
#include "modelSpec.cc"

void modelDefinition(NNmodel &model)
{
  // definition of tenHHModel
  initGeNN();
  model.setName("tenHHRingModel");
  double p[7]= {
    7.15,          // 0 - gNa: Na conductance in muS
    50.0,          // 1 - ENa: Na equi potential in mV
    1.43,          // 2 - gK: K conductance in muS
   -95.0,          // 3 - EK: K equi potential in mV
    0.02672,       // 4 - gl: leak conductance in muS
    -63.563,       // 5 - El: leak equi potential in mV
    0.143          // 6 - Cmem: membr. capacity density in nF
  };

  double ini[4]= {
    -60.0,         // 0 - membrane potential V
    0.0529324,     // 1 - prob. for Na channel activation m
    0.3176767,     // 2 - prob. for not Na channel blocking h
    0.5961207      // 3 - prob. for K channel activation n
  };
  model.addNeuronPopulation("Pop1", 10, TRAUBMILES, p, ini);
  model.activateDirectInput("Pop1", CONSTINP);
  model.setConstInp("Pop1", 0.1);
  double s_ini[1] = {
    0.0       // 0 - g: the synaptic conductance value
  };
  double *s_p= NULL;
  double *ps_ini= NULL;
  double ps_p[2]= {
    1.0,      // 0 - tau_S: decay time constant for S [ms]
    -80.0     // 1 - Erev: Reversal potential
  };
  model.addSynapsePopulation("Pop1self", NSYNAPSE,
      DENSE, INDIVIDUALG, NO_DELAY, EXPDECAY, "Pop1", "Pop1", s_ini, s_p, ps_ini,
       ps_p);
  model.finalize();
}
```

8.2 Defining the Detailed Synaptic Connections

Open the `tenHHRingSimulation.cu` file and update the file names of includes:

```
// tenHHRingModel simulation code
#include "tenHHRingModel.cc"
#include "tenHHRingModel_CODE/runner.cc"
```

Now we need to add code to generate the desired ring connectivity.

```
allocateMem();
initialize();
// define the connectivity
int pre, post;
for (int i= 0; i < 10; i++) {
  pre= i;
  post= (i+1)%10;
  gPop1self[pre*10+post]= 0.01;
}
pushPop1selftoDevice();
```

After memory allocation and initialization `gPop1self` will contain only zeros. We then assign in the loop a non-zero conductivity of 0.01 μS to all synapses from neuron `i` to `i+1` (and `9` to `0` to close the ring).

After adjusting the GNUmakefile to read

```
EXECUTABLE      :=tenHHRingSimulation
SOURCES         :=tenHHRingSimulation.cu

include $(GENN_PATH)/userproject/include/makefile_common_gnu.mk
```

we can build the model

```
>> buildmodel.sh tenHHRingModel
```

and make it

```
>> make clean all
```

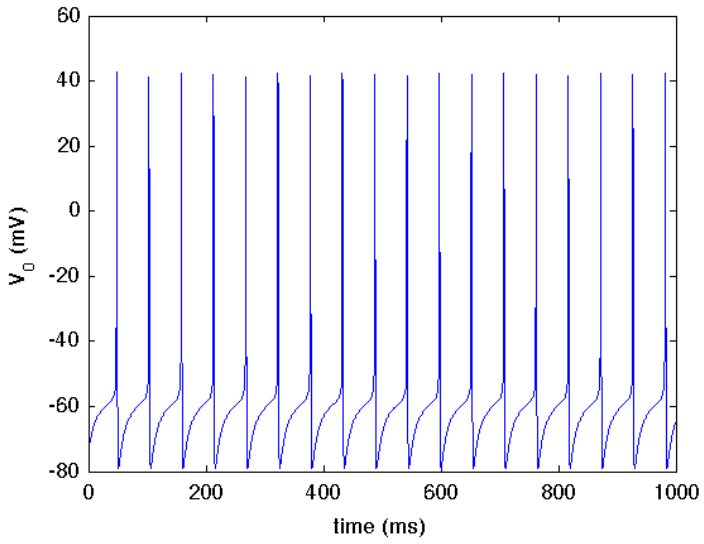
After this there should be an exectable `tenHHRingSimulation`, which can be executed,

```
>> ./tenHHRingSimulation
```

which should again result in

```
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
-64.9054 0.0147837 0.981337 0.030886
```

If we plot the content of columns one and two of `tenHHexample.V.dat` it looks very similar as in Tutorial 1



This is because the inhibitory synapses we created were all triggered at the same time so that they act during a post-synaptic spike which makes their effect all but invisible.

8.3 Setting Heterogeneous Initial Conditions

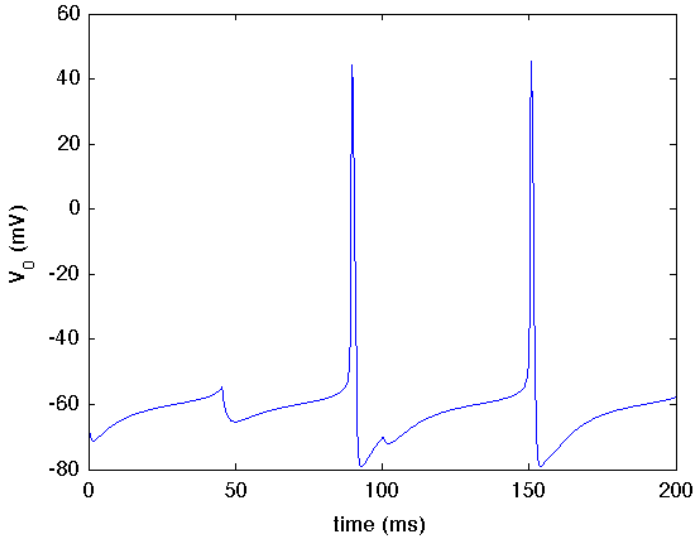
If we define different initial conditions for each of the ten neurons, i.e. add after `initialize()`,

```
// set initial variables
for (int i= 0; i < 10; i++) {
  VPop1[i]= -60.0+i;
}
pushPop1toDevice();
```

then we observe different final values for each neuron,

```
-57.3412 0.06223 0.981374 0.104417
-53.3189 0.442962 0.0664687 0.764513
-73.1413 0.00253709 0.927251 0.0277236
-67.1179 0.00927304 0.986692 0.0206106
-63.5878 0.01938 0.991071 0.0387962
-62.2114 0.0255295 0.990933 0.0504799
-61.404 0.0298902 0.990949 0.0586459
-60.6691 0.034405 0.990225 0.0668015
-59.8977 0.0397467 0.988701 0.0758159
-58.9727 0.0470178 0.98615 0.0866963
```

and zooming in on the first 200 ms, the voltage of the first neuron now looks like this



The complete codes for this tutorial are in `userproject\tenHHRing_project`.

9 Suggestions for users

GeNN generates code according to the network model defined by the user, but lets users use generated code the way they want. Here we explain in some more detail how to setup GeNN and how to use generated functions. We recommend users to also have a look at the Examples, and to follow the tutorials Tutorial 1 and Tutorial 2.

9.1 Creating and simulating a network model

The user is first expected to create an object of class NNmodel by creating the function modelDefinition() which includes calls to following methods in correct order:

- initGeNN();
- NNmodel::setName();

Then add neuron populations by:

- NNmodel::addNeuronPopulation();

for each neuron population. Add synapse populations by:

- NNmodel::addSynapsePopulation();

for each synapse population.

The modelDefinition() needs to end with calling NNmodel::finalize().

Other optional functions are explained in NNmodel class reference. At the end the function should look like this:

```
void modelDefinition(NNModel &model) {
  initGeNN();
  model.setName("YourModelName");
  model.addNeuronPopulation(...);
  ...
  model.addSynapsePopulation(...);
  ...
  model.finalize();
}
```

modelSpec.h and modelSpec.cc should be included in the file where this function is defined.

This function will be called by generateALL.cc to create corresponding CPU and GPU simulation codes under the <YourModelName>_CODE directory.

These functions can then be used in a .cu file which runs the simulation. This file should include <YourModel↩ Name>_CODE/runner.cc. Generated code differ from one model to the other, but core functions are the same and they should be called in correct order. First, the following variables should be defined and initialized:

- NNmodel model // initialized by calling modelDefinition(model)
- Array containing current input (if any)

The following are declared by GeNN but should be initialized by the user:

- Poisson neuron offset and rates (if any)
- Connectivity matrices (if sparse)
- Neuron and synapse variables (if not initialising to the homogeneous initial value provided during `model↩ Definition`)

Core functions generated by GeNN to be included in the user code include:

- allocateMem()
- deviceMemAllocate()
- initialize()
- initializeAllSparseArrays()
- convertProbabilityToRandomNumberThreshold()
- convertRateToRandomNumberThreshold()
- copyStateToDevice()
- push<neuron or synapse name>StatetoDevice()
- pull<neuron or synapse name>StatefromDevice()
- copyStateFromDevice()
- copySpikeNFromDevice()
- copySpikesFromDevice()
- stepTimeCPU() //arguments depend on model
- stepTimeGPU() //arguments depend on model

- freeMem()
- freeDeviceMem()

Copying elements from GPU to host memory is very costly in terms of performance and should only be done when needed.

9.2 Floating point precision

Double precision floating point numbers are supported by devices with compute capability 1.3 or higher. If you have an older GPU, you need to use single precision floating point in your models and simulation.

GPUs are designed to work better with single precision while double precision is the standard for CPUs. This difference should be kept in mind while comparing performance.

While setting up the network for GeNN, double precision floating point numbers are used as this part is done on the CPU. For the simulation, GeNN lets users choose between single or double precision. Overall, new variables in the generated code are defined with the precision specified by NNmodel::setPrecision(unsigned int), providing GENN_FLOAT or GENN_DOUBLE as argument. GENN_FLOAT is the default value. The keyword `scalar` can be used in the user-defined model codes for a variable that could either be single or double precision. This keyword is detected at code generation and substituted with "float" or "double" according to the precision set by NNmodel::setPrecision(unsigned int).

There may be ambiguities in arithmetic operations using explicit numbers. Standard C compilers presume that any number defined as "X" is an integer and any number defined as "X.Y" is a double. Make sure to use the same precision in your operations in order to avoid performance loss.

9.3 Working with variables in GeNN

9.3.1 Model variables

User-defined model variables originate from core units such as neuronModel, weightUpdateModel or postSynModel objects. The name of a variable is defined when the model type is introduced, i.e. with a statement such as

```
neuronModel model;
model.varNames.push_back(String("x"));
model.varTypes.push_back(String("double"));
...
int myModel= nModels.size();
nModels.push_back(model);
```

This declares that whenever the defined model type of cardinal number `myModel` is used, there will be a variable of core name `x`. varType can be of `scalar` type (see Floating point precision). The full GeNN name of this variable is obtained by directly concatenating the core name with the name of the neuron population in which the model type has been used, i.e. after a definition

```
networkModel.addNeuronPopulation("EN", n, myModel, ...);
```

there will be a variable `xEN` of type `double*` available in the global namespace of the simulation program. GeNN will pre-allocate this C array to the correct size of elements corresponding to the size of the neuron population, `n` in the example above. GeNN will also free these variables when the provided function `freeMem()` is called. Users can otherwise manipulate these variable arrays as they wish. For convenience, GeNN provides functions `pullXXStatefromDevice()` and `pushXXStatetoDevice()` to copy the variables associated to a neuron population `XX` from the device into host memory and vice versa. E.g.

```
pullENStatefromDevice();
```

would copy the C array xEN from device memory into host memory (and any other variables that the neuron type of the population EN may have).

The user can also directly use CUDA memory copy commands independent of the provided convenience functions. The relevant device pointers for all variables that exist in host memory have the same name as the host variable but are prefixed with d_. For example, the copy command that would be contained in pullENStatefromDevice() will look like

```
unsigned int size;
size = sizeof(double) * nEN;
cudaMemcpy(xEN, d_xEN, size, cudaMemcpyDeviceToHost);
```

where nEN is an integer containing the population size of the EN neuron population.

The same convention as for neuron variables applies for the variables of synapse populations, both for those originating from weightupdate models and from post-synaptic models, e.g. the variables in type NSYNAPSE contain the variable g of type float. Then, after

```
networkModel.addSynapsePopulation("ENIN", NSYNAPSE, ...);
```

there will be a global variable of type float* with the name gENIN that is pre-allocated to the right size. There will also be a matching device pointer with the name d_gENIN.

Note

The content of gENIN needs to be interpreted differently for DENSE connectivity and sparse matrix based SPARSE connectivity representations. For DENSE connectivity gENIN would contain "n_pre" times "n_post" elements, ordered along the pre-synaptic neurons as the major dimension, i.e. the value of gENIN for the ith pre-synaptic neuron and the jth post-synaptic neuron would be gENIN[i*n_post+j]. The arrangement of values in the SPARSE representation is explained in section Connectivity types
Be aware that the above naming conventions do assume that variables from the weightupdate models and the postSynModels that are used together in a synapse population are unique. If both the weightupdate model and the postSynModel have a variable of the same name, the behaviour is undefined.

9.3.2 Built-in Variables in GeNN

Since GeNN 2.0, there are no more explicitly hard-coded synapse and neuron variables. Users are free to name the variable of their models as they want. However, there are some reserved variables that are used for intermediary calculations and communication between different parts of the generated code. They can be used in the user defined code but no other variables should be defined with these names.

- DT : Time step (typically in ms) for simulation; Neuron integration can be done in multiple sub-steps inside the neuron model for numerical stability (see Traub-Miles and Izhikevich neuron model variations in Neuron models).
- addtoinSyn : This variable is used by weightUpdateModel for updating synaptic input. The way it is modified is defined in weightUpdateModel.simCode or weightUpdateModel.simCodeEvnt, therefore if a user defines her own model she should update this variable to contain the input to the post-synaptic model.
- updatelinsyn : At the end of the synaptic update by addtoinSyn, final values are copied back to the d_inSyn<synapsePopulation> variables which will be used in the next step of the neuron update to provide the input to the postsynaptic neurons. This keyword designated where the changes to addtoinSyn have been completed and it is safe to update the summed synaptic input and write back to d_inSyn<synapsePopulation> in device memory.
- inSyn: This is an intermediary synapse variable which contains the summed input into a postsynaptic neuron (originating from the addtoinSyn variables of the incoming synapses) .
- Isyn : This is a local variable which contains the (summed) input current to a neuron. It is typically the sum of any explicit current input and all synaptic inputs. The way its value is calculated during the update of the postsynaptic neuron is defined by the code provided in the postSynModel. For example, the standard EXPDECAY postsynaptic model defines ps.postSyntoCurrent= String("$(inSyn)*($(E)-$(V))"); which implements a conductance based synapse in which the postsynaptic current is given by $I_{\text{syn}} = g * s * (V_{\text{rev}} - V_{\text{post}})$.

Note

The `addtoinSyn` variables from all incoming synapses are automatically summed and added to the current value of `inSyn`.

The value resulting from the `postSyntoCurrent` code is assigned to `Isyn` and can then be used in neuron simCode like so:

```
$(V)+= (-$(V)+$(Isyn))*DT
```

- `sT` : As a neuron variable, this is the last spike time in a neuron and is automatically generated for pre and postsynaptic neuron groups of a synapse group i that follows a spike based learning rule (indicated by usesPostLearning[i]= TRUE for the ith synapse population).

In addition to these variables, neuron variables can be referred to in the synapse models by calling $(<neuronVarName>_pre) for the presynaptic neuron population, and $(<neuronVarName>_post) for the postsynaptic population. For example, $(sT_pre), $(sT_post), $(V_pre), etc.

9.4 Debugging suggestions

In Linux, users can use cuda-gdb to debug on the GPU. Example projects in userproject and lib/bin/buildmodel.sh come with a flag to enable debugging. If you are using a project with debugging, the code will be compiled with -g -G flags. In CPU mode the executable will be run in gdb, and in GPU mode it will be run in cuda-gdb in tui mode.

Note

Do not forget to switch debugging flags -g and -G off after debugging is complete as they negatively affect performance.

9.4.1

Previous | Top | Next

10 Credits

GeNN was created by Thomas Nowotny.

Addition of Izhikevich model, debugging modes and sparse connectivity by Esin Yavuz.

Block size optimisations and delayed synapses by James Turner.

Automatic brackets and dense to sparse network conversion by Alan Diamond.

User-defined synaptic and postsynaptic methods by Alex Cope and Esin Yavuz.

Example projects were provided by Alan Diamond, James Turner, Esin Yavuz and Thomas Nowotny.

Previous | Top

11 Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

classIzh 47

classol 50

CodeHelper 58

CStopWatch 59

dpclass 60

expDecayDp 61

pwSTDP 90

pwSTDP_userdef 91

rulkovdp 94

errTupel 61

inputSpec 62

neuronModel 62

neuronpop 64

NNmodel 66

Parameter 79

postSynModel 80

postVarsInPostLearn 81

postVarsInPostLearn_sparse 81

postVarsInSimCode 82

postVarsInSimCode_sparse 83

postVarsInSynapseDynamics 84

postVarsInSynapseDynamics_sparse 84

preVarsInPostLearn 85

preVarsInPostLearn_sparse 86

preVarsInSimCode 87

preVarsInSimCode_sparse 87

preVarsInSynapseDynamics 88

preVarsInSynapseDynamics_sparse 89

randomGauss 92

randomGen 93

Schmuker2014_classifier 94

stdRG 100

stopWatch 101

SynDelay 101

weightUpdateModel 102

12 Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

classIzh 47

classol
This class cpontains the methods for running the MBody1 example model 50

CodeHelper 58

CStopWatch 59

dpclass 60

errTupel 61

expDecayDp
Class defining the dependent parameter for exponential decay 61

inputSpec 62

neuronModel
Class (struct) for specifying a neuron model 62

neuronpop 64

NNmodel
Structure to hold the information that defines synapse dynamics (a model of how synapse variables change over time, independent of or in addition to changes when spikes occur) 66

Parameter 79

postSynModel
Structure to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn" 80

postVarsInPostLearn 81

postVarsInPostLearn_sparse 81

postVarsInSimCode 82

postVarsInSimCode_sparse 83

postVarsInSynapseDynamics 84

postVarsInSynapseDynamics_sparse 84

preVarsInPostLearn 85

preVarsInPostLearn_sparse 86

preVarsInSimCode 87

preVarsInSimCode_sparse 87

preVarsInSynapseDynamics 88

preVarsInSynapseDynamics_sparse 89

pwSTDP
TODO This class definition may be code-generated in a future release 90

pwSTDP_userdef
TODO This class definition may be code-generated in a future release 91

randomGauss
Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution 92

randomGen
Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers 93

rulkovdp
Class defining the dependent parameters of teh Rulkov map neuron 94

Schmuker2014_classifier
This class cpontains the methods for running the Schmuker_2014_classifier example model 94

stdRG 100

stopWatch 101

SynDelay 101

weightUpdateModel
Structure to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events 102

13 File Index

13.1 File List

Here is a list of all files with brief descriptions:

MBody1_project/model/classol_sim.cu 104

MBody_delayedSyn_project/model/classol_sim.cu 105

MBody_individualID_project/model/classol_sim.cu 105

MBody_userdef_project/model/classol_sim.cu 105

MBody1_project/model/classol_sim.h 106

MBody_delayedSyn_project/model/classol_sim.h 107

MBody_individualID_project/model/classol_sim.h 108

MBody_userdef_project/model/classol_sim.h 109

CodeHelper.cc 110

ensureFtype.h 111

experiment.cu 111

experiment.h 113

extra_neurons.h 114

lib/include/extra_postsynapses.h 116

SpineML_to_GeNN/extra_postsynapses.h 117

extra_weightupdates.h 118

GA.cc 118

gauss.cc
Contains the implementation of the Gaussian random number generator class randomGauss 118

gauss.h
Random number generator for Gaussian random variable with mean 0 and standard deviation 1 119

gen_input_structured.cc 119

gen_kcdn_syns.cc
This file is part of a tool chain for running the classol/MBody1 example model 120

gen_kcdn_syns_fixto10K.cc 120

gen_pnkc_syns.cc
This file is part of a tool chain for running the classol/MBody1 example model 121

gen_pnkc_syns_indivID.cc
This file is part of a tool chain for running the classol/MBody1 example model 122

gen_pnlhi_syns.cc
This file is part of a tool chain for running the classol/MBody1 example model 122

gen_syns_sparse.cc
This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity 123

gen_syns_sparse_izhModel.cc
This file is part of a tool chain for running the Izhikevich network model 124

HHVclampGA_project/generate_run.cc 126

Izh_sparse_project/generate_run.cc 126

MBody1_project/generate_run.cc 127

MBody_delayedSyn_project/generate_run.cc 128

MBody_individualID_project/generate_run.cc 129

MBody_userdef_project/generate_run.cc 130

OneComp_project/generate_run.cc 131

Poissonlzh_project/generate_run.cc 131

generateALL.cc
Main file combining the code for code generation. Part of the code generation section 132

generateCPU.cc
Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section 133

generateKernels.cc
Contains functions that generate code for CUDA kernels. Part of the code generation section 134

generateRunner.cc
Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section 136

GeNNHelperKrnls.cu 136

global.h
Global header file containing a few global variables. Part of the code generation section 137

helper.h 139

HHVClamp.cc
This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques 141

HHVClampParameters.h 142

hr_time.cpp
This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock 142

hr_time.h
This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock 143

lzh_sim_sparse.cu 143

lzh_sparse.cc 143

lzh_sparse_model.cc 145

lzh_sparse_model.h 145

lzh_sparse_sim.h 146

MBody1_project/model/map_classol.cc 146

MBody_delayedSyn_project/model/map_classol.cc 147

MBody_individualID_project/model/map_classol.cc 147

MBody_userdef_project/model/map_classol.cc 147

MBody1_project/model/map_classol.h 148

MBody_delayedSyn_project/model/map_classol.h 148

MBody_individualID_project/model/map_classol.h 148

MBody_userdef_project/model/map_classol.h 148

HHVclampGA_project/model/MBody1.cc 148

MBody1_project/model/MBody1.cc 151

MBody_delayedSyn.cc
This file contains the model definition of the mushroom body "MBody_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim) 155

MBody_individualID.cc
This file contains the model definition of the mushroom body "MBody_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC 159

MBody_userdef.cc
This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol_sim) 163

Model_Schmuker_2014_classifier.cc 167

modelSpec.cc 168

modelSpec.h
Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections 168

OneComp.cc 174

OneComp_model.cc 175

OneComp_model.h 175

OneComp_sim.cu 175

OneComp_sim.h 176

PoissonIzh-model.cc 176

PoissonIzh-model.h 177

PoissonIzh.cc 177

PoissonIzh_sim.cu 179

PoissonIzh_sim.h 179

postVarsInPostLearn.cc 180

postVarsInPostLearn_sparse.cc 181

postVarsInSimCode.cc 183

postVarsInSimCode_sparse.cc 184

postVarsInSynapseDynamics.cc 185

postVarsInSynapseDynamics_sparse.cc 186

preVarsInPostLearn.cc 187

preVarsInPostLearn_sparse.cc 188

preVarsInSimCode.cc 189

preVarsInSimCode_sparse.cc 190

preVarsInSynapseDynamics.cc 191

preVarsInSynapseDynamics_sparse.cc 192

randomGen.cc
Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand() 193

randomGen.h
Header file containing the class definition for a uniform random generator based on the ISAAC random number generator 194

Schmuker2014_classifier.cu 194

Schmuker2014_classifier.h
Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014" 195

simpleBit.h
Contains three macros that allow simple bit manipulations on an (presumably unsigned) 32 bit integer 195

sizes.h 196

sparseUtils.cc 196

stringutils.h 197

SynDelay.cc 199

SynDelaySim.cu 201

SynDelaySim.h 202

postsynapticVariables/testHelper.h 202

presynapticVariables/testHelper.h 202

testPostVarsInPostLearn.cu 203

testPostVarsInPostLearn.h 203

testPostVarsInPostLearn_sparse.cu 204

testPostVarsInPostLearn_sparse.h 204

testPostVarsInSimCode.cu 205

testPostVarsInSimCode.h 205

testPostVarsInSimCode_sparse.cu 206

testPostVarsInSimCode_sparse.h 206

testPostVarsInSynapseDynamics.cu 207

testPostVarsInSynapseDynamics.h 207

testPostVarsInSynapseDynamics_sparse.cu 208

testPostVarsInSynapseDynamics_sparse.h 208

testPreVarsInPostLearn.cu 209

testPreVarsInPostLearn.h 209

testPreVarsInPostLearn_sparse.cu 210

testPreVarsInPostLearn_sparse.h 210

testPreVarsInSimCode.cu 211

testPreVarsInSimCode.h 211

testPreVarsInSimCode_sparse.cu 212

testPreVarsInSimCode_sparse.h 212

testPreVarsInSynapseDynamics.cu 213

testPreVarsInSynapseDynamics.h 213

testPreVarsInSynapseDynamics_sparse.cu 214

testPreVarsInSynapseDynamics_sparse.h 214

toString.h
Contains a template function for string conversion from const char* to C++ string 214

utils.h
This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models 215

VClampGA.cu
Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU 218

VClampGA.h 219

14 Class Documentation

14.1 classIzh Class Reference

`#include <Izh_sparse_model.h>`

Public Member Functions

- classIzh ()
- ~classIzh ()
- void init (unsigned int)

- void allocate_device_mem_patterns ()
- void allocate_device_mem_input ()
- void copy_device_mem_input ()
- void read_sparsesyns_par (int, struct SparseProjection, FILE *, FILE *, FILE *, scalar *)
- void gen_alltoall_syns (scalar *, unsigned int, unsigned int, scalar)
- void free_device_mem ()
- void write_input_to_file (FILE *)
- void read_input_values (FILE *)
- void create_input_values ()
- void run (double, unsigned int)
- void getSpikesFromGPU ()

  Method for copying all spikes of the last time step from the GPU.
- void getSpikeNumbersFromGPU ()

  Method for copying the number of spikes in all neuron populations that have occurred during the last time step.
- void output_state (FILE *, unsigned int)
- void output_spikes (FILE *, unsigned int)
- void output_params (FILE *, FILE *)
- void sum_spikes ()
- void setInput (unsigned int)
- void randomizeVar (scalar *, scalar, unsigned int)
- void randomizeVarSq (scalar *, scalar, unsigned int)
- void initializeAllVars (unsigned int)

Public Attributes

- NNmodel model
- scalar * input1
- scalar * input2
- scalar * d_input1
- scalar * d_input2
- unsigned int sumPExc
- unsigned int sumPInh

14.1.1 Constructor & Destructor Documentation

14.1.1.1 classlzh::classlzh ( )

14.1.1.2 classlzh::~classlzh ( )

14.1.2 Member Function Documentation

14.1.2.1 void classlzh::allocate_device_mem_input ( )

14.1.2.2 void classlzh::allocate_device_mem_patterns ( )

14.1.2.3 void classlzh::copy_device_mem_input ( )

14.1.2.4 void classlzh::create_input_values ( )

14.1.2.5 void classlzh::free_device_mem ( )

14.1.2.6 void classlzh::gen_alltoall_syns (scalar * *g,* unsigned int *nPre,* unsigned int *nPost,* scalar *gscale*)

Parameters

gscale	Generate random conductivity values for an all to all network

14.1.2.7 void classlzh::getSpikeNumbersFromGPU ( )

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.

14.1.2.8 void classlzh::getSpikesFromGPU ( )

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

14.1.2.9 void classlzh::init (unsigned int *which*)

14.1.2.10 void classlzh::initializeAllVars (unsigned int *which*)

14.1.2.11 void classlzh::output_params (FILE * *f,* FILE * *f2*)

14.1.2.12 void classlzh::output_spikes (FILE * *f,* unsigned int *which*)

14.1.2.13 void classlzh::output_state (FILE * *f,* unsigned int *which*)

14.1.2.14 void classlzh::randomizeVar (scalar * *Var,* scalar *strength,* unsigned int *neuronGrp*)

14.1.2.15 void classlzh::randomizeVarSq (scalar * *Var,* scalar *strength,* unsigned int *neuronGrp*)

14.1.2.16 void classlzh::read_input_values (FILE *)

14.1.2.17 void classlzh::read_sparsesyns_par (int *synInd,* struct SparseProjection *C,* FILE * *f_ind,* FILE * *f_indInG,* FILE * *f_g,* scalar * *g*)

Parameters

g	File handle for a file containing sparse conductivity values

14.1.2.18 void classlzh::run (double *runtime,* unsigned int *which*)

14.1.2.19 void classlzh::setInput (unsigned int *which*)

14.1.2.20 void classlzh::sum_spikes ( )

14.1.2.21 void classlzh::write_input_to_file (FILE * *f*)

14.1.3 Member Data Documentation

14.1.3.1 scalar* classlzh::d_input1

14.1.3.2 scalar * classlzh::d_input2

14.1.3.3 scalar* classlzh::input1

14.1.3.4 scalar * classlzh::input2

14.1.3.5 NNmodel classlzh::model

14.1.3.6 unsigned int classlzh::sumPExc

14.1.3.7 unsigned int classlzh::sumPInh

The documentation for this class was generated from the following files:

- lzh_sparse_model.h
- lzh_sparse_model.cc

14.2 classol Class Reference

This class cpontains the methods for running the MBody1 example model.

`#include <map_classol.h>`

Public Member Functions

- classol ()
- ~classol ()
- void init (unsigned int)

  Method for initialising variables.
- void allocate_device_mem_patterns ()

  Method for allocating memory on the GPU device to hold the input patterns.
- void free_device_mem ()

  Methods for unallocating the memory for input patterns on the GPU device.
- void read_pnkcsyns (FILE *)

  Method for reading the connectivity between PNs and KCs from a file.
- void read_sparsesyns_par (int, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)
- void write_pnkcsyns (FILE *)

  Method for writing the conenctivity between PNs and KCs back into file.
- void read_pnlhisyns (FILE *)

  Method for reading the connectivity between PNs and LHIs from a file.
- void write_pnlhisyns (FILE *)

  Method for writing the connectivity between PNs and LHIs to a file.
- void read_kcdnsyns (FILE *)

  Method for reading the connectivity between KCs and DNs (detector neurons) from a file.
- void write_kcdnsyns (FILE *)

  Method to write the connectivity between KCs and DNs (detector neurons) to a file.
- void read_input_patterns (FILE *)

  Method for reading the input patterns from a file.
- void generate_baserates ()

  Method for calculating the baseline rates of the Poisson input neurons.
- void runGPU (scalar)

  Method for simulating the model for a given period of time on the GPU.
- void runCPU (scalar)

  Method for simulating the model for a given period of time on the CPU.
- void output_state (FILE *, unsigned int)

  Method for copying from device and writing out to file of the entire state of the model.
- void getSpikesFromGPU ()

  Method for copying all spikes of the last time step from the GPU.
- void getSpikeNumbersFromGPU ()

  Method for copying the number of spikes in all neuron populations that have occurred during the last time step.
- void output_spikes (FILE *, unsigned int)

  Method for writing the spikes occurred in the last time step to a file.

- void sum_spikes ()

  Method for summing up spike numbers.
- void get_kcdnsyns ()

  Method for copying the synaptic conductances of the learning synapses between KCs and DNs (detector neurons) back to the CPU memory.
- classol ()
- ~classol ()
- void init (unsigned int)
- void allocate_device_mem_patterns ()
- void free_device_mem ()
- void read_pnkcsyns (FILE *)
- void read_sparsesyns_par (int, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)
- void write_pnkcsyns (FILE *)
- void read_pnlhisyns (FILE *)
- void write_pnlhisyns (FILE *)
- void read_kcdnsyns (FILE *)
- void write_kcdnsyns (FILE *)
- void read_input_patterns (FILE *)
- void generate_baserates ()
- void run (scalar, unsigned int)

  Method for simulating the model for a given period of time.
- void output_state (FILE *, unsigned int)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void output_spikes (FILE *, unsigned int)
- void sum_spikes ()
- void get_kcdnsyns ()
- classol ()
- ~classol ()
- void init (unsigned int)
- void allocate_device_mem_patterns ()
- void free_device_mem ()
- void read_pnkcsyns (FILE *)
- void read_sparsesyns_par (int, struct SparseProjection, scalar *, FILE *, FILE *, FILE *)
- void write_pnkcsyns (FILE *)
- void read_pnlhisyns (FILE *)
- void write_pnlhisyns (FILE *)
- void read_kcdnsyns (FILE *)
- void write_kcdnsyns (FILE *)
- void read_input_patterns (FILE *)
- void generate_baserates ()
- void run (scalar, unsigned int)
- void output_state (FILE *, unsigned int)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void output_spikes (FILE *, unsigned int)
- void sum_spikes ()
- void get_kcdnsyns ()
- classol ()
- ~classol ()
- void init (unsigned int)
- void allocate_device_mem_patterns ()
- void free_device_mem ()
- void read_pnkcsyns (FILE *)

- template<class DATATYPE >
  void read_sparsesyns_par (DATATYPE *, int, struct SparseProjection, FILE *, FILE *, FILE *)
- void write_pnkcsyns (FILE *)
- void read_pnlhisyns (FILE *)
- void write_pnlhisyns (FILE *)
- void read_kcdnsyns (FILE *)
- void write_kcdnsyns (FILE *)
- void read_input_patterns (FILE *)
- void generate_baserates ()
- void runGPU (scalar)
- void runCPU (scalar)
- void output_state (FILE *, unsigned int)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void output_spikes (FILE *, unsigned int)
- void sum_spikes ()
- void get_kcdnsyns ()
- classol ()
- ~classol ()
- void init (unsigned int)
- void allocate_device_mem_input ()
- void free_device_mem ()
- void read_PNlzh1syns (scalar *, FILE *)
- void read_sparsesyns_par (int, struct SparseProjection, FILE *, FILE *, FILE *, double *)
- void generate_baserates ()
- void run (float, unsigned int)
- void output_state (FILE *, unsigned int)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void output_spikes (FILE *, unsigned int)
- void sum_spikes ()

Public Attributes

- NNmodel model
- unsigned int offset
- uint64_t * theRates
- scalar * p_pattern
- uint64_t * pattern
- uint64_t * baserates
- uint64_t * d_pattern
- uint64_t * d_baserates
- unsigned int sumPN
- unsigned int sumKC
- unsigned int sumLHI
- unsigned int sumDN
- unsigned int size_g
- unsigned int sumlzh1

14.2.1 Detailed Description

This class cpontains the methods for running the MBody1 example model.

This class cpontains the methods for running the MBody_delayedSyn example model.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 classol::classol ( )

14.2.2.2 classol::~classol ( )

14.2.2.3 classol::classol ( )

14.2.2.4 classol::~classol ( )

14.2.2.5 classol::classol ( )

14.2.2.6 classol::~classol ( )

14.2.2.7 classol::classol ( )

14.2.2.8 classol::~classol ( )

14.2.2.9 classol::classol ( )

14.2.2.10 classol::~classol ( )

14.2.3 Member Function Documentation

14.2.3.1 void classol::allocate_device_mem_input ( )

14.2.3.2 void classol::allocate_device_mem_patterns ( )

Method for allocating memory on the GPU device to hold the input patterns.

14.2.3.3 void classol::allocate_device_mem_patterns ( )

14.2.3.4 void classol::allocate_device_mem_patterns ( )

14.2.3.5 void classol::allocate_device_mem_patterns ( )

14.2.3.6 void classol::free_device_mem ( )

14.2.3.7 void classol::free_device_mem ( )

14.2.3.8 void classol::free_device_mem ( )

14.2.3.9 void classol::free_device_mem ( )

Methods for unallocating the memory for input patterns on the GPU device.

14.2.3.10 void classol::free_device_mem ( )

14.2.3.11 void classol::generate_baserates ( )

14.2.3.12 void classol::generate_baserates ( )

14.2.3.13 void classol::generate_baserates ( )

Method for calculating the baseline rates of the Poisson input neurons.

14.2.3.14 void classol::generate_baserates ( )

14.2.3.15 void classol::generate_baserates ( )

14.2.3.16 void classol::get_kcdnsyns ( )

14.2.3.17 void classol::get_kcdnsyns ( )

14.2.3.18 void classol::get_kcdnsyns ( )

Method for copying the synaptic conductances of the learning synapses between KCs and DNs (detector neurons) back to the CPU memory.

14.2.3.19 void classol::get_kcdnsyns ( )

14.2.3.20 void classol::getSpikeNumbersFromGPU ( )

14.2.3.21 void classol::getSpikeNumbersFromGPU ( )

14.2.3.22 void classol::getSpikeNumbersFromGPU ( )

14.2.3.23 void classol::getSpikeNumbersFromGPU ( )

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.

14.2.3.24 void classol::getSpikeNumbersFromGPU ( )

14.2.3.25 void classol::getSpikesFromGPU ( )

14.2.3.26 void classol::getSpikesFromGPU ( )

14.2.3.27 void classol::getSpikesFromGPU ( )

14.2.3.28 void classol::getSpikesFromGPU ( )

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

14.2.3.29 void classol::getSpikesFromGPU ( )

14.2.3.30 void classol::init (unsigned *int*)

14.2.3.31 void classol::init (unsigned int *which*)

Method for initialising variables.

Parameters

which	Flag defining whether GPU or CPU only version is run

14.2.3.32 void classol::init (unsigned *int*)

14.2.3.33 void classol::init (unsigned *int*)

14.2.3.34 void classol::init (unsigned *int*)

14.2.3.35 void classol::output_spikes (FILE * , unsigned *int*)

14.2.3.36 void classol::output_spikes (FILE * , unsigned *int*)

14.2.3.37 void classol::output_spikes (FILE * , unsigned *int*)

14.2.3.38 void classol::output_spikes (FILE * *f,* unsigned int *which*)

Method for writing the spikes occurred in the last time step to a file.

Parameters

f	File handle for a file to write spike times to
which	Flag determining whether using GPU or CPU only

14.2.3.39 void classol::output_spikes (FILE * , unsigned *int*)

14.2.3.40 void classol::output_state (FILE * , unsigned *int*)

14.2.3.41 void classol::output_state (FILE * , unsigned *int*)

14.2.3.42 void classol::output_state (FILE * , unsigned *int*)

14.2.3.43 void classol::output_state (FILE * *f,* unsigned int *which*)

Method for copying from device and writing out to file of the entire state of the model.

Parameters

f	File handle for a file to write the model state to
which	Flag determining whether using GPU or CPU only

14.2.3.44 void classol::output_state (FILE * , unsigned *int*)

14.2.3.45 void classol::read_input_patterns (FILE * *f*)

Method for reading the input patterns from a file.

Parameters

f	File handle for a file containing input patterns

14.2.3.46 void classol::read_input_patterns (FILE *)

14.2.3.47 void classol::read_input_patterns (FILE *)

14.2.3.48 void classol::read_input_patterns (FILE *)

14.2.3.49 void classol::read_kcdnsyns (FILE * *f*)

Method for reading the connectivity between KCs and DNs (detector neurons) from a file.

Parameters

f	File handle for a file containing KC to DN (detector neuron) conductivity values

14.2.3.50 void classol::read_kcdnsyns (FILE *)

14.2.3.51 void classol::read_kcdnsyns (FILE *)

14.2.3.52 void classol::read_kcdnsyns (FILE *)

14.2.3.53 void classol::read_PNlzh1syns (scalar * *gp,* FILE * *f*)

14.2.3.54 void classol::read_pnkcsyns (FILE *)

14.2.3.55 void classol::read_pnkcsyns (FILE * *f*)

Method for reading the connectivity between PNs and KCs from a file.

Parameters

f	File handle for a file containing PN to KC conductivity values

14.2.3.56 void classol::read_pnkcsyns (FILE *)

14.2.3.57 void classol::read_pnkcsyns (FILE *)

14.2.3.58 void classol::read_pnlhisyns (FILE * *f*)

Method for reading the connectivity between PNs and LHIs from a file.

Parameters

f	File handle for a file containing PN to LHI conductivity values

14.2.3.59 void classol::read_pnlhisyns (FILE *)

14.2.3.60 void classol::read_pnlhisyns (FILE *)

14.2.3.61 void classol::read_pnlhisyns (FILE *)

14.2.3.62 void classol::read_sparsesyns_par (int *synInd,* struct SparseProjection *C,* FILE * *f_ind,* FILE * *f_indInG,* FILE * *f_g,* double * *g*)

Parameters

g	File handle for a file containing sparse conductivity values

14.2.3.63 void classol::read_sparsesyns_par (int *synInd,* struct SparseProjection *C,* scalar * *g,* FILE * *f_ind,* FILE * *f_indInG,* FILE * *f_g*)

Parameters

f_g	File handle for a file containing sparse connectivity values

14.2.3.64 void classol::read_sparsesyns_par (int , struct SparseProjection , scalar * , FILE * , FILE * , FILE *)

14.2.3.65 void classol::read_sparsesyns_par (int , struct SparseProjection , scalar * , FILE * , FILE * , FILE *)

14.2.3.66 template<class DATATYPE > void classol::read_sparsesyns_par (DATATYPE * *wuvar,* int *synInd,* struct SparseProjection *C,* FILE * *f_ind,* FILE * *f_indInG,* FILE * *f_g*)

Parameters

f_g	File handle for a file containing sparse conductivity values

14.2.3.67 void classol::run (float *runtime,* unsigned int *which*)

14.2.3.68 void classol::run (scalar *runtime,* unsigned int *which*)

Method for simulating the model for a given period of time.

Parameters

runtime	Duration of time to run the model for

which	Flag determining whether to run on GPU or CPU only

14.2.3.69 void classol::run (scalar , unsigned *int*)

14.2.3.70 void classol::runCPU (scalar *runtime*)

Method for simulating the model for a given period of time on the CPU.

Method for simulating the model for a given period of time on th CPU.

Parameters

runtime	Duration of time to run the model for

14.2.3.71 void classol::runCPU (scalar)

14.2.3.72 void classol::runGPU (scalar *runtime*)

Method for simulating the model for a given period of time on the GPU.

Method for simulating the model for a given period of time on th GPU.

Parameters

runtime	Duration of time to run the model for

14.2.3.73 void classol::runGPU (scalar)

14.2.3.74 void classol::sum_spikes ( )

14.2.3.75 void classol::sum_spikes ( )

14.2.3.76 void classol::sum_spikes ( )

14.2.3.77 void classol::sum_spikes ( )

Method for summing up spike numbers.

14.2.3.78 void classol::sum_spikes ( )

14.2.3.79 void classol::write_kcdnsyns (FILE * *f*)

Method to write the connectivity between KCs and DNs (detector neurons) to a file.

Parameters

f	File handle for a file to write KC to DN (detectore neuron) conductivity values to

14.2.3.80 void classol::write_kcdnsyns (FILE *)

14.2.3.81 void classol::write_kcdnsyns (FILE *)

14.2.3.82 void classol::write_kcdnsyns (FILE *)

14.2.3.83 void classol::write_pnkcsyns (FILE *)

14.2.3.84 void classol::write_pnkcsyns (FILE *)

14.2.3.85 void classol::write_pnkcsyns (FILE * *f*)

Method for writing the conenctivity between PNs and KCs back into file.

Parameters

f	File handle for a file to write PN to KC conductivity values to

14.2.3.86 void classol::write_pnkcsyns (FILE *)

14.2.3.87 void classol::write_pnlhisyns (FILE *)

14.2.3.88 void classol::write_pnlhisyns (FILE * *f*)

Method for writing the connectivity between PNs and LHIs to a file.

Parameters

f	File handle for a file to write PN to LHI conductivity values to

14.2.3.89 void classol::write_pnlhisyns (FILE *)

14.2.3.90 void classol::write_pnlhisyns (FILE *)

14.2.4 Member Data Documentation

14.2.4.1 uint64_t * classol::baserates

14.2.4.2 uint64_t * classol::d_baserates

14.2.4.3 uint64_t * classol::d_pattern

14.2.4.4 NNmodel classol::model

14.2.4.5 unsigned int classol::offset

14.2.4.6 scalar * classol::p_pattern

14.2.4.7 uint64_t * classol::pattern

14.2.4.8 unsigned int classol::size_g

14.2.4.9 unsigned int classol::sumDN

14.2.4.10 unsigned int classol::sumIzh1

14.2.4.11 unsigned int classol::sumKC

14.2.4.12 unsigned int classol::sumLHI

14.2.4.13 unsigned int classol::sumPN

14.2.4.14 uint64_t * classol::theRates

The documentation for this class was generated from the following files:

- MBody1_project/model/map_classol.h
- PoissonIzh-model.h
- MBody1_project/model/map_classol.cc
- PoissonIzh-model.cc

14.3 CodeHelper Class Reference

Public Member Functions

- CodeHelper ()
- void setVerbose (bool isVerbose)
- string openBrace (unsigned int level)
- string closeBrace (unsigned int level)
- string endl ()

Public Attributes

- vector< unsigned int > braces
- bool verbose

14.3.1 Constructor & Destructor Documentation

14.3.1.1 CodeHelper::CodeHelper ( ) [inline]

14.3.2 Member Function Documentation

14.3.2.1 string CodeHelper::closeBrace (unsigned int *level*) [inline]

14.3.2.2 string CodeHelper::endl ( ) [inline]

14.3.2.3 string CodeHelper::openBrace (unsigned int *level*) [inline]

14.3.2.4 void CodeHelper::setVerbose (bool *isVerbose*) [inline]

14.3.3 Member Data Documentation

14.3.3.1 vector<unsigned int> CodeHelper::braces

14.3.3.2 bool CodeHelper::verbose

The documentation for this class was generated from the following file:

- CodeHelper.cc

14.4 CStopWatch Class Reference

```
#include <hr_time.h>
```

Public Member Functions

- CStopWatch ()
- void startTimer ()

  This method starts the timer.
- void stopTimer ()

  This method stops the timer.
- double getElapsedTime ()

  This method returns the time elapsed between start and stop of the timer in seconds.

14.4.1 Constructor & Destructor Documentation

14.4.1.1 CStopWatch::CStopWatch ( ) [inline]

14.4.2 Member Function Documentation

14.4.2.1 double CStopWatch::getElapsedTime ( )

This method returns the time elapsed between start and stop of the timer in seconds.

14.4.2.2 void CStopWatch::startTimer ( )

This method starts the timer.

14.4.2.3 void CStopWatch::stopTimer ( )

This method stops the timer.

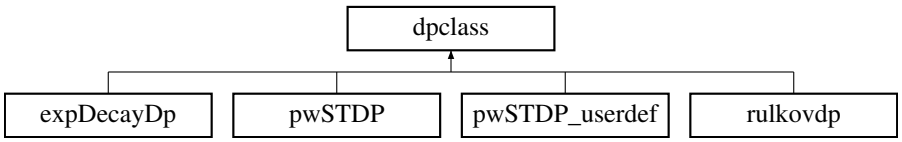
The documentation for this class was generated from the following files:

- hr_time.h
- hr_time.cpp

14.5 dpclass Class Reference

`#include <modelSpec.h>`

Inheritance diagram for dpclass:



Public Member Functions

- dpclass ()
- virtual double calculateDerivedParameter (int index, vector< double > pars, double dt=1.0)

14.5.1 Constructor & Destructor Documentation

14.5.1.1 dpclass::dpclass ( ) [inline]

14.5.2 Member Function Documentation

14.5.2.1 virtual double dpclass::calculateDerivedParameter (int *index*, vector< double > *pars*, double *dt* = 1.0) [inline], [virtual]

Reimplemented in expDecayDp, rulkovdp, pwSTDP_userdef, and pwSTDP.

The documentation for this class was generated from the following file:

- modelSpec.h

14.6 errTupel Struct Reference

Public Attributes

- unsigned int id
- double err

14.6.1 Member Data Documentation

14.6.1.1 double errTupel::err

14.6.1.2 unsigned int errTupel::id

The documentation for this struct was generated from the following file:

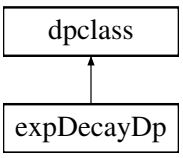
- GA.cc

14.7 expDecayDp Class Reference

Class defining the dependent parameter for exponential decay.

`#include <utils.h>`

Inheritance diagram for expDecayDp:



Public Member Functions

- double calculateDerivedParameter (int index, vector< double > pars, double dt=1.0)
- double expDecay (vector< double > pars, double dt)

14.7.1 Detailed Description

Class defining the dependent parameter for exponential decay.

14.7.2 Member Function Documentation

14.7.2.1 double expDecayDp::calculateDerivedParameter (int *index*, vector< double > *pars*, double *dt* = `1.0`) `[inline]`, `[virtual]`

Reimplemented from dpclass.

14.7.2.2 double expDecayDp::expDecay (vector< double > *pars*, double *dt*) `[inline]`

The documentation for this class was generated from the following file:

- utils.h

14.8 inputSpec Struct Reference

`#include <helper.h>`

Public Attributes

- double t
- double baseV
- int N
- vector< double > st
- vector< double > V

14.8.1 Member Data Documentation

14.8.1.1 double inputSpec::baseV

14.8.1.2 int inputSpec::N

14.8.1.3 vector<double> inputSpec::st

14.8.1.4 double inputSpec::t

14.8.1.5 vector<double> inputSpec::V

The documentation for this struct was generated from the following file:

- helper.h

14.9 neuronModel Struct Reference

class (struct) for specifying a neuron model.

`#include <modelSpec.h>`

Public Attributes

- string simCode

  Code that defines the execution of one timestep of integration of the neuron model.
- string thresholdConditionCode

  Code evaluating to a bool (e.g. "V > 20") that defines the condition for a true spike in the described neuron model.
- string resetCode

  Code that defines the reset action taken after a spike occurred. This can be empty.
- vector< string > varNames

  Names of the variables in the neuron model.
- vector< string > tmpVarNames

  never used
- vector< string > varTypes

  Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.
- vector< string > tmpVarTypes

  never used
- vector< string > pNames

  Names of (independent) parameters of the model.
- vector< string > dpNames

Names of dependent parameters of the model.

- vector< string > extraGlobalNeuronKernelParameters

  Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

- vector< string > extraGlobalNeuronKernelParameterTypes

  Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

- dpclass * dps

  Derived parameters.

- bool needPreSt

  Whether presynaptic spike times are needed or not.

- bool needPostSt

  Whether postsynaptic spike times are needed or not.

14.9.1 Detailed Description

class (struct) for specifying a neuron model.

14.9.2 Member Data Documentation

14.9.2.1 vector<string> neuronModel::dpNames

Names of dependent parameters of the model.

The dependent parameters are functions of independent parameters that enter into the neuron model. To avoid unecessary computational overhead, these parameters are calculated at compile time and inserted as explicit values into the generated code. See method NNmodel::initDerivedNeuronPara for how this is done.

14.9.2.2 dpclass* neuronModel::dps

Derived parameters.

14.9.2.3 vector<string> neuronModel::extraGlobalNeuronKernelParameters

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

14.9.2.4 vector<string> neuronModel::extraGlobalNeuronKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per neuron.

14.9.2.5 bool neuronModel::needPostSt

Whether postsynaptic spike times are needed or not.

14.9.2.6 bool neuronModel::needPreSt

Whether presynaptic spike times are needed or not.

14.9.2.7 vector<string> neuronModel::pNames

Names of (independent) parameters of the model.

14.9.2.8 string neuronModel::resetCode

Code that defines the reset action taken after a spike occurred. This can be empty.

14.9.2.9 string neuronModel::simCode

Code that defines the execution of one timestep of integration of the neuron model.

The code will refer to for the value of the variable with name "NN". It needs to refer to the predefined variable "ISYN", i.e. contain , if it is to receive input.

14.9.2.10 string neuronModel::thresholdConditionCode

Code evaluating to a bool (e.g. "V > 20") that defines the condition for a true spike in the described neuron model.

14.9.2.11 vector<string> neuronModel::tmpVarNames

never used

14.9.2.12 vector<string> neuronModel::tmpVarTypes

never used

14.9.2.13 vector<string> neuronModel::varNames

Names of the variables in the neuron model.

14.9.2.14 vector<string> neuronModel::varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this struct was generated from the following file:

- modelSpec.h

14.10 neuronpop Class Reference

```
#include <OneComp_model.h>
```

Public Member Functions

- neuronpop ()
- ~neuronpop ()
- void init (unsigned int)
- void allocate_device_mem_patterns ()
- void allocate_device_mem_input ()
- void copy_device_mem_input ()
- void write_input_to_file (FILE *)
- void read_input_values (FILE *)
- void create_input_values (float t)
- void run (float, unsigned int)
- void getSpikesFromGPU ()

  Method for copying all spikes of the last time step from the GPU.
- void getSpikeNumbersFromGPU ()

  Method for copying the number of spikes in all neuron populations that have occurred during the last time step.
- void output_state (FILE *, unsigned int)
- void output_spikes (FILE *, unsigned int)
- void sum_spikes ()

Public Attributes

- NNmodel model
- float * input1
- float * d_input1
- unsigned int sumIzh1

14.10.1 Constructor & Destructor Documentation

14.10.1.1 neuronpop::neuronpop ( )

14.10.1.2 neuronpop::~neuronpop ( )

14.10.2 Member Function Documentation

14.10.2.1 void neuronpop::allocate_device_mem_input ( )

14.10.2.2 void neuronpop::allocate_device_mem_patterns ( )

14.10.2.3 void neuronpop::copy_device_mem_input ( )

14.10.2.4 void neuronpop::create_input_values (float *t*)

14.10.2.5 void neuronpop::getSpikeNumbersFromGPU ( )

Method for copying the number of spikes in all neuron populations that have occurred during the last time step.

This method is a simple wrapper for the convenience function copySpikeNFromDevice() provided by GeNN.

14.10.2.6 void neuronpop::getSpikesFromGPU ( )

Method for copying all spikes of the last time step from the GPU.

This is a simple wrapper for the convenience function copySpikesFromDevice() which is provided by GeNN.

14.10.2.7 void neuronpop::init (unsigned int *which*)

14.10.2.8 void neuronpop::output_spikes (FILE * *f,* unsigned int *which*)

14.10.2.9 void neuronpop::output_state (FILE * *f,* unsigned int *which*)

14.10.2.10 void neuronpop::read_input_values (FILE * *f*)

14.10.2.11 void neuronpop::run (float *runtime,* unsigned int *which*)

14.10.2.12 void neuronpop::sum_spikes ( )

14.10.2.13 void neuronpop::write_input_to_file (FILE * *f*)

14.10.3 Member Data Documentation

14.10.3.1 float* neuronpop::d_input1

14.10.3.2 float* neuronpop::input1

14.10.3.3 NNmodel neuronpop::model

14.10.3.4 unsigned int neuronpop::sumIzh1

The documentation for this class was generated from the following files:

- OneComp_model.h

- OneComp_model.cc

14.11 NNmodel Class Reference

Structure to hold the information that defines synapse dynamics (a model of how synapse variables change over time, independent of or in addition to changes when spikes occur).

`#include <modelSpec.h>`

Public Member Functions

- NNmodel ()
- ~NNmodel ()
- void setName (const string)

  Method to set the neuronal network model name.
- void setPrecision (unsigned int)

  Set numerical precision for floating point.
- void setTiming (bool)

  Set whether timers and timing commands are to be included.
- void setSeed (unsigned int)

  Set the random seed (disables automatic seeding if argument not 0).
- void checkSizes (unsigned int *, unsigned int *, unsigned int *)
- void setGPUDevice (int)

  Method to choose the GPU to be used for the model. If "AUTODEVICE' (-1), GeNN will choose the device based on a heuristic rule.
- void addNeuronPopulation (const char *, unsigned int, unsigned int, double *, double *)

  Method for adding a neuron population to a neuronal network model, using C style character array for the name of the population.
- void addNeuronPopulation (const string, unsigned int, unsigned int, double *, double *)

  Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.
- void addNeuronPopulation (const string, unsigned int, unsigned int, vector< double >, vector< double >)

  Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.
- void activateDirectInput (const string, unsigned int)

  Method for activating explicit current input.
- void setConstInp (const string, double)

  Method for setting the global input value for a neuron population if CONSTINP.
- void setNeuronClusterIndex (const string neuronGroup, int hostID, int deviceID)

  Function for setting which host and which device a neuron group will be simulated on.
- void addSynapsePopulation (const string name, unsigned int syntype, unsigned int conntype, unsigned int gtype, const string src, const string trg, double *p)

  Overload of method for backwards compatibility.
- void addSynapsePopulation (const char *, unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const char *, const char *, double *, double *, double *)

  Method for adding a synapse population to a neuronal network model, using C style character array for the name of the population.
- void addSynapsePopulation (const string, unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const string, const string, double *, double *, double *)

  Overloaded version without initial variables for synapses.
- void addSynapsePopulation (const string, unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const string, const string, double *, double *, double *, double *)

  Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.
- void addSynapsePopulation (const string, unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, const string, const string, vector< double >, vector< double >, vector< double >, vector< double >)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

- void setSynapseG (const string, double)

  Method for setting the conductance (g) value for a synapse population with "GLOBALG" charactertistic.
- void setMaxConn (const string, unsigned int)

  This function defines the maximum number of connections for a neuron in the population.
- void setSynapseClusterIndex (const string synapseGroup, int hostID, int deviceID)

  Function for setting which host and which device a synapse group will be simulated on.
- void initLearnGrps ()
- void finalize ()

Public Attributes

- string name

  Name of the neuronal newtwork model.
- string ftype

  Type of floating point variables (float, double, ...; default: float)
- string RNtype

  Underlying type for random number generation (default: long)
- int final

  Flag for whether the model has been finalized.
- unsigned int needSt

  Whether last spike times are needed at all in this network model (related to STDP)
- unsigned int needSynapseDelay

  Whether delayed synapse conductance is required in the network.
- int chooseGPUDevice
- bool timing
- unsigned int seed
- unsigned int neuronGrpN

  Number of neuron groups.
- vector< string > neuronName

  Names of neuron groups.
- vector< unsigned int > neuronN

  Number of neurons in group.
- vector< unsigned int > sumNeuronN

  Summed neuron numbers.
- vector< unsigned int > padSumNeuronN

  Padded summed neuron numbers.
- vector< unsigned int > neuronPostSyn
- vector< unsigned int > neuronType

  Postsynaptic methods to the neuron.
- vector< vector< double > > neuronPara

  Parameters of neurons.
- vector< vector< double > > dnp

  Derived neuron parameters.
- vector< vector< double > > neuronIni

  Initial values of neurons.
- vector< vector< unsigned int > > inSyn

  The ids of the incoming synapse groups.
- vector< vector< unsigned int > > outSyn

  The ids of the outgoing synapse groups.
- vector< unsigned int > receivesInputCurrent

flags whether neurons of a population receive explicit input currents

- vector< bool > neuronNeedSt

  Whether last spike time needs to be saved for a group.
- vector< bool > neuronNeedTrueSpk

  Whether spike-like events from a group are required.
- vector< bool > neuronNeedSpkEvnt

  Whether spike-like events from a group are required.
- vector< vector< bool > > neuronVarNeedQueue

  Whether a neuron variable needs queueing for syn code.
- vector< string > neuronSpkEvntCondition

  Will contain the spike event condition code when spike events are used.
- vector< unsigned int > neuronDelaySlots

  The number of slots needed in the synapse delay queues of a neuron group.
- vector< int > neuronHostID

  The ID of the cluster node which the neuron groups are computed on.
- vector< int > neuronDeviceID

  The ID of the CUDA device which the neuron groups are comnputed on.
- unsigned int synapseGrpN

  Number of synapse groups.
- vector< string > synapseName

  Names of synapse groups.
- vector< unsigned int > maxConn

  Padded summed maximum number of connections for a neuron in the neuron groups.
- vector< unsigned int > padSumSynapseKrnl
- vector< unsigned int > synapseType

  Types of synapses.
- vector< unsigned int > synapseConnType

  Connectivity type of synapses.
- vector< unsigned int > synapseGType

  Type of specification method for synaptic conductance.
- vector< unsigned int > synapseSource

  Presynaptic neuron groups.
- vector< unsigned int > synapseTarget

  Postsynaptic neuron groups.
- vector< unsigned int > synapseInSynNo

  IDs of the target neurons' incoming synapse variables for each synapse group.
- vector< unsigned int > synapseOutSynNo

  The target neurons' outgoing synapse for each synapse group.
- vector< bool > synapseUsesTrueSpikes

  Defines if synapse update is done after detection of real spikes (only one point after threshold)
- vector< bool > synapseUsesSpikeEvents

  Defines if synapse update is done after detection of spike events (every point above threshold)
- vector< bool > synapseUsesPostLearning

  Defines if anything is done in case of postsynaptic neuron spiking before presynaptic neuron (punishment in STDP etc.)
- vector< vector< string > > synapseSpkEvntVars

  Defines variable names that are needed in the SpkEvnt condition and that are pre-fetched for that purpose into shared memory.
- vector< vector< double > > synapsePara

  parameters of synapses
- vector< vector< double > > synapseIni

Initial values of synapse variables.

- vector< vector< double > > dsp_w

  Derived synapse parameters (weightUpdateModel only)
- vector< unsigned int > postSynapseType

  Types of post-synaptic model.
- vector< vector< double > > postSynapsePara

  parameters of postsynapses
- vector< vector< double > > postSynIni

  Initial values of postsynaptic variables.
- vector< vector< double > > dpsp

  Derived postsynapse parameters.
- vector< double > globalInp

  Global explicit input if CONSTINP is chosen.
- unsigned int lrnGroups

  Number of synapse groups with learning.
- vector< unsigned int > padSumLearnN

  Padded summed neuron numbers of learn group source populations.
- vector< unsigned int > lrnSynGrp

  Enumeration of the IDs of synapse groups that learn.
- vector< unsigned int > synapseDelay

  Global synaptic conductance delay for the group (in time steps)
- vector< int > synapseHostID

  The ID of the cluster node which the synapse groups are computed on.
- vector< int > synapseDeviceID

  The ID of the CUDA device which the synapse groups are comnputed on.

14.11.1 Detailed Description

Structure to hold the information that defines synapse dynamics (a model of how synapse variables change over time, independent of or in addition to changes when spikes occur).

14.11.2 Constructor & Destructor Documentation

14.11.2.1 NNmodel::NNmodel ( )

14.11.2.2 NNmodel::~NNmodel ( )

14.11.3 Member Function Documentation

14.11.3.1 void NNmodel::activateDirectInput (const string *name*, unsigned int *type*)

Method for activating explicit current input.

This function defines the type of the explicit input to the neuron model. Current options are common constant input to all neurons, input from a file and input defines as a rule.

Parameters

name	Name of the neuron population
type	Type of input: 1 if common input, 2 if custom input from file, 3 if custom input as a rule

14.11.3.2 void NNmodel::addNeuronPopulation (const char * *name*, unsigned int *nNo*, unsigned int *type*, double * *p*, double * *ini*)

Method for adding a neuron population to a neuronal network model, using C style character array for the name of the population.

Parameters

name	Name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
p	Parameters of this neuron type
ini	Initial values for variables of this neuron type

14.11.3.3 void NNmodel::addNeuronPopulation (const string *name,* unsigned int *nNo,* unsigned int *type,* double * *p,* double * *ini*)

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

Parameters

name	The name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
p	Parameters of this neuron type
ini	Initial values for variables of this neuron type

14.11.3.4 void NNmodel::addNeuronPopulation (const string *name,* unsigned int *nNo,* unsigned int *type,* vector< double > *p,* vector< double > *ini*)

Method for adding a neuron population to a neuronal network model, using C++ string for the name of the population.

This function adds a neuron population to a neuronal network models, assigning the name, the number of neurons in the group, the neuron type, parameters and initial values. The latter two defined as STL vectors of double.

Parameters

name	The name of the neuron population
nNo	Number of neurons in the population
type	Type of the neurons, refers to either a standard type or user-defined type
p	Parameters of this neuron type
ini	Initial values for variables of this neuron type

14.11.3.5 void NNmodel::addSynapsePopulation (const string *name,* unsigned int *syntype,* unsigned int *conntype,* unsigned int *gtype,* const string *src,* const string *target,* double * *params*)

Overload of method for backwards compatibility.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
src	Name of the (existing!) pre-synaptic neuron population
target	Name of the (existing!) post-synaptic neuron population
params	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.

14.11.3.6 void NNmodel::addSynapsePopulation (const char * *name,* unsigned int *syntype,* unsigned int *conntype,* unsigned int *gtype,* unsigned int *delaySteps,* unsigned int *postsyn,* const char * *src,* const char * *trg,* double * *p,* double * *PSVini,* double * *ps*)

Method for adding a synapse population to a neuronal network model, using C style character array for the name of the population.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
p	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.

14.11.3.7 void NNmodel::addSynapsePopulation (const string *name,* unsigned int *syntype,* unsigned int *conntype,* unsigned int *gtype,* unsigned int *delaySteps,* unsigned int *postsyn,* const string *src,* const string *trg,* double * *p,* double * *PSVini,* double * *ps*)

Overloaded version without initial variables for synapses.

Overloaded old version.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
p	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.

14.11.3.8 void NNmodel::addSynapsePopulation (const string *name,* unsigned int *syntype,* unsigned int *conntype,* unsigned int *gtype,* unsigned int *delaySteps,* unsigned int *postsyn,* const string *src,* const string *trg,* double * *synini,* double * *p,* double * *PSVini,* double * *ps*)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

This function adds a synapse population to a neuronal network model, assigning the name, the synapse type, the connectivity type, the type of conductance specification, the source and destination neuron populations, and the synaptic parameters.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots
postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
synini	A C-type array of doubles that contains the initial values for synapse variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
p	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.

14.11.3.9 void NNmodel::addSynapsePopulation (const string *name,* unsigned int *syntype,* unsigned int *conntype,* unsigned int *gtype,* unsigned int *delaySteps,* unsigned int *postsyn,* const string *src,* const string *trg,* vector< double > *synini,* vector< double > *p,* vector< double > *PSVini,* vector< double > *ps*)

Method for adding a synapse population to a neuronal network model, using C++ string for the name of the population.

This function adds a synapse population to a neuronal network model, assigning the name, the synapse type, the connectivity type, the type of conductance specification, the source and destination neuron populations, and the synaptic parameters.

Parameters

name	The name of the synapse population
syntype	The type of synapse to be added (i.e. learning mode)
conntype	The type of synaptic connectivity
gtype	The way how the synaptic conductivity g will be defined
delaySteps	Number of delay slots

postsyn	Postsynaptic integration method
src	Name of the (existing!) pre-synaptic neuron population
trg	Name of the (existing!) post-synaptic neuron population
synini	A C-type array of doubles that contains the initial values for synapse variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
p	A C-type array of doubles that contains synapse parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
PSVini	A C-type array of doubles that contains the initial values for postsynaptic mechanism variables (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.
ps	A C-type array of doubles that contains postsynaptic mechanism parameter values (common to all synapses of the population) which will be used for the defined synapses. The array must contain the right number of parameters in the right order for the chosen synapse type. If too few, segmentation faults will occur, if too many, excess will be ignored.

14.11.3.10 void NNmodel::checkSizes (unsigned int * , unsigned int * , unsigned int *)

14.11.3.11 void NNmodel::finalize ( )

14.11.3.12 void NNmodel::initLearnGrps ( )

14.11.3.13 void NNmodel::setConstInp (const string *sName,* double *globalInp0*)

Method for setting the global input value for a neuron population if CONSTINP.

This function sets a global input value to the specified neuron group.

14.11.3.14 void NNmodel::setGPUDevice (int *device*)

Method to choose the GPU to be used for the model. If "AUTODEVICE' (-1), GeNN will choose the device based on a heuristic rule.

This function defines the way how the GPU is chosen. If "AUTODEVICE" (-1) is given as the argument, GeNN will use internal heuristics to choose the device. Otherwise the argument is the device number and the indicated device will be used.

14.11.3.15 void NNmodel::setMaxConn (const string *sname,* unsigned int *maxConnP*)

This function defines the maximum number of connections for a neuron in the population.

14.11.3.16 void NNmodel::setName (const string *inname*)

Method to set the neuronal network model name.

14.11.3.17 void NNmodel::setNeuronClusterIndex (const string *neuronGroup,* int *hostID,* int *deviceID*)

Function for setting which host and which device a neuron group will be simulated on.

This function is for setting which host and which device a neuron group will be simulated on.

Parameters

neuronGroup	Name of the neuron population
hostID	ID of the host
deviceID	ID of the device

14.11.3.18 void NNmodel::setPrecision (unsigned int *floattype*)

Set numerical precision for floating point.

This function sets the numerical precision of floating type variables. By default, it is GENN_GENN_FLOAT.

14.11.3.19 void NNmodel::setSeed (unsigned int *inseed*)

Set the random seed (disables automatic seeding if argument not 0).

This function sets the random seed. If the passed argument is $>$ 0, automatic seeding is disabled. If the argument is 0, the underlying seed is obtained from the time() function.

Parameters

inseed	the new seed

14.11.3.20 void NNmodel::setSynapseClusterIndex (const string *synapseGroup*, int *hostID*, int *deviceID*)

Function for setting which host and which device a synapse group will be simulated on.

This function is for setting which host and which device a synapse group will be simulated on.

Parameters

synapseGroup	Name of the synapse population
hostID	ID of the host
deviceID	ID of the device

14.11.3.21 void NNmodel::setSynapseG (const string *sName*, double *g*)

Method for setting the conductance (g) value for a synapse population with "GLOBALG" charactertistic.

This functions sets the global value of the maximal synaptic conductance for a synapse population that was idfentified as conductance specifcation method "GLOBALG".

14.11.3.22 void NNmodel::setTiming (bool *theTiming*)

Set whether timers and timing commands are to be included.

This function sets a flag to determine whether timers and timing commands are to be included in generated code.

14.11.4 Member Data Documentation

14.11.4.1 int NNmodel::chooseGPUDevice

14.11.4.2 vector$<$vector$<$double$>$ $>$ NNmodel::dnp

Derived neuron parameters.

14.11.4.3 vector$<$vector$<$double$>$ $>$ NNmodel::dpsp

Derived postsynapse parameters.

14.11.4.4 vector$<$vector$<$double$>$ $>$ NNmodel::dsp_w

Derived synapse parameters (weightUpdateModel only)

14.11.4.5 int NNmodel::final

Flag for whether the model has been finalized.

14.11.4.6 string NNmodel::ftype

Type of floating point variables (float, double, ...; default: float)

14.11.4.7 vector<double> NNmodel::globalInp

Global explicit input if CONSTINP is chosen.

14.11.4.8 vector<vector<unsigned int> > NNmodel::inSyn

The ids of the incoming synapse groups.

14.11.4.9 unsigned int NNmodel::lrnGroups

Number of synapse groups with learning.

14.11.4.10 vector<unsigned int> NNmodel::lrnSynGrp

Enumeration of the IDs of synapse groups that learn.

14.11.4.11 vector<unsigned int> NNmodel::maxConn

Padded summed maximum number of connections for a neuron in the neuron groups.

14.11.4.12 string NNmodel::name

Name of the neuronal newtwork model.

14.11.4.13 unsigned int NNmodel::needSt

Whether last spike times are needed at all in this network model (related to STDP)

14.11.4.14 unsigned int NNmodel::needSynapseDelay

Whether delayed synapse conductance is required in the network.

14.11.4.15 vector<unsigned int> NNmodel::neuronDelaySlots

The number of slots needed in the synapse delay queues of a neuron group.

14.11.4.16 vector<int> NNmodel::neuronDeviceID

The ID of the CUDA device which the neuron groups are comnputed on.

14.11.4.17 unsigned int NNmodel::neuronGrpN

Number of neuron groups.

14.11.4.18 vector<int> NNmodel::neuronHostID

The ID of the cluster node which the neuron groups are computed on.

14.11.4.19 vector<vector<double> > NNmodel::neuronIni

Initial values of neurons.

14.11.4.20 vector<unsigned int> NNmodel::neuronN

Number of neurons in group.

14.11.4.21 vector<string> NNmodel::neuronName

Names of neuron groups.

14.11.4.22 vector<bool> NNmodel::neuronNeedSpkEvnt

Whether spike-like events from a group are required.

14.11.4.23 vector<bool> NNmodel::neuronNeedSt

Whether last spike time needs to be saved for a group.

14.11.4.24 vector<bool> NNmodel::neuronNeedTrueSpk

Whether spike-like events from a group are required.

14.11.4.25 vector<vector<double> > NNmodel::neuronPara

Parameters of neurons.

14.11.4.26 vector<unsigned int> NNmodel::neuronPostSyn

14.11.4.27 vector<string> NNmodel::neuronSpkEvntCondition

Will contain the spike event condition code when spike events are used.

14.11.4.28 vector<unsigned int> NNmodel::neuronType

Postsynaptic methods to the neuron.
Types of neurons

14.11.4.29 vector<vector<bool> > NNmodel::neuronVarNeedQueue

Whether a neuron variable needs queueing for syn code.

14.11.4.30 vector<vector<unsigned int> > NNmodel::outSyn

The ids of the outgoing synapse groups.

14.11.4.31 vector<unsigned int> NNmodel::padSumLearnN

Padded summed neuron numbers of learn group source populations.

14.11.4.32 vector<unsigned int> NNmodel::padSumNeuronN

Padded summed neuron numbers.

14.11.4.33 vector<unsigned int> NNmodel::padSumSynapseKrnl

14.11.4.34 vector<vector<double> > NNmodel::postSynapsePara

parameters of postsynapses

14.11.4.35 vector<unsigned int> NNmodel::postSynapseType

Types of post-synaptic model.

14.11.4.36 vector<vector<double> > NNmodel::postSynIni

Initial values of postsynaptic variables.

14.11.4.37 vector<unsigned int> NNmodel::receivesInputCurrent

flags whether neurons of a population receive explicit input currents

14.11.4.38 string NNmodel::RNtype

Underlying type for random number generation (default: long)

14.11.4.39 unsigned int NNmodel::seed

14.11.4.40 vector<unsigned int> NNmodel::sumNeuronN

Summed neuron numbers.

14.11.4.41 vector<unsigned int> NNmodel::synapseConnType

Connectivity type of synapses.

14.11.4.42 vector<unsigned int> NNmodel::synapseDelay

Global synaptic conductance delay for the group (in time steps)

14.11.4.43 vector<int> NNmodel::synapseDeviceID

The ID of the CUDA device which the synapse groups are comnputed on.

14.11.4.44 unsigned int NNmodel::synapseGrpN

Number of synapse groups.

14.11.4.45 vector<unsigned int> NNmodel::synapseGType

Type of specification method for synaptic conductance.

14.11.4.46 vector<int> NNmodel::synapseHostID

The ID of the cluster node which the synapse groups are computed on.

14.11.4.47 vector<vector<double> > NNmodel::synapseIni

Initial values of synapse variables.

14.11.4.48 vector<unsigned int> NNmodel::synapseInSynNo

IDs of the target neurons' incoming synapse variables for each synapse group.

14.11.4.49 vector<string> NNmodel::synapseName

Names of synapse groups.

14.11.4.50 vector<unsigned int> NNmodel::synapseOutSynNo

The target neurons' outgoing synapse for each synapse group.

14.11.4.51 vector<vector<double> > NNmodel::synapsePara

parameters of synapses

14.11.4.52 vector<unsigned int> NNmodel::synapseSource

Presynaptic neuron groups.

14.11.4.53 vector<vector<string> > NNmodel::synapseSpkEvntVars

Defines variable names that are needed in the SpkEvnt condition and that are pre-fetched for that purpose into shared memory.

14.11.4.54 vector<unsigned int> NNmodel::synapseTarget

Postsynaptic neuron groups.

14.11.4.55 vector<unsigned int> NNmodel::synapseType

Types of synapses.

14.11.4.56 vector<bool> NNmodel::synapseUsesPostLearning

Defines if anything is done in case of postsynaptic neuron spiking before presynaptic neuron (punishment in STDP etc.)

14.11.4.57 vector<bool> NNmodel::synapseUsesSpikeEvents

Defines if synapse update is done after detection of spike events (every point above threshold)

14.11.4.58 vector<bool> NNmodel::synapseUsesTrueSpikes

Defines if synapse update is done after detection of real spikes (only one point after threshold)

14.11.4.59 bool NNmodel::timing

The documentation for this class was generated from the following files:

- modelSpec.h
- modelSpec.cc

14.12 Parameter Struct Reference

Public Attributes

- string name
- string value

14.12.1 Member Data Documentation

14.12.1.1 string Parameter::name

14.12.1.2 string Parameter::value

The documentation for this struct was generated from the following file:

- experiment.cu

14.13 postSynModel Struct Reference

Structure to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

`#include <modelSpec.h>`

Public Attributes

- string postSyntoCurrent

  Code that defines how postsynaptic update is translated to current.
- string postSynDecay

  Code that defines how postsynaptic current decays.
- vector< string > varNames

  Names of the variables in the postsynaptic model.
- vector< string > varTypes

  Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.
- vector< string > pNames

  Names of (independent) parameters of the model.
- vector< string > dpNames

  Names of dependent parameters of the model.
- dpclass * dps

  Derived parameters.

14.13.1 Detailed Description

Structure to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

14.13.2 Member Data Documentation

14.13.2.1 vector<string> postSynModel::dpNames

Names of dependent parameters of the model.

14.13.2.2 dpclass* postSynModel::dps

Derived parameters.

14.13.2.3 vector<string> postSynModel::pNames

Names of (independent) parameters of the model.

14.13.2.4 string postSynModel::postSynDecay

Code that defines how postsynaptic current decays.

14.13.2.5 string postSynModel::postSyntoCurrent

Code that defines how postsynaptic update is translated to current.

14.13.2.6 vector<string> postSynModel::varNames

Names of the variables in the postsynaptic model.

14.13.2.7 vector<string> postSynModel::varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this struct was generated from the following file:

- modelSpec.h

14.14 postVarsInPostLearn Class Reference

`#include <testPostVarsInPostLearn.h>`

Public Member Functions

- postVarsInPostLearn ()
- ~postVarsInPostLearn ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.14.1 Constructor & Destructor Documentation

14.14.1.1 postVarsInPostLearn::postVarsInPostLearn ( )

14.14.1.2 postVarsInPostLearn::~postVarsInPostLearn ( )

14.14.2 Member Function Documentation

14.14.2.1 void postVarsInPostLearn::init_neurons ( )

14.14.2.2 void postVarsInPostLearn::init_synapses ( )

14.14.2.3 void postVarsInPostLearn::run (float *t*, int *which*)

14.14.3 Member Data Documentation

14.14.3.1 float** postVarsInPostLearn::theW

The documentation for this class was generated from the following files:

- testPostVarsInPostLearn.h
- testPostVarsInPostLearn.cu

14.15 postVarsInPostLearn_sparse Class Reference

`#include <testPostVarsInPostLearn_sparse.h>`

Public Member Functions

- postVarsInPostLearn_sparse ()
- ~postVarsInPostLearn_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.15.1 Constructor & Destructor Documentation

14.15.1.1 postVarsInPostLearn_sparse::postVarsInPostLearn_sparse ( )

14.15.1.2 postVarsInPostLearn_sparse::~postVarsInPostLearn_sparse ( )

14.15.2 Member Function Documentation

14.15.2.1 void postVarsInPostLearn_sparse::init_neurons ( )

14.15.2.2 void postVarsInPostLearn_sparse::init_synapses ( )

14.15.2.3 void postVarsInPostLearn_sparse::run (float *t,* int *which*)

14.15.3 Member Data Documentation

14.15.3.1 float** postVarsInPostLearn_sparse::theW

The documentation for this class was generated from the following files:

- testPostVarsInPostLearn_sparse.h
- testPostVarsInPostLearn_sparse.cu

14.16 postVarsInSimCode Class Reference

`#include <testPostVarsInSimCode.h>`

Public Member Functions

- postVarsInSimCode ()
- ~postVarsInSimCode ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.16.1 Constructor & Destructor Documentation

14.16.1.1 postVarsInSimCode::postVarsInSimCode ( )

14.16.1.2 postVarsInSimCode::~postVarsInSimCode ( )

14.16.2 Member Function Documentation

14.16.2.1 void postVarsInSimCode::init_neurons ( )

14.16.2.2 void postVarsInSimCode::init_synapses ( )

14.16.2.3 void postVarsInSimCode::run (float *t,* int *which*)

14.16.3 Member Data Documentation

14.16.3.1 float** postVarsInSimCode::theW

The documentation for this class was generated from the following files:

- testPostVarsInSimCode.h
- testPostVarsInSimCode.cu

14.17 postVarsInSimCode_sparse Class Reference

`#include <testPostVarsInSimCode_sparse.h>`

Public Member Functions

- postVarsInSimCode_sparse ()
- ~postVarsInSimCode_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.17.1 Constructor & Destructor Documentation

14.17.1.1 postVarsInSimCode_sparse::postVarsInSimCode_sparse ( )

14.17.1.2 postVarsInSimCode_sparse::~postVarsInSimCode_sparse ( )

14.17.2 Member Function Documentation

14.17.2.1 void postVarsInSimCode_sparse::init_neurons ( )

14.17.2.2 void postVarsInSimCode_sparse::init_synapses ( )

14.17.2.3 void postVarsInSimCode_sparse::run (float *t*, int *which*)

14.17.3 Member Data Documentation

14.17.3.1 float** postVarsInSimCode_sparse::theW

The documentation for this class was generated from the following files:

- testPostVarsInSimCode_sparse.h
- testPostVarsInSimCode_sparse.cu

14.18 postVarsInSynapseDynamics Class Reference

`#include <testPostVarsInSynapseDynamics.h>`

Public Member Functions

- postVarsInSynapseDynamics ()
- ~postVarsInSynapseDynamics ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.18.1 Constructor & Destructor Documentation

14.18.1.1 postVarsInSynapseDynamics::postVarsInSynapseDynamics ( )

14.18.1.2 postVarsInSynapseDynamics::~postVarsInSynapseDynamics ( )

14.18.2 Member Function Documentation

14.18.2.1 void postVarsInSynapseDynamics::init_neurons ( )

14.18.2.2 void postVarsInSynapseDynamics::init_synapses ( )

14.18.2.3 void postVarsInSynapseDynamics::run (float *t,* int *which*)

14.18.3 Member Data Documentation

14.18.3.1 float** postVarsInSynapseDynamics::theW

The documentation for this class was generated from the following files:

- testPostVarsInSynapseDynamics.h
- testPostVarsInSynapseDynamics.cu

14.19 postVarsInSynapseDynamics_sparse Class Reference

`#include <testPostVarsInSynapseDynamics_sparse.h>`

Public Member Functions

- postVarsInSynapseDynamics_sparse ()
- ~postVarsInSynapseDynamics_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.19.1 Constructor & Destructor Documentation

14.19.1.1 postVarsInSynapseDynamics_sparse::postVarsInSynapseDynamics_sparse ( )

14.19.1.2 postVarsInSynapseDynamics_sparse::~postVarsInSynapseDynamics_sparse ( )

14.19.2 Member Function Documentation

14.19.2.1 void postVarsInSynapseDynamics_sparse::init_neurons ( )

14.19.2.2 void postVarsInSynapseDynamics_sparse::init_synapses ( )

14.19.2.3 void postVarsInSynapseDynamics_sparse::run (float *t*, int *which*)

14.19.3 Member Data Documentation

14.19.3.1 float** postVarsInSynapseDynamics_sparse::theW

The documentation for this class was generated from the following files:

- testPostVarsInSynapseDynamics_sparse.h
- testPostVarsInSynapseDynamics_sparse.cu

14.20 preVarsInPostLearn Class Reference

`#include <testPreVarsInPostLearn.h>`

Public Member Functions

- preVarsInPostLearn ()
- ~preVarsInPostLearn ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.20.1 Constructor & Destructor Documentation

14.20.1.1 preVarsInPostLearn::preVarsInPostLearn ( )

14.20.1.2 preVarsInPostLearn::~preVarsInPostLearn ( )

14.20.2 Member Function Documentation

14.20.2.1 void preVarsInPostLearn::init_neurons ( )

14.20.2.2 void preVarsInPostLearn::init_synapses ( )

14.20.2.3 void preVarsInPostLearn::run (float *t*, int *which*)

14.20.3 Member Data Documentation

14.20.3.1 float** preVarsInPostLearn::theW

The documentation for this class was generated from the following files:

- testPreVarsInPostLearn.h
- testPreVarsInPostLearn.cu

14.21 preVarsInPostLearn_sparse Class Reference

`#include <testPreVarsInPostLearn_sparse.h>`

Public Member Functions

- preVarsInPostLearn_sparse ()
- ~preVarsInPostLearn_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.21.1 Constructor & Destructor Documentation

14.21.1.1 preVarsInPostLearn_sparse::preVarsInPostLearn_sparse ( )

14.21.1.2 preVarsInPostLearn_sparse::~preVarsInPostLearn_sparse ( )

14.21.2 Member Function Documentation

14.21.2.1 void preVarsInPostLearn_sparse::init_neurons ( )

14.21.2.2 void preVarsInPostLearn_sparse::init_synapses ( )

14.21.2.3 void preVarsInPostLearn_sparse::run (float *t*, int *which*)

14.21.3 Member Data Documentation

14.21.3.1 float** preVarsInPostLearn_sparse::theW

The documentation for this class was generated from the following files:

- testPreVarsInPostLearn_sparse.h
- testPreVarsInPostLearn_sparse.cu

14.22 preVarsInSimCode Class Reference

`#include <testPreVarsInSimCode.h>`

Public Member Functions

- preVarsInSimCode ()
- ~preVarsInSimCode ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.22.1 Constructor & Destructor Documentation

14.22.1.1 preVarsInSimCode::preVarsInSimCode ( )

14.22.1.2 preVarsInSimCode::~preVarsInSimCode ( )

14.22.2 Member Function Documentation

14.22.2.1 void preVarsInSimCode::init_neurons ( )

14.22.2.2 void preVarsInSimCode::init_synapses ( )

14.22.2.3 void preVarsInSimCode::run (float *t*, int *which*)

14.22.3 Member Data Documentation

14.22.3.1 float** preVarsInSimCode::theW

The documentation for this class was generated from the following files:

- testPreVarsInSimCode.h
- testPreVarsInSimCode.cu

14.23 preVarsInSimCode_sparse Class Reference

`#include <testPreVarsInSimCode_sparse.h>`

Public Member Functions

- preVarsInSimCode_sparse ()
- ~preVarsInSimCode_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.23.1 Constructor & Destructor Documentation

14.23.1.1 preVarsInSimCode_sparse::preVarsInSimCode_sparse ( )

14.23.1.2 preVarsInSimCode_sparse::∼preVarsInSimCode_sparse ( )

14.23.2 Member Function Documentation

14.23.2.1 void preVarsInSimCode_sparse::init_neurons ( )

14.23.2.2 void preVarsInSimCode_sparse::init_synapses ( )

14.23.2.3 void preVarsInSimCode_sparse::run (float *t,* int *which*)

14.23.3 Member Data Documentation

14.23.3.1 float∗∗ preVarsInSimCode_sparse::theW

The documentation for this class was generated from the following files:

- testPreVarsInSimCode_sparse.h
- testPreVarsInSimCode_sparse.cu

14.24 preVarsInSynapseDynamics Class Reference

`#include <testPreVarsInSynapseDynamics.h>`

Public Member Functions

- preVarsInSynapseDynamics ()
- ∼preVarsInSynapseDynamics ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ∗∗ theW

14.24.1 Constructor & Destructor Documentation

14.24.1.1 preVarsInSynapseDynamics::preVarsInSynapseDynamics ( )

14.24.1.2 preVarsInSynapseDynamics::∼preVarsInSynapseDynamics ( )

14.24.2 Member Function Documentation

14.24.2.1 void preVarsInSynapseDynamics::init_neurons ( )

14.24.2.2 void preVarsInSynapseDynamics::init_synapses ( )

14.24.2.3 void preVarsInSynapseDynamics::run (float *t,* int *which*)

14.24.3 Member Data Documentation

14.24.3.1 float** preVarsInSynapseDynamics::theW

The documentation for this class was generated from the following files:

- testPreVarsInSynapseDynamics.h
- testPreVarsInSynapseDynamics.cu

14.25 preVarsInSynapseDynamics_sparse Class Reference

`#include <testPreVarsInSynapseDynamics_sparse.h>`

Public Member Functions

- preVarsInSynapseDynamics_sparse ()
- ~preVarsInSynapseDynamics_sparse ()
- void init_synapses ()
- void init_neurons ()
- void run (float, int)

Public Attributes

- float ** theW

14.25.1 Constructor & Destructor Documentation

14.25.1.1 preVarsInSynapseDynamics_sparse::preVarsInSynapseDynamics_sparse ( )

14.25.1.2 preVarsInSynapseDynamics_sparse::~preVarsInSynapseDynamics_sparse ( )

14.25.2 Member Function Documentation

14.25.2.1 void preVarsInSynapseDynamics_sparse::init_neurons ( )

14.25.2.2 void preVarsInSynapseDynamics_sparse::init_synapses ( )

14.25.2.3 void preVarsInSynapseDynamics_sparse::run (float *t*, int *which*)

14.25.3 Member Data Documentation

14.25.3.1 float** preVarsInSynapseDynamics_sparse::theW

The documentation for this class was generated from the following files:

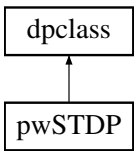
- testPreVarsInSynapseDynamics_sparse.h
- testPreVarsInSynapseDynamics_sparse.cu

14.26 pwSTDP Class Reference

TODO This class definition may be code-generated in a future release.

`#include <utils.h>`

Inheritance diagram for pwSTDP:



Public Member Functions

- double calculateDerivedParameter (int index, vector< double > pars, double dt)
- double lim0 (vector< double > pars, double dt)
- double lim1 (vector< double > pars, double dt)
- double slope0 (vector< double > pars, double dt)
- double slope1 (vector< double > pars, double dt)
- double off0 (vector< double > pars, double dt)
- double off1 (vector< double > pars, double dt)
- double off2 (vector< double > pars, double dt)

14.26.1 Detailed Description

TODO This class definition may be code-generated in a future release.

This class defines derived parameters for the learn1synapse standard weightupdate model

14.26.2 Member Function Documentation

14.26.2.1 double pwSTDP::calculateDerivedParameter (int *index*, vector< double > *pars*, double *dt*) [inline], [virtual]

Reimplemented from dpclass.

14.26.2.2 double pwSTDP::lim0 (vector< double > *pars*, double *dt*) [inline]

14.26.2.3 double pwSTDP::lim1 (vector< double > *pars*, double *dt*) [inline]

14.26.2.4 double pwSTDP::off0 (vector< double > *pars*, double *dt*) [inline]

14.26.2.5 double pwSTDP::off1 (vector< double > *pars*, double *dt*) [inline]

14.26.2.6 double pwSTDP::off2 (vector< double > *pars*, double *dt*) [inline]

14.26.2.7 double pwSTDP::slope0 (vector< double > *pars*, double *dt*) [inline]

14.26.2.8 double pwSTDP::slope1 (vector< double > *pars*, double *dt*) [inline]

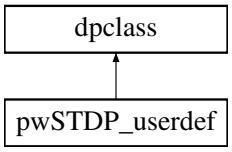
The documentation for this class was generated from the following file:

- utils.h

14.27 pwSTDP_userdef Class Reference

TODO This class definition may be code-generated in a future release.

Inheritance diagram for pwSTDP_userdef:



Public Member Functions

- double calculateDerivedParameter (int index, vector< double > pars, double dt=DT)
- double lim0 (vector< double > pars, double dt)
- double lim1 (vector< double > pars, double dt)
- double slope0 (vector< double > pars, double dt)
- double slope1 (vector< double > pars, double dt)
- double off0 (vector< double > pars, double dt)
- double off1 (vector< double > pars, double dt)
- double off2 (vector< double > pars, double dt)

14.27.1 Detailed Description

TODO This class definition may be code-generated in a future release.

14.27.2 Member Function Documentation

14.27.2.1 double pwSTDP_userdef::calculateDerivedParameter (int *index*, vector< double > *pars*, double *dt* = DT) [inline], [virtual]

Reimplemented from dpclass.

14.27.2.2 double pwSTDP_userdef::lim0 (vector< double > *pars*, double *dt*) [inline]

14.27.2.3 double pwSTDP_userdef::lim1 (vector< double > *pars*, double *dt*) [inline]

14.27.2.4 double pwSTDP_userdef::off0 (vector< double > *pars*, double *dt*) [inline]

14.27.2.5 double pwSTDP_userdef::off1 (vector< double > *pars*, double *dt*) [inline]

14.27.2.6 double pwSTDP_userdef::off2 (vector< double > *pars*, double *dt*) [inline]

14.27.2.7 double pwSTDP_userdef::slope0 (vector< double > *pars*, double *dt*) [inline]

14.27.2.8 double pwSTDP_userdef::slope1 (vector< double > *pars*, double *dt*) [inline]

The documentation for this class was generated from the following file:

- MBody_userdef.cc

14.28 randomGauss Class Reference

Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

`#include <gauss.h>`

Public Member Functions

- randomGauss ()

Constructor for the Gaussian random number generator class without giving explicit seeds.

- randomGauss (unsigned long, unsigned long, unsigned long)

  Constructor for the Gaussian random number generator class when seeds are provided explicitly.

- ~randomGauss ()
- double n ()

  Method for obtaining a random number with Gaussian distribution.

- void srand (unsigned long, unsigned long, unsigned long)

  Function for seeding with fixed seeds.

14.28.1 Detailed Description

Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

A random number from a Gaussian distribution of mean 0 and standard deviation 1 is obtained by calling the method randomGauss::n().

14.28.2 Constructor & Destructor Documentation

14.28.2.1 randomGauss::randomGauss ( ) [explicit]

Constructor for the Gaussian random number generator class without giving explicit seeds.

The seeds for random number generation are generated from the internal clock of the computer during execution.

14.28.2.2 randomGauss::randomGauss (unsigned long *seed1*, unsigned long *seed2*, unsigned long *seed3*)

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

The seeds are three arbitrary unsigned long integers.

14.28.2.3 randomGauss::~randomGauss ( ) [inline]

14.28.3 Member Function Documentation

14.28.3.1 double randomGauss::n ( )

Method for obtaining a random number with Gaussian distribution.

Function for generating a pseudo random number from a Gaussian distribution.

14.28.3.2 void randomGauss::srand (unsigned long *seed1*, unsigned long *seed2*, unsigned long *seed3*)

Function for seeding with fixed seeds.

The documentation for this class was generated from the following files:

- gauss.h
- gauss.cc

14.29 randomGen Class Reference

Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.

```
#include <randomGen.h>
```

Public Member Functions

- randomGen ()

  Constructor for the ISAAC random number generator class without giving explicit seeds.
- randomGen (unsigned long, unsigned long, unsigned long)

  Constructor for the Gaussian random number generator class when seeds are provided explicitly.
- ~randomGen ()
- double n ()

  Method to obtain a random number from a uniform ditribution on [0,1].
- void srand (unsigned long, unsigned long, unsigned long)

  Function for seeding with fixed seeds.

14.29.1 Detailed Description

Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.

The random number generator initializes with system timea or explicit seeds and returns a random number according to a uniform distribution on [0,1]; making use of the ISAAC random number generator; C++ Implementation by Quinn Tyler Jackson of the RG invented by Bob Jenkins Jr.

14.29.2 Constructor & Destructor Documentation

14.29.2.1 randomGen::randomGen ( ) [explicit]

Constructor for the ISAAC random number generator class without giving explicit seeds.

The seeds for random number generation are generated from the internal clock of the computer during execution.

14.29.2.2 randomGen::randomGen (unsigned long *seed1*, unsigned long *seed2*, unsigned long *seed3*)

Constructor for the Gaussian random number generator class when seeds are provided explicitly.

The seeds are three arbitrary unsigned long integers.

14.29.2.3 randomGen::~randomGen ( ) [inline]

14.29.3 Member Function Documentation

14.29.3.1 double randomGen::n ( )

Method to obtain a random number from a uniform ditribution on [0,1].

Function for generating a pseudo random number from a uniform distribution on the interval [0,1].

14.29.3.2 void randomGen::srand (unsigned long *seed1*, unsigned long *seed2*, unsigned long *seed3*)

Function for seeding with fixed seeds.

The documentation for this class was generated from the following files:

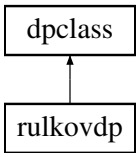
- randomGen.h
- randomGen.cc

14.30 rulkovdp Class Reference

Class defining the dependent parameters of teh Rulkov map neuron.

`#include <utils.h>`

Inheritance diagram for rulkovdp:



Public Member Functions

- double calculateDerivedParameter (int index, vector< double > pars, double dt=1.0)
- double ip0 (vector< double > pars)
- double ip1 (vector< double > pars)
- double ip2 (vector< double > pars)

14.30.1 Detailed Description

Class defining the dependent parameters of teh Rulkov map neuron.

14.30.2 Member Function Documentation

14.30.2.1 double rulkovdp::calculateDerivedParameter (int *index*, vector< double > *pars*, double *dt* = `1.0`) [inline], [virtual]

Reimplemented from dpclass.

14.30.2.2 double rulkovdp::ip0 (vector< double > *pars*) [inline]

14.30.2.3 double rulkovdp::ip1 (vector< double > *pars*) [inline]

14.30.2.4 double rulkovdp::ip2 (vector< double > *pars*) [inline]

The documentation for this class was generated from the following file:

- utils.h

14.31 Schmuker2014_classifier Class Reference

This class cpontains the methods for running the Schmuker_2014_classifier example model.

`#include <Schmuker2014_classifier.h>`

Public Types

- enum data_type { data_type_int, data_type_uint, data_type_float, data_type_double }

Public Member Functions

- Schmuker2014_classifier ()
- ~Schmuker2014_classifier ()
- void allocateHostAndDeviceMemory ()
- void populateDeviceMemory ()
- void update_input_data_on_device ()
- void clearDownDevice ()

- void run (float runtime, string filename_rasterPlot, bool usePlasticity)
- void getSpikesFromGPU ()
- void getSpikeNumbersFromGPU ()
- void outputSpikes (FILE *, string delim)
- void initialiseWeights_SPARSE_RN_PN ()
- void initialiseWeights_WTA_PN_PN ()
- void initialiseWeights_DENSE_PN_AN ()
- void initialiseWeights_WTA_AN_AN ()
- void createWTAConnectivity (float *synapse, UINT populationSize, UINT clusterSize, float synapseWeight, float probability)
- bool randomEventOccurred (float probability)
- void updateWeights_PN_AN_on_device ()
- void generate_or_load_inputrates_dataset (unsigned int recordingIdx)
- void generate_inputrates_dataset (unsigned int recordingIdx)
- FILE * openRecordingFile (UINT recordingIndex)
- void applyLearningRuleSynapses (float *synapsesPNAN)
- void initialiseInputData ()
- void load_VR_data ()
- void setCorrectClass (UINT recordingIdx)
- UINT getClassCluster (UINT anIdx)
- void loadClassLabels ()
- void addInputRate (float *samplePoint, UINT timeStep)
- uint64_t convertToRateCode (float inputRateHz)
- float calculateVrResponse (float *samplePoint, float *vrPoint)
- void setMaxMinSampleDistances ()
- void findMaxMinSampleDistances (float *samples, UINT startAt, UINT totalSamples)
- float getSampleDistance (UINT max0_min1)
- float getManhattanDistance (float *pointA, float *pointB, UINT numElements)
- float getRand0to1 ()
- UINT calculateOverallWinner ()
- UINT calculateWinner (unsigned int *clusterSpikeCount)
- UINT calculateCurrentWindowWinner ()
- void updateIndividualSpikeCountPN ()
- void resetIndividualSpikeCountPN ()
- void updateClusterSpikeCountAN ()
- void resetClusterSpikeCountAN ()
- void resetOverallWinner ()
- void updateWeights_PN_AN ()
- UINT getClusterIndex (UINT neuronIndex, UINT clusterSize)
- void generateSimulatedTimeSeriesData ()
- string getRecordingFilename (UINT recordingIdx)
- bool loadArrayFromTextFile (string path, void *array, string delim, UINT arrayLen, data_type dataType)
- void checkContents (string title, void *array, UINT howMany, UINT displayPerLine, data_type dataType, UINT decimalPoints)
- void checkContents (string title, void *array, UINT howMany, UINT displayPerLine, data_type dataType, UINT decimalPoints, string delim)
- void printSeparator ()
- void resetDevice ()
- void startLog ()

Public Attributes

- double d_maxRandomNumber
- float t
- NNmodel model
- uint64_t * inputRates
- unsigned int inputRatesSize
- float * vrData
- unsigned int * classLabel
- unsigned int * individualSpikeCountPN
- unsigned int * overallWinnerSpikeCountAN
- unsigned int * clusterSpikeCountAN
- float * plasticWeights
- uint64_t * d_inputRates
- unsigned int countRN
- unsigned int countPN
- unsigned int countAN
- unsigned int countPNAN
- float * sampleDistance
- string recordingsDir
- string cacheDir
- string outputDir
- string datasetName
- string uniqueRunId
- UINT correctClass
- int winningClass
- FILE * log
- UINT param_SPIKING_ACTIVITY_THRESHOLD_HZ
- UINT param_MAX_FIRING_RATE_HZ
- UINT param_MIN_FIRING_RATE_HZ
- float param_GLOBAL_WEIGHT_SCALING
- float param_WEIGHT_RN_PN
- float param_CONNECTIVITY_RN_PN
- float param_WEIGHT_WTA_PN_PN
- float param_WEIGHT_WTA_AN_AN
- float param_CONNECTIVITY_PN_PN
- float param_CONNECTIVITY_AN_AN
- float param_CONNECTIVITY_PN_AN
- float param_MIN_WEIGHT_PN_AN
- float param_MAX_WEIGHT_PN_AN
- float param_WEIGHT_DELTA_PN_AN
- float param_PLASTICITY_INTERVAL_MS
- bool clearedDownDevice

Static Public Attributes

- static const unsigned int timestepsPerRecording = RECORDING_TIME_MS / DT

14.31.1 Detailed Description

This class cpontains the methods for running the Schmuker_2014_classifier example model.

14.31.2 Member Enumeration Documentation

14.31.2.1 enum Schmuker2014_classifier::data_type

Enumerator

data_type_int

data_type_uint

data_type_float

data_type_double

14.31.3 Constructor & Destructor Documentation

14.31.3.1 Schmuker2014_classifier::Schmuker2014_classifier ( )

14.31.3.2 Schmuker2014_classifier::∼Schmuker2014_classifier ( )

14.31.4 Member Function Documentation

14.31.4.1 void Schmuker2014_classifier::addInputRate (float ∗ *samplePoint,* UINT *timeStep*)

14.31.4.2 void Schmuker2014_classifier::allocateHostAndDeviceMemory ( )

14.31.4.3 void Schmuker2014_classifier::applyLearningRuleSynapses (float ∗ *synapsesPNAN*)

14.31.4.4 UINT Schmuker2014_classifier::calculateCurrentWindowWinner ( )

14.31.4.5 UINT Schmuker2014_classifier::calculateOverallWinner ( )

14.31.4.6 float Schmuker2014_classifier::calculateVrResponse (float ∗ *samplePoint,* float ∗ *vrPoint*)

14.31.4.7 UINT Schmuker2014_classifier::calculateWinner (unsigned int ∗ *clusterSpikeCount*)

14.31.4.8 void Schmuker2014_classifier::checkContents (string *title,* void ∗ *array,* UINT *howMany,* UINT *displayPerLine,* data_type *dataType,* UINT *decimalPoints*)

14.31.4.9 void Schmuker2014_classifier::checkContents (string *title,* void ∗ *array,* UINT *howMany,* UINT *displayPerLine,* data_type *dataType,* UINT *decimalPoints,* string *delim*)

14.31.4.10 void Schmuker2014_classifier::clearDownDevice ( )

14.31.4.11 uint64_t Schmuker2014_classifier::convertToRateCode (float *inputRateHz*)

14.31.4.12 void Schmuker2014_classifier::createWTAConnectivity (float ∗ *synapse,* UINT *populationSize,* UINT *clusterSize,* float *synapseWeight,* float *probability*)

14.31.4.13 void Schmuker2014_classifier::findMaxMinSampleDistances (float ∗ *samples,* UINT *startAt,* UINT *totalSamples*)

14.31.4.14 void Schmuker2014_classifier::generate_inputrates_dataset (unsigned int *recordingIdx*)

14.31.4.15 void Schmuker2014_classifier::generate_or_load_inputrates_dataset (unsigned int *recordingIdx*)

14.31.4.16 void Schmuker2014_classifier::generateSimulatedTimeSeriesData ( )

14.31.4.17 UINT Schmuker2014_classifier::getClassCluster (UINT *anIdx*)

14.31.4.18 UINT Schmuker2014_classifier::getClusterIndex (UINT *neuronIndex,* UINT *clusterSize*)

14.31.4.19 float Schmuker2014_classifier::getManhattanDistance (float ∗ *pointA,* float ∗ *pointB,* UINT *numElements*)

14.31.4.20 float Schmuker2014_classifier::getRand0to1 ( )

14.31.4.21 string Schmuker2014_classifier::getRecordingFilename (UINT *recordingIdx*)

14.31.4.22 float Schmuker2014_classifier::getSampleDistance (UINT *max0_min1*)

14.31.4.23 void Schmuker2014_classifier::getSpikeNumbersFromGPU ( )

14.31.4.24 void Schmuker2014_classifier::getSpikesFromGPU ( )

14.31.4.25 void Schmuker2014_classifier::initialiseInputData ( )

14.31.4.26 void Schmuker2014_classifier::initialiseWeights_DENSE_PN_AN ( )

14.31.4.27 void Schmuker2014_classifier::initialiseWeights_SPARSE_RN_PN ( )

14.31.4.28 void Schmuker2014_classifier::initialiseWeights_WTA_AN_AN ( )

14.31.4.29 void Schmuker2014_classifier::initialiseWeights_WTA_PN_PN ( )

14.31.4.30 void Schmuker2014_classifier::load_VR_data ( )

14.31.4.31 bool Schmuker2014_classifier::loadArrayFromTextFile (string *path,* void * *array,* string *delim,* UINT *arrayLen,* data_type *dataType*)

14.31.4.32 void Schmuker2014_classifier::loadClassLabels ( )

14.31.4.33 FILE * Schmuker2014_classifier::openRecordingFile (UINT *recordingIndex*)

14.31.4.34 void Schmuker2014_classifier::outputSpikes (FILE * *f,* string *delim*)

14.31.4.35 void Schmuker2014_classifier::populateDeviceMemory ( )

14.31.4.36 void Schmuker2014_classifier::printSeparator ( )

14.31.4.37 bool Schmuker2014_classifier::randomEventOccurred (float *probability*)

14.31.4.38 void Schmuker2014_classifier::resetClusterSpikeCountAN ( )

14.31.4.39 void Schmuker2014_classifier::resetDevice ( )

14.31.4.40 void Schmuker2014_classifier::resetIndividualSpikeCountPN ( )

14.31.4.41 void Schmuker2014_classifier::resetOverallWinner ( )

14.31.4.42 void Schmuker2014_classifier::run (float *runtime,* string *filename_rasterPlot,* bool *usePlasticity*)

14.31.4.43 void Schmuker2014_classifier::setCorrectClass (UINT *recordingIdx*)

14.31.4.44 void Schmuker2014_classifier::setMaxMinSampleDistances ( )

14.31.4.45 void Schmuker2014_classifier::startLog ( )

14.31.4.46 void Schmuker2014_classifier::update_input_data_on_device ( )

14.31.4.47 void Schmuker2014_classifier::updateClusterSpikeCountAN ( )

14.31.4.48 void Schmuker2014_classifier::updateIndividualSpikeCountPN ( )

14.31.4.49 void Schmuker2014_classifier::updateWeights_PN_AN ( )

14.31.4.50 void Schmuker2014_classifier::updateWeights_PN_AN_on_device ( )

14.31.5 Member Data Documentation

14.31.5.1 string Schmuker2014_classifier::cacheDir

14.31.5.2 unsigned int* Schmuker2014_classifier::classLabel

14.31.5.3 bool Schmuker2014_classifier::clearedDownDevice

14.31.5.4 unsigned int* Schmuker2014_classifier::clusterSpikeCountAN

14.31.5.5 UINT Schmuker2014_classifier::correctClass

14.31.5.6 unsigned int Schmuker2014_classifier::countAN

14.31.5.7 unsigned int Schmuker2014_classifier::countPN

14.31.5.8 unsigned int Schmuker2014_classifier::countPNAN

14.31.5.9 unsigned int Schmuker2014_classifier::countRN

14.31.5.10 uint64_t* Schmuker2014_classifier::d_inputRates

14.31.5.11 double Schmuker2014_classifier::d_maxRandomNumber

14.31.5.12 string Schmuker2014_classifier::datasetName

14.31.5.13 unsigned int* Schmuker2014_classifier::individualSpikeCountPN

14.31.5.14 uint64_t* Schmuker2014_classifier::inputRates

14.31.5.15 unsigned int Schmuker2014_classifier::inputRatesSize

14.31.5.16 FILE* Schmuker2014_classifier::log

14.31.5.17 NNmodel Schmuker2014_classifier::model

14.31.5.18 string Schmuker2014_classifier::outputDir

14.31.5.19 unsigned int* Schmuker2014_classifier::overallWinnerSpikeCountAN

14.31.5.20 float Schmuker2014_classifier::param_CONNECTIVITY_AN_AN

14.31.5.21 float Schmuker2014_classifier::param_CONNECTIVITY_PN_AN

14.31.5.22 float Schmuker2014_classifier::param_CONNECTIVITY_PN_PN

14.31.5.23 float Schmuker2014_classifier::param_CONNECTIVITY_RN_PN

14.31.5.24 float Schmuker2014_classifier::param_GLOBAL_WEIGHT_SCALING

14.31.5.25 UINT Schmuker2014_classifier::param_MAX_FIRING_RATE_HZ

14.31.5.26 float Schmuker2014_classifier::param_MAX_WEIGHT_PN_AN

14.31.5.27 UINT Schmuker2014_classifier::param_MIN_FIRING_RATE_HZ

14.31.5.28 float Schmuker2014_classifier::param_MIN_WEIGHT_PN_AN

14.31.5.29 float Schmuker2014_classifier::param_PLASTICITY_INTERVAL_MS

14.31.5.30 UINT Schmuker2014_classifier::param_SPIKING_ACTIVITY_THRESHOLD_HZ

14.31.5.31 float Schmuker2014_classifier::param_WEIGHT_DELTA_PN_AN

14.31.5.32 float Schmuker2014_classifier::param_WEIGHT_RN_PN

14.31.5.33 float Schmuker2014_classifier::param_WEIGHT_WTA_AN_AN

14.31.5.34 float Schmuker2014_classifier::param_WEIGHT_WTA_PN_PN

14.31.5.35 float* Schmuker2014_classifier::plasticWeights

14.31.5.36 string Schmuker2014_classifier::recordingsDir

14.31.5.37 float* Schmuker2014_classifier::sampleDistance

14.31.5.38 float Schmuker2014_classifier::t

14.31.5.39 const unsigned int Schmuker2014_classifier::timestepsPerRecording = RECORDING_TIME_MS / DT [static]

14.31.5.40 string Schmuker2014_classifier::uniqueRunId

14.31.5.41 float* Schmuker2014_classifier::vrData

14.31.5.42 int Schmuker2014_classifier::winningClass

The documentation for this class was generated from the following files:

- Schmuker2014_classifier.h
- Schmuker2014_classifier.cu

14.32 stdRG Class Reference

`#include <randomGen.h>`

Public Member Functions

- stdRG ()
  Constructor of the standard random number generator class without explicit seed.
- stdRG (unsigned int)
  Constructor of the standard random number generator class with explicit seed.
- ~stdRG ()
- double n ()
  Method to generate a uniform random number.
- unsigned long nlong ()

14.32.1 Constructor & Destructor Documentation

14.32.1.1 stdRG::stdRG ( ) [explicit]

Constructor of the standard random number generator class without explicit seed.

The seed is taken from teh internal clock of the computer.

14.32.1.2 stdRG::stdRG (unsigned int *seed*)

Constructor of the standard random number generator class with explicit seed.

The seed is an arbitrary unsigned int

14.32.1.3 stdRG::~stdRG ( ) [inline]

14.32.2 Member Function Documentation

14.32.2.1 double stdRG::n ( )

Method to generate a uniform random number.

The moethod is a wrapper for the C function rand() and returns a pseudo random number in the interval [0,1[

14.32.2.2 unsigned long stdRG::nlong ( )

The documentation for this class was generated from the following files:

- randomGen.h
- randomGen.cc

14.33 stopWatch Struct Reference

`#include <hr_time.h>`

Public Attributes

- timeval start
- timeval stop

14.33.1 Member Data Documentation

14.33.1.1 timeval stopWatch::start

14.33.1.2 timeval stopWatch::stop

The documentation for this struct was generated from the following file:

- hr_time.h

14.34 SynDelay Class Reference

`#include <SynDelaySim.h>`

Public Member Functions

- SynDelay (bool usingGPU)
- ~SynDelay ()
- void run (float t)

14.34.1 Constructor & Destructor Documentation

14.34.1.1 SynDelay::SynDelay (bool *usingGPU*)

14.34.1.2 SynDelay::~SynDelay ( )

14.34.2 Member Function Documentation

14.34.2.1 void SynDelay::run (float *t*)

The documentation for this class was generated from the following files:

- SynDelaySim.h
- SynDelaySim.cu

14.35 weightUpdateModel Class Reference

Structure to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

`#include <modelSpec.h>`

Public Member Functions

- weightUpdateModel ()

Public Attributes

- string simCode

  Simulation code that is used for true spikes (only one time step after spike detection)
- string simCodeEvnt

  Simulation code that is used for spike events (all the instances where event threshold condition is met)
- string simLearnPost

  Simulation code which is used in the learnSynapsesPost kernel/function, where postsynaptic neuron spikes before the presynaptic neuron in the STDP window.
- string evntThreshold

  Simulation code for spike event detection.
- string synapseDynamics

  Simulation code for synapse dynamics independent of spike detection.
- vector< string > varNames

  Names of the variables in the postsynaptic model.
- vector< string > varTypes

  Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.
- vector< string > pNames

  Names of (independent) parameters of the model.
- vector< string > dpNames

  Names of dependent parameters of the model.
- vector< string > extraGlobalSynapseKernelParameters

  Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.
- vector< string > extraGlobalSynapseKernelParameterTypes

  Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.
- dpclass * dps
- bool needPreSt

  Whether presynaptic spike times are needed or not.
- bool needPostSt

  Whether postsynaptic spike times are needed or not.

14.35.1 Detailed Description

Structure to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

14.35.2 Constructor & Destructor Documentation

14.35.2.1 weightUpdateModel::weightUpdateModel ( ) [inline]

14.35.3 Member Data Documentation

14.35.3.1 vector<string> weightUpdateModel::dpNames

Names of dependent parameters of the model.

14.35.3.2 dpclass* weightUpdateModel::dps

14.35.3.3 string weightUpdateModel::evntThreshold

Simulation code for spike event detection.

14.35.3.4 vector<string> weightUpdateModel::extraGlobalSynapseKernelParameters

Additional parameter in the neuron kernel; it is translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

14.35.3.5 vector<string> weightUpdateModel::extraGlobalSynapseKernelParameterTypes

Additional parameters in the neuron kernel; they are translated to a population specific name but otherwise assumed to be one parameter per population rather than per synapse.

14.35.3.6 bool weightUpdateModel::needPostSt

Whether postsynaptic spike times are needed or not.

14.35.3.7 bool weightUpdateModel::needPreSt

Whether presynaptic spike times are needed or not.

14.35.3.8 vector<string> weightUpdateModel::pNames

Names of (independent) parameters of the model.

14.35.3.9 string weightUpdateModel::simCode

Simulation code that is used for true spikes (only one time step after spike detection)

14.35.3.10 string weightUpdateModel::simCodeEvnt

Simulation code that is used for spike events (all the instances where event threshold condition is met)

14.35.3.11 string weightUpdateModel::simLearnPost

Simulation code which is used in the learnSynapsesPost kernel/function, where postsynaptic neuron spikes before the presynaptic neuron in the STDP window.

14.35.3.12 string weightUpdateModel::synapseDynamics

Simulation code for synapse dynamics independent of spike detection.

14.35.3.13 vector<string> weightUpdateModel::varNames

Names of the variables in the postsynaptic model.

14.35.3.14 vector<string> weightUpdateModel::varTypes

Types of the variable named above, e.g. "float". Names and types are matched by their order of occurrence in the vector.

The documentation for this class was generated from the following file:

- modelSpec.h

15 File Documentation

15.1 00_MainPage.dox File Reference

15.2 01_Installation.dox File Reference

15.3 02_Quickstart.dox File Reference

15.4 03_Examples.dox File Reference

15.5 09_ReleaseNotes_v2.dox File Reference

15.6 10_UserManual.dox File Reference

15.7 11_Tutorial.dox File Reference

15.8 12_Tutorial.dox File Reference

15.9 13_UserGuide.dox File Reference

15.10 14_Credits.dox File Reference

15.11 classol_sim.cu File Reference

```
#include "classol_sim.h"
```

Functions

- int main (int argc, char *argv[])

  This function is the entry point for running the simulation of the MBody1 model network.

15.11.1 Function Documentation

15.11.1.1 int main (int *argc*, char * *argv[]*)

This function is the entry point for running the simulation of the MBody1 model network.

15.12 classol_sim.cu File Reference

```
#include "classol_sim.h"
```

Functions

- int main (int argc, char ∗argv[])

  This function is the entry point for running the simulation of the MBody_delayedSyn model network.

15.12.1 Function Documentation

15.12.1.1 int main (int *argc,* char ∗ *argv[ ]*)

This function is the entry point for running the simulation of the MBody_delayedSyn model network.

15.13 classol_sim.cu File Reference

```
#include "classol_sim.h"
```

Functions

- int main (int argc, char ∗argv[])

  This function is the entry point for running the simulation of the MBody1 model network.

15.13.1 Function Documentation

15.13.1.1 int main (int *argc,* char ∗ *argv[ ]*)

This function is the entry point for running the simulation of the MBody1 model network.

15.14 classol_sim.cu File Reference

```
#include "classol_sim.h"
```

Functions

- int main (int argc, char ∗argv[])

  This function is the entry point for running the simulation of the MBody1 model network.

15.14.1 Function Documentation

15.14.1.1 int main (int *argc,* char ∗ *argv[ ]*)

This function is the entry point for running the simulation of the MBody1 model network.

15.15 classol_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "MBody1.cc"
#include "map_classol.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);
- #define DBG_SIZE 10000
- #define PATTERNNO 100
- #define T_REPORT_TME 10000.0
- #define SYN_OUT_TME 20000.0
- #define PAT_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL_TME 500.0

Variables

- scalar t = 0.0f
- unsigned int iT = 0
- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- CStopWatch timer

15.15.1 Macro Definition Documentation

15.15.1.1 #define DBG_SIZE 10000

15.15.1.2 #define MYRAND(*Y*, *X*) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);

15.15.1.3 #define PAT_TIME 100.0

15.15.1.4 #define PATFTIME 1.5

15.15.1.5 #define PATTERNNO 100

15.15.1.6 #define SYN_OUT_TME 20000.0

15.15.1.7 #define T_REPORT_TME 10000.0

15.15.1.8 #define TOTAL_TME 500.0

15.15.2 Variable Documentation

15.15.2.1 scalar InputBaseRate = 2e-04

15.15.2.2 unsigned int iT = 0

15.15.2.3 int patFireTime

15.15.2.4 int patSetTime

15.15.2.5 scalar t = 0.0f

15.15.2.6 CStopWatch timer

15.16 classol_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "MBody_delayedSyn.cc"
#include "map_classol.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);
- #define DBG_SIZE 10000
- #define PATTERNNO 100
- #define T_REPORT_TME 10000.0
- #define SYN_OUT_TME 20000.0
- #define PAT_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL_TME 5000.0

Variables

- scalar t = 0.0f
- unsigned int iT = 0
- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- CStopWatch timer

15.16.1 Macro Definition Documentation

15.16.1.1 #define DBG_SIZE 10000

15.16.1.2 #define MYRAND(*Y*, *X*) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);

15.16.1.3 #define PAT_TIME 100.0

15.16.1.4 #define PATFTIME 1.5

15.16.1.5 #define PATTERNNO 100

15.16.1.6 #define SYN_OUT_TME 20000.0

15.16.1.7 #define T_REPORT_TME 10000.0

15.16.1.8 #define TOTAL_TME 5000.0

15.16.2 Variable Documentation

15.16.2.1 scalar InputBaseRate = 2e-04

15.16.2.2 unsigned int iT = 0

15.16.2.3 int patFireTime

15.16.2.4 int patSetTime

15.16.2.5 scalar t = 0.0f

15.16.2.6 CStopWatch timer

15.17 classol_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "MBody_individualID.cc"
#include "map_classol.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);
- #define DBG_SIZE 10000
- #define PATTERNNO 100
- #define T_REPORT_TME 10000.0
- #define SYN_OUT_TME 20000.0
- #define PAT_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL_TME 5000.0

Variables

- scalar t = 0.0f
- unsigned int iT = 0
- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- CStopWatch timer

15.17.1 Macro Definition Documentation

15.17.1.1 #define DBG_SIZE 10000

15.17.1.2 #define MYRAND(*Y*, *X*) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);

15.17.1.3 #define PAT_TIME 100.0

15.17.1.4 #define PATFTIME 1.5

15.17.1.5 #define PATTERNNO 100

15.17.1.6 #define SYN_OUT_TME 20000.0

15.17.1.7 #define T_REPORT_TME 10000.0

15.17.1.8 #define TOTAL_TME 5000.0

15.17.2 Variable Documentation

15.17.2.1 scalar InputBaseRate = 2e-04

15.17.2.2 unsigned int iT = 0

15.17.2.3 int patFireTime

15.17.2.4 int patSetTime

15.17.2.5 scalar t = 0.0f

15.17.2.6 CStopWatch timer

15.18 classol_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include <cfloat>
#include "MBody_userdef.cc"
#include "map_classol.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);
- #define DBG_SIZE 10000
- #define PATTERNNO 100
- #define T_REPORT_TME 10000.0
- #define SYN_OUT_TME 20000.0
- #define PAT_TIME 100.0
- #define PATFTIME 1.5
- #define TOTAL_TME 5000.0

Variables

- scalar t = 0.0f
- unsigned int iT = 0
- scalar InputBaseRate = 2e-04
- int patSetTime
- int patFireTime
- CStopWatch timer

15.18.1 Macro Definition Documentation

15.18.1.1 #define DBG_SIZE 10000

15.18.1.2 #define MYRAND(*Y*, *X*) Y = Y ∗ 1103515245 +12345; X= (Y >> 16);

15.18.1.3 #define PAT_TIME 100.0

15.18.1.4 #define PATFTIME 1.5

15.18.1.5 #define PATTERNNO 100

15.18.1.6 #define SYN_OUT_TME 20000.0

15.18.1.7 #define T_REPORT_TME 10000.0

15.18.1.8 #define TOTAL_TME 5000.0

15.18.2 Variable Documentation

15.18.2.1 scalar InputBaseRate = 2e-04

15.18.2.2 unsigned int iT = 0

15.18.2.3 int patFireTime

15.18.2.4 int patSetTime

15.18.2.5 scalar t = 0.0f

15.18.2.6 CStopWatch timer

15.19 CodeHelper.cc File Reference

```
#include <iostream>
#include <cstring>
#include <string>
#include <sstream>
#include <vector>
```

Classes

- class CodeHelper

Macros

- #define __CODE_HELPER_CC
- #define SAVEP(X) "(" << X << ")"
- #define OB(X) hlp.openBrace(X)
- #define CB(X) hlp.closeBrace(X)
- #define ENDL hlp.endl()

15.19.1 Macro Definition Documentation

15.19.1.1 #define __CODE_HELPER_CC

15.19.1.2 #define CB(*X*) hlp.closeBrace(X)

15.19.1.3 #define ENDL hlp.endl()

15.19.1.4 #define OB(*X*) hlp.openBrace(X)

15.19.1.5 #define SAVEP(*X*) "(" << X << ")"

15.20 ensureFtype.h File Reference

```
#include <string>
```

Functions

- void doFinal (string &code, unsigned int i, string type, unsigned int &state)
- string ensureFtype (string oldcode, string type)

Variables

- string digits = string("0123456789")

  Function for converting code to contain only explicit single precision (float) constants.
- string op = string("+-*/(<>= ,;")+string("\n")+string("\t")

15.20.1 Function Documentation

15.20.1.1 void doFinal (string & *code,* unsigned int *i,* string *type,* unsigned int & *state*)

15.20.1.2 string ensureFtype (string *oldcode,* string *type*)

15.20.2 Variable Documentation

15.20.2.1 string digits = string("0123456789")

Function for converting code to contain only explicit single precision (float) constants.

15.20.2.2 string op = string("+-*/(<>= ,;")+string("\n")+string("\t")

15.21 experiment.cu File Reference

```
#include "experiment.h"
#include <time.h>
#include <algorithm>
#include <sys/types.h>
#include <sys/stat.h>
```

Classes

- struct Parameter

Macros

- #define RAND(Y, X) Y = Y * 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767
- #define S_ISDIR(mode) (((mode) & S_IFMT) == S_IFDIR)

Typedefs

- typedef struct Parameter Parameter

Functions

- bool directoryExists (string const &path)
- bool createDirectory (string path)
- float getAverage (vector< float > &v)
- float getStdDev (vector< float > &v, float avg)
- bool printTextFile (string path)
- string getUniqueRunId ()
- void outputRunParameters ()
- bool applyInputToClassifier (UINT recordingIdx, bool usePlasticity)
- bool vectorContains (vector< int > &vec, int lookingFor)
- void setDefaultParamValues ()
- int main (int argc, char *argv[])

Variables

- Schmuker2014_classifier classifier

15.21.1 Macro Definition Documentation

15.21.1.1 #define RAND(*Y, X*) Y = Y * 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767

15.21.1.2 #define S_ISDIR(*mode*) (((mode) & S_IFMT) == S_IFDIR)

15.21.2 Typedef Documentation

15.21.2.1 typedef struct Parameter Parameter

15.21.3 Function Documentation

15.21.3.1 bool applyInputToClassifier (UINT *recordingIdx,* bool *usePlasticity*)

15.21.3.2 bool createDirectory (string *path*)

15.21.3.3 bool directoryExists (string const & *path*)

15.21.3.4 float getAverage (vector< float > & *v*)

15.21.3.5 float getStdDev (vector< float > & *v,* float *avg*)

15.21.3.6 string getUniqueRunId ( )

15.21.3.7 int main (int *argc,* char * *argv[]*)

15.21.3.8 void outputRunParameters ( )

15.21.3.9 bool printTextFile (string *path*)

15.21.3.10 void setDefaultParamValues ( )

15.21.3.11 bool vectorContains (vector< int > & *vec,* int *lookingFor*)

15.21.4 Variable Documentation

15.21.4.1 Schmuker2014_classifier classifier

15.22 experiment.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include "Schmuker2014_classifier.cu"
```

Macros

- #define divi "/"
- #define D_MAX_RANDOM_NUM 32767
- #define RECORDINGS_DIR "recordings_iris_data"
- #define CACHE_DIR "cached_iris_data"
- #define OUTPUT_DIR "output_iris"
- #define VR_DATA_FILENAME "VR-recordings-iris.data"
- #define DATASET_NAME "Iris"
- #define TOTAL_RECORDINGS 150
- #define N_FOLDING 5
- #define RECORDING_TIME_MS 1000
- #define REPEAT_LEARNING_SET 2
- #define SPIKING_ACTIVITY_THRESHOLD_HZ 5
- #define FLAG_RUN_ON_CPU 1
- #define MAX_FIRING_RATE_HZ 70
- #define MIN_FIRING_RATE_HZ 20
- #define GLOBAL_WEIGHT_SCALING 1.0
- #define WEIGHT_RN_PN 0.5
- #define CONNECTIVITY_RN_PN 0.5
- #define WEIGHT_WTA_PN_PN 0.01
- #define WEIGHT_WTA_AN_AN 0.01
- #define CONNECTIVITY_PN_PN 0.5
- #define CONNECTIVITY_AN_AN 0.5
- #define CONNECTIVITY_PN_AN 0.5
- #define MIN_WEIGHT_PN_AN 0.1
- #define MAX_WEIGHT_PN_AN 0.4
- #define WEIGHT_DELTA_PN_AN 0.04
- #define PLASTICITY_INTERVAL_MS 330

Typedefs

- typedef unsigned int UINT

Variables

- CStopWatch timer

15.22.1 Macro Definition Documentation

15.22.1.1 #define CACHE_DIR "cached_iris_data"

15.22.1.2 #define CONNECTIVITY_AN_AN 0.5

15.22.1.3 #define CONNECTIVITY_PN_AN 0.5

15.22.1.4 #define CONNECTIVITY_PN_PN 0.5

15.22.1.5 #define CONNECTIVITY_RN_PN 0.5

15.22.1.6 #define D_MAX_RANDOM_NUM 32767

15.22.1.7 #define DATASET_NAME "Iris"

15.22.1.8 #define divi "/"

15.22.1.9 #define FLAG_RUN_ON_CPU 1

15.22.1.10 #define GLOBAL_WEIGHT_SCALING 1.0

15.22.1.11 #define MAX_FIRING_RATE_HZ 70

15.22.1.12 #define MAX_WEIGHT_PN_AN 0.4

15.22.1.13 #define MIN_FIRING_RATE_HZ 20

15.22.1.14 #define MIN_WEIGHT_PN_AN 0.1

15.22.1.15 #define N_FOLDING 5

15.22.1.16 #define OUTPUT_DIR "output_iris"

15.22.1.17 #define PLASTICITY_INTERVAL_MS 330

15.22.1.18 #define RECORDING_TIME_MS 1000

15.22.1.19 #define RECORDINGS_DIR "recordings_iris_data"

15.22.1.20 #define REPEAT_LEARNING_SET 2

15.22.1.21 #define SPIKING_ACTIVITY_THRESHOLD_HZ 5

15.22.1.22 #define TOTAL_RECORDINGS 150

15.22.1.23 #define VR_DATA_FILENAME "VR-recordings-iris.data"

15.22.1.24 #define WEIGHT_DELTA_PN_AN 0.04

15.22.1.25 #define WEIGHT_RN_PN 0.5

15.22.1.26 #define WEIGHT_WTA_AN_AN 0.01

15.22.1.27 #define WEIGHT_WTA_PN_PN 0.01

15.22.2 Typedef Documentation

15.22.2.1 typedef unsigned int UINT

15.22.3 Variable Documentation

15.22.3.1 CStopWatch timer

15.23 extra_neurons.h File Reference

Functions

- n varNames clear ()
- n varNames push_back (tS("V"))

- n varTypes push_back (tS("float"))
- n varNames push_back (tS("V_NB"))
- n varNames push_back (tS("tSpike_NB"))
- n varNames push_back (tS("__regime_val"))
- n varTypes push_back (tS("int"))
- n pNames push_back (tS("VReset_NB"))
- n pNames push_back (tS("VThresh_NB"))
- n pNames push_back (tS("tRefrac_NB"))
- n pNames push_back (tS("VRest_NB"))
- n pNames push_back (tS("TAUm_NB"))
- n pNames push_back (tS("Cm_NB"))
- nModels push_back (n)
- n varNames push_back (tS("count_t_NB"))
- n pNames push_back (tS("max_t_NB"))

Variables

- n simCode

15.23.1 Function Documentation

15.23.1.1 ps dpNames clear ( )

15.23.1.2 n varNames push_back (tS("V"))

15.23.1.3 ps varTypes push_back (tS("float"))

15.23.1.4 n varNames push_back (tS("V_NB"))

15.23.1.5 n varNames push_back (tS("tSpike_NB"))

15.23.1.6 n varNames push_back (tS("__regime_val"))

15.23.1.7 n varTypes push_back (tS("int"))

15.23.1.8 n pNames push_back (tS("VReset_NB"))

15.23.1.9 n pNames push_back (tS("VThresh_NB"))

15.23.1.10 n pNames push_back (tS("tRefrac_NB"))

15.23.1.11 n pNames push_back (tS("VRest_NB"))

15.23.1.12 n pNames push_back (tS("TAUm_NB"))

15.23.1.13 n pNames push_back (tS("Cm_NB"))

15.23.1.14 nModels push_back (n)

15.23.1.15 n varNames push_back (tS("count_t_NB"))

15.23.1.16 n pNames push_back (tS("max_t_NB"))

15.23.2 Variable Documentation

15.23.2.1 n simCode

Initial value:

```
= tS(" \
          $(V) = -1000000; \
          if ($(__regime_val)==1) { \n \
$(V_NB) += (Isyn_NB/$(Cm_NB)+($(VRest_NB)-$(V_NB))/$(TAUm_NB))*DT; \n \
                  if ($(V_NB)>$(VThresh_NB)) { \n \
$(V_NB) = $(VReset_NB); \n \
$(tSpike_NB) = t; \n \
                  $(V) = 100000; \
$(__regime_val) = 2; \n \
} \n \
} \n \
if ($(__regime_val)==2) { \n \
if (t-$(tSpike_NB) > $(tRefrac_NB)) { \n \
$(__regime_val) = 1; \n \
} \n \
} \n \
")
```

15.24 extra_postsynapses.h File Reference

Functions

- ps varNames clear ()
- ps varNames push_back (tS("g_PS"))
- ps varTypes push_back (tS("float"))
- ps pNames push_back (tS("tau_syn_PS"))
- ps pNames push_back (tS("E_PS"))
- postSynModels push_back (ps)

Variables

- ps postSyntoCurrent
- ps postSynDecay

15.24.1 Function Documentation

15.24.1.1 ps varNames clear ( )

15.24.1.2 ps varNames push_back (tS("g_PS"))

15.24.1.3 ps varTypes push_back (tS("float"))

15.24.1.4 ps pNames push_back (tS("tau_syn_PS"))

15.24.1.5 ps pNames push_back (tS("E_PS"))

15.24.1.6 postSynModels push_back (ps)

15.24.2 Variable Documentation

15.24.2.1 ps postSynDecay

Initial value:

```
= tS(" \
          $(g_PS) += (-$(g_PS)/$(tau_syn_PS))*DT; \n \
                  $(inSyn) = 0; \
        ")
```

15.24.2.2 ps postSyntoCurrent

Initial value:

```
= tS(" \
  0; \n \
        float Isyn_NB = 0; \n \
         { \n \
        float v_PS = lV_NB; \n \
         float g_in_PS = $(inSyn); \
$(g_PS) = $(g_PS)+g_in_PS; \n \
Isyn_NB += ($(g_PS)*($(E_PS)-v_PS)); \n \
          } \n \
")
```

15.25 extra_postsynapses.h File Reference

Functions

- ps varNames clear ()
- postSynModels push_back (ps)
- ps varNames push_back (tS("g_PS"))
- ps varTypes push_back (tS("float"))
- ps pNames push_back (tS("tau_syn_PS"))
- ps pNames push_back (tS("E_PS"))

Variables

- postSynModel ps
- ps postSyntoCurrent
- ps postSynDecay

15.25.1 Function Documentation

15.25.1.1 ps varNames clear ( )

15.25.1.2 postSynModels push_back (ps)

15.25.1.3 ps varNames push_back (tS("g_PS"))

15.25.1.4 ps varTypes push_back (tS("float"))

15.25.1.5 ps pNames push_back (tS("tau_syn_PS"))

15.25.1.6 ps pNames push_back (tS("E_PS"))

15.25.2 Variable Documentation

15.25.2.1 ps postSynDecay

Initial value:

```
= tS(" \
                $(inSyn) = 0; \
        ")
```

15.25.2.2 ps postSyntoCurrent

Initial value:

```
= tS(" \
  0; \n \
        float I_sum_NB = 0; \n \
         { \n \
        float in_PS = $(inSyn); \n \
        I_sum_NB += (in_PS); \n \
          } \n \
")
```

15.25.2.3 postSynModel ps

15.26 extra_weightupdates.h File Reference

15.27 GA.cc File Reference

```
#include <algorithm>
```

Classes

- struct errTupel

Functions

- int compareErrTupel (const void *x, const void *y)
- void procreatePop (FILE *osb)

15.27.1 Function Documentation

15.27.1.1 int compareErrTupel (const void * *x*, const void * *y*)

15.27.1.2 void procreatePop (FILE * *osb*)

15.28 gauss.cc File Reference

Contains the implementation of the Gaussian random number generator class randomGauss.

```
#include "gauss.h"
```

Macros

- #define GAUSS_CC

  macro for avoiding multiple inclusion during compilation

15.28.1 Detailed Description

Contains the implementation of the Gaussian random number generator class randomGauss.

15.28.2 Macro Definition Documentation

15.28.2.1 #define GAUSS_CC

macro for avoiding multiple inclusion during compilation

15.29 gauss.h File Reference

Random number generator for Gaussian random variable with mean 0 and standard deviation 1.

```
#include <cmath>
#include "randomGen.h"
#include "randomGen.cc"
#include "gauss.cc"
```

Classes

- class randomGauss

  Class random Gauss encapsulates the methods for generating random neumbers with Gaussian distribution.

Macros

- #define GAUSS_H

  macro for avoiding multiple inclusion during compilation

15.29.1 Detailed Description

Random number generator for Gaussian random variable with mean 0 and standard deviation 1.

This random number generator is based on the ratio of uniforms method by A.J. Kinderman and J.F. Monahan and improved with quadratic boundind curves by J.L. Leva. Taken from Algorithm 712 ACM Trans. Math. Softw. 18 p. 454. (the necessary uniform random variables are obtained from the ISAAC random number generator; C++ Implementation by Quinn Tyler Jackson of the RG invented by Bob Jenkins Jr.).

15.29.2 Macro Definition Documentation

15.29.2.1 #define GAUSS_H

macro for avoiding multiple inclusion during compilation

15.30 gen_input_structured.cc File Reference

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "randomGen.cc"
```

Functions

- int main (int argc, char ∗argv[])

Variables

- randomGen R

15.30.1 Function Documentation

15.30.1.1 int main (int *argc,* char * *argv[]*)

15.30.2 Variable Documentation

15.30.2.1 randomGen R

15.31 gen_kcdn_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

Functions

- int main (int argc, char *argv[])

Variables

- randomGen R
- randomGauss RG

15.31.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between KCs and DNs (detector neurons) in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

15.31.2 Function Documentation

15.31.2.1 int main (int *argc,* char * *argv[]*)

15.31.3 Variable Documentation

15.31.3.1 randomGen R

15.31.3.2 randomGauss RG

15.32 gen_kcdn_syns_fixto10K.cc File Reference

```
#include <iostream>
```

```
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

Functions

- int main (int argc, char ∗argv[])

Variables

- randomGen R
- randomGen R2
- randomGauss RG

15.32.1 Function Documentation

15.32.1.1 int main (int *argc,* char ∗ *argv[]*)

15.32.2 Variable Documentation

15.32.2.1 randomGen R

15.32.2.2 randomGen R2

15.32.2.3 randomGauss RG

15.33 gen_pnkc_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "randomGen.h"
#include "gauss.h"
#include "randomGen.cc"
```

Functions

- int main (int argc, char ∗argv[])

Variables

- randomGen R
- randomGauss RG

15.33.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and KCs in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

15.33.2 Function Documentation

15.33.2.1 int main (int *argc,* char * *argv[]*)

15.33.3 Variable Documentation

15.33.3.1 randomGen R

15.33.3.2 randomGauss RG

15.34 gen_pnkc_syns_indivID.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <cstdint>
#include "randomGen.h"
#include "gauss.h"
#include "simpleBit.h"
#include "randomGen.cc"
```

Functions

- int main (int argc, char *argv[])

Variables

- randomGen R
- randomGauss RG

15.34.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and KCs in the model. In contrast to the gen_pnkc_syns.cc tool, here the output is in a format that is suited for the "INDIVIDUALID" method for specifying connectivity. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

15.34.2 Function Documentation

15.34.2.1 int main (int *argc,* char * *argv[]*)

15.34.3 Variable Documentation

15.34.3.1 randomGen R

15.34.3.2 randomGauss RG

15.35 gen_pnlhi_syns.cc File Reference

This file is part of a tool chain for running the classol/MBody1 example model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
```

Functions

- int main (int argc, char ∗argv[])

15.35.1 Detailed Description

This file is part of a tool chain for running the classol/MBody1 example model.

This file compiles to a tool to generate appropriate connectivity patterns between PNs and LHIs (lateral horn interneurons) in the model. The connectivity is saved to file and can then be read by the classol method for reading this connectivity.

15.35.2 Function Documentation

15.35.2.1 int main (int *argc,* char ∗ *argv[]*)

15.36 gen_syns_sparse.cc File Reference

This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity.

```
#include <iostream>
#include <fstream>
#include <string.h>
#include "randomGen.h"
#include "gauss.h"
#include <vector>
```

Functions

- int main (int argc, char ∗argv[])

Variables

- randomGen R
- randomGauss RG

15.36.1 Detailed Description

This file generates the arrays needed for sparse connectivity. The connectivity is saved to a file for each variable and can then be read to fill the struct of connectivity.

15.36.2 Function Documentation

15.36.2.1 int main (int *argc,* char ∗ *argv[]*)

15.36.3 Variable Documentation

15.36.3.1 randomGen R

15.36.3.2 randomGauss RG

15.37 gen_syns_sparse_izhModel.cc File Reference

This file is part of a tool chain for running the Izhikevich network model.

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <vector>
#include "randomGen.h"
#include "randomGen.cc"
```

Functions

- int printVector (vector< unsigned int > &)
- int printVector (vector< double > &)
- int main (int argc, char *argv[])

Variables

- randomGen R
- randomGen Rind
- double gsyn
- double * garray
- unsigned int * ind
- double * garray_ee
- std::vector< double > g_ee
- std::vector< unsigned int > indInG_ee
- std::vector< unsigned int > ind_ee
- double * garray_ei
- std::vector< double > g_ei
- std::vector< unsigned int > indInG_ei
- std::vector< unsigned int > ind_ei
- double * garray_ie
- std::vector< double > g_ie
- std::vector< unsigned int > indInG_ie
- std::vector< unsigned int > ind_ie
- double * garray_ii
- std::vector< double > g_ii
- std::vector< unsigned int > indInG_ii
- std::vector< unsigned int > ind_ii

15.37.1 Detailed Description

This file is part of a tool chain for running the Izhikevich network model.

15.37.2 Function Documentation

15.37.2.1 int main (int *argc,* char * *argv[]*)

15.37.2.2 int printVector (vector< unsigned int > & *v*)

15.37.2.3 int printVector (vector< double > & *v*)

15.37.3 Variable Documentation

15.37.3.1 std::vector<double> g_ee

15.37.3.2 std::vector<double> g_ei

15.37.3.3 std::vector<double> g_ie

15.37.3.4 std::vector<double> g_ii

15.37.3.5 double* garray

15.37.3.6 double* garray_ee

15.37.3.7 double* garray_ei

15.37.3.8 double* garray_ie

15.37.3.9 double* garray_ii

15.37.3.10 double gsyn

15.37.3.11 unsigned int* ind

15.37.3.12 std::vector<unsigned int> ind_ee

15.37.3.13 std::vector<unsigned int> ind_ei

15.37.3.14 std::vector<unsigned int> ind_ie

15.37.3.15 std::vector<unsigned int> ind_ii

15.37.3.16 std::vector<unsigned int> indInG_ee

15.37.3.17 std::vector<unsigned int> indInG_ei

15.37.3.18 std::vector<unsigned int> indInG_ie

15.37.3.19 std::vector<unsigned int> indInG_ii

15.37.3.20 randomGen R

15.37.3.21 randomGen Rind

15.38 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include "toString.h"
#include <sys/stat.h>
```

Functions

- int main (int argc, char ∗argv[])

  Main entry point for generate_run.

15.38.1 Function Documentation

15.38.1.1 int main (int *argc*, char ∗ *argv[]*)

Main entry point for generate_run.

15.39 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <locale>
#include <sys/stat.h>
```

Macros

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<typename T >
  std::string toString (T t)

  Template function for string conversion.
- string toUpper (string s)
- string toLower (string s)
- unsigned int openFileGetMax (unsigned int ∗array, unsigned int size, string name)
- int main (int argc, char ∗argv[])

  Main entry point for generate_run.

15.39.1 Macro Definition Documentation

15.39.1.1 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.39.2 Function Documentation

15.39.2.1 int main (int *argc,* char * *argv[]*)

Main entry point for generate_run.

15.39.2.2 unsigned int openFileGetMax (unsigned int * *array,* unsigned int *size,* string *name*)

15.39.2.3 string toLower (string *s*)

15.39.2.4 template<typename T > std::string toString (T *t*)

Template function for string conversion.

15.39.2.5 string toUpper (string *s*)

15.40 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
```

Macros

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<typename T >
  std::string toString (T t)

  template function for string conversion from const char to C++ string*
- string toUpper (string s)
- string toLower (string s)
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.40.1 Macro Definition Documentation

15.40.1.1 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.40.2 Function Documentation

15.40.2.1 int main (int *argc,* char * *argv[]*)

Main entry point for generate_run.

15.40.2.2 string toLower (string *s*)

15.40.2.3 template<typename T > std::string toString (T *t*)

template function for string conversion from const char* to C++ string

15.40.2.4 string toUpper (string *s*)

15.41 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
```

Macros

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<typename T >
  std::string toString (T t)

  template function for string conversion from const char to C++ string*
- string toUpper (string s)
- string toLower (string s)
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.41.1 Macro Definition Documentation

15.41.1.1 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.41.2 Function Documentation

15.41.2.1 int main (int *argc,* char * *argv[]*)

Main entry point for generate_run.

15.41.2.2 string toLower (string *s*)

15.41.2.3 template<typename T > std::string toString (T *t*)

template function for string conversion from const char* to C++ string

15.41.2.4 string toUpper (string *s*)

15.42 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
```

Macros

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<typename T >
  std::string toString (T t)

  template function for string conversion from const char to C++ string*
- string toUpper (string s)
- string toLower (string s)
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.42.1 Macro Definition Documentation

15.42.1.1 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.42.2 Function Documentation

15.42.2.1 int main (int *argc,* char * *argv[]*)

Main entry point for generate_run.

15.42.2.2 string toLower (string *s*)

15.42.2.3 template<typename T > std::string toString (T *t*)

template function for string conversion from const char* to C++ string

15.42.2.4 string toUpper (string *s*)

15.43 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <cfloat>
#include <locale>
#include <sys/stat.h>
```

Macros

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<typename T >
  std::string toString (T t)

  Template function for string conversion.
- string toUpper (string s)
- string toLower (string s)
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.43.1 Macro Definition Documentation

15.43.1.1 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.43.2 Function Documentation

15.43.2.1 int main (int *argc,* char * *argv[]*)

Main entry point for generate_run.

15.43.2.2 string toLower (string *s*)

15.43.2.3 template<typename T > std::string toString (T *t*)

Template function for string conversion.

15.43.2.4 string toUpper (string *s*)

15.44 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <sys/stat.h>
```

Functions

- template<typename T >
  std::string toString (T t)

  Template function for string conversion.
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.44.1 Function Documentation

15.44.1.1 int main (int *argc*, char * *argv[]*)

Main entry point for generate_run.

15.44.1.2 template<typename T > std::string toString (T *t*)

Template function for string conversion.

15.45 generate_run.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <sys/stat.h>
```

Functions

- template<typename T >
  std::string toString (T t)

  Template function for string conversion.
- int main (int argc, char *argv[])

  Main entry point for generate_run.

15.45.1 Function Documentation

15.45.1.1 int main (int *argc*, char * *argv[]*)

Main entry point for generate_run.

15.45.1.2 template<typename T > std::string toString (T *t*)

Template function for string conversion.

15.46 generateALL.cc File Reference

Main file combining the code for code generation. Part of the code generation section.

```
#include "global.h"
#include "modelSpec.h"
#include "modelSpec.cc"
#include "generateKernels.cc"
#include "generateRunner.cc"
#include "generateCPU.cc"
#include <sys/stat.h>
```

Functions

- void generate_model_runner (NNmodel &model, string path)

  This function will call the necessary sub-functions to generate the code for simulating a model.
- int chooseDevice (ostream &mos, NNmodel *&model, string path)

  Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.
- int main (int argc, char *argv[])

  Main entry point for the generateALL executable that generates the code for GPU and CPU.

15.46.1 Detailed Description

Main file combining the code for code generation. Part of the code generation section.

The file includes separate files for generating kernels (generateKernels.cc), generating the CPU side code for running simulations on either the CPU or GPU (generateRunner.cc) and for CPU-only simulation code (generateCPU.cc).

15.46.2 Function Documentation

15.46.2.1 int chooseDevice (ostream & *mos,* NNmodel *& *model,* string *path*)

Helper function that prepares data structures and detects the hardware properties to enable the code generation code that follows.

The main tasks in this function are the detection and characterization of the GPU device present (if any), choosing which GPU device to use, finding and appropriate block size, taking note of the major and minor version of the CUDA enabled device chosen for use, and populating the list of standard neuron models. The chosen device number is returned.

Parameters

mos	output stream for messages
model	the nn model we are generating code for
path	path the generated code will be deposited

15.46.2.2 void generate_model_runner (NNmodel & *model,* string *path*)

This function will call the necessary sub-functions to generate the code for simulating a model.

Parameters

model	Model description
path	Path where the generated code will be deposited

15.46.2.3 int main (int *argc*, char * *argv[]*)

Main entry point for the generateALL executable that generates the code for GPU and CPU.

The main function is the entry point for the code generation engine. It prepares the system and then invokes generate_model_runner to inititate the different parts of actual code generation.

Parameters

argc	number of arguments; expected to be 2
argv	Arguments; expected to contain the target directory for code generation.

15.47 generateCPU.cc File Reference

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

```
#include <string>
#include "CodeHelper.cc"
#include <cfloat>
```

Functions

- void genNeuronFunction (NNmodel &model, string &path, ostream &mos)

  Function that generates the code of the function the will simulate all neurons on the CPU.
- void generate_process_presynaptic_events_code_CPU (ostream &os, NNmodel &model, unsigned int src, unsigned int trg, int i, string &localID, unsigned int inSynNo, string postfix)

  Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.
- void genSynapseFunction (NNmodel &model, string &path, ostream &mos)

  Function that generates code that will simulate all synapses of the model on the CPU.

15.47.1 Detailed Description

Functions for generating code that will run the neuron and synapse simulations on the CPU. Part of the code generation section.

15.47.2 Function Documentation

15.47.2.1 void generate_process_presynaptic_events_code_CPU (ostream & *os*, NNmodel & *model*, unsigned int *src*, unsigned int *trg*, int *i*, string & *localID*, unsigned int *inSynNo*, string *postfix*)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

Parameters

os	output stream for code

model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed
localID	the variable name of the local ID of the thread within the synapse group
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron population
postfix	whether to generate code for true spikes or spike type events

15.47.2.2 void genNeuronFunction (NNmodel & *model,* string & *path,* ostream & *mos*)

Function that generates the code of the function the will simulate all neurons on the CPU.

Parameters

model	Model description
path	output stream for code
mos	output stream for messages

15.47.2.3 void genSynapseFunction (NNmodel & *model,* string & *path,* ostream & *mos*)

Function that generates code that will simulate all synapses of the model on the CPU.

Parameters

model	Model description
path	Path for code generation
mos	output stream for messages

15.48 generateKernels.cc File Reference

Contains functions that generate code for CUDA kernels. Part of the code generation section.

```
#include <string>
#include "CodeHelper.cc"
#include "global.h"
```

Functions

- void genNeuronKernel (NNmodel &model, string &path, ostream &mos)

  Function for generating the CUDA kernel that simulates all neurons in the model.

- void generate_process_presynaptic_events_code (ostream &os, NNmodel &model, unsigned int src, unsigned int trg, int i, string &localID, unsigned int inSynNo, string postfix)

  Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

- void genSynapseKernel (NNmodel &model, string &path, ostream &mos)

  Function for generating a CUDA kernel for simulating all synapses.

Variables

- short * isGrpVarNeeded
- CodeHelper hlp

15.48.1 Detailed Description

Contains functions that generate code for CUDA kernels. Part of the code generation section.

15.48.2 Function Documentation

15.48.2.1 void generate_process_presynaptic_events_code (ostream & *os,* NNmodel & *model,* unsigned int *src,* unsigned int *trg,* int *i,* string & *localID,* unsigned int *inSynNo,* string *postfix*)

Function for generating the CUDA synapse kernel code that handles presynaptic spikes or spike type events.

Parameters

os	output stream for code
model	the neuronal network model to generate code for
src	the number of the src neuron population
trg	the number of the target neuron population
i	the index of the synapse group being processed
localID	the variable name of the local ID of the thread within the synapse group
inSynNo	the ID number of the current synapse population as the incoming population to the target neuron population
postfix	whether to generate code for true spikes or spike type events

15.48.2.2 void genNeuronKernel (NNmodel & *model,* string & *path,* ostream & *mos*)

Function for generating the CUDA kernel that simulates all neurons in the model.

The code generated upon execution of this function is for defining GPU side global variables that will hold model state in the GPU global memory and for the actual kernel function for simulating the neurons for one time step. Binary flag for the sparse synapses to use atomic operations when the number of connections is bigger than the block size, and shared variables otherwise

Parameters

model	Model description
path	path for code output
mos	output stream for messages

15.48.2.3 void genSynapseKernel (NNmodel & *model,* string & *path,* ostream & *mos*)

Function for generating a CUDA kernel for simulating all synapses.

This functions generates code for global variables on the GPU side that are synapse-related and the actual CUDA kernel for simulating one time step of the synapses. $<$ "id" if first synapse group, else "lid". lid =(thread index- last thread of the last synapse group)

Parameters

model	Model description
path	Path for code output
mos	output stream for messages

15.48.3 Variable Documentation

15.48.3.1 CodeHelper hlp

15.48.3.2 short* isGrpVarNeeded

15.49 generateRunner.cc File Reference

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

```
#include <cfloat>
```

Functions

- void genRunner (NNmodel &model, string path, ostream &mos)

  A function that generates predominantly host-side code.
- void genRunnerGPU (NNmodel &model, string &path, ostream &mos)

  A function to generate the code that simulates the model on the GPU.

15.49.1 Detailed Description

Contains functions to generate code for running the simulation on the GPU, and for I/O convenience functions between GPU and CPU space. Part of the code generation section.

15.49.2 Function Documentation

15.49.2.1 void genRunner (NNmodel & *model,* string *path,* ostream & *mos*)

A function that generates predominantly host-side code.

In this function host-side functions and other code are generated, including: Global host variables, "allocatedMem()" function for allocating memories, "freeMem" function for freeing the allocated memories, "initialize" for initializing host variables, "gFunc" and "initGRaw()" for use with plastic synapses if such synapses exist in the model. Method for cleaning up and resetting device while quitting GeNN

Parameters

model	Model description
path	path for code generation
mos	output stream for messages

15.49.2.2 void genRunnerGPU (NNmodel & *model,* string & *path,* ostream & *mos*)

A function to generate the code that simulates the model on the GPU.

The function generates functions that will spawn kernel grids onto the GPU (but not the actual kernel code which is generated in "genNeuronKernel()" and "genSynpaseKernel()"). Generated functions include "copyGToDevice()", "copyGFromDevice()", "copyStateToDevice()", "copyStateFromDevice()", "copySpikesFromDevice()", "copySpike↩ NFromDevice()" and "stepTimeGPU()". The last mentioned function is the function that will initialize the execution on the GPU in the generated simulation engine. All other generated functions are "convenience functions" to handle data transfer from and to the GPU.

Parameters

model	Model description
path	path for code generation
mos	output stream for messages

15.50 GeNNHelperKrnls.cu File Reference

```
#include <curand_kernel.h>
```

Macros

- #define BlkSz 256

Functions

- __global__ void setup_kernel (curandState *state, unsigned long seed, int sizeofResult)
- template<class T >
  __global__ void generate_random_gpuInput_xorwow (curandState *state, T *result, int sizeofResult, T Rstrength, T Rshift)
- void xorwow_setup (curandState *devStates, long int sampleSize)

15.50.1 Macro Definition Documentation

15.50.1.1 #define BlkSz 256

15.50.2 Function Documentation

15.50.2.1 template<class T > __global__ void generate_random_gpuInput_xorwow (curandState * *state,* T * *result,* int *sizeofResult,* T *Rstrength,* T *Rshift*)

15.50.2.2 __global__ void setup_kernel (curandState * *state,* unsigned long *seed,* int *sizeofResult*)

15.50.2.3 void xorwow_setup (curandState * *devStates,* long int *sampleSize*)

15.51 global.h File Reference

Global header file containing a few global variables. Part of the code generation section.

```
#include <iostream>
#include <cstring>
#include <string>
#include <sstream>
#include <vector>
#include <cmath>
#include <cuda_runtime.h>
#include "toString.h"
#include <stdint.h>
```

Macros

- #define _GLOBAL_H_

  macro for avoiding multiple inclusion during compilation

Variables

- int neuronBlkSz
- int synapseBlkSz
- int learnBlkSz
- cudaDeviceProp * deviceProp
- int theDev
- int hostCount

Global variable containing the number of hosts within the local compute cluster.

- int deviceCount

  Global variable containing the number of CUDA devices found on this host.

- int optimiseBlockSize = 1

  Flag for signalling whether or not block size optimisation should be performed.

- int UIntSz = sizeof(unsigned int) ∗ 8

  size of the unsigned int variable type on the local architecture

- int logUIntSz = (int) (logf((float) UIntSz) / logf(2.0f) + 1e-5f)

  logarithm of the size of the unsigned int variable type on the local architecture

- double asGoodAsZero = 1e-19

  Global variable that is used when detecting close to zero values, for example when setting sparse connectivity from a dense matrix.

15.51.1 Detailed Description

Global header file containing a few global variables. Part of the code generation section.

This global header file also takes care of including some generally used cuda support header files.

15.51.2 Macro Definition Documentation

15.51.2.1 #define _GLOBAL_H_

macro for avoiding multiple inclusion during compilation

15.51.3 Variable Documentation

15.51.3.1 double asGoodAsZero = 1e-19

Global variable that is used when detecting close to zero values, for example when setting sparse connectivity from a dense matrix.

15.51.3.2 int deviceCount

Global variable containing the number of CUDA devices found on this host.

15.51.3.3 cudaDeviceProp∗ deviceProp

15.51.3.4 int hostCount

Global variable containing the number of hosts within the local compute cluster.

15.51.3.5 int learnBlkSz

15.51.3.6 int logUIntSz = (int) (logf((float) UIntSz) / logf(2.0f) + 1e-5f)

logarithm of the size of the unsigned int variable type on the local architecture

15.51.3.7 int neuronBlkSz

15.51.3.8 int optimiseBlockSize = 1

Flag for signalling whether or not block size optimisation should be performed.

15.51.3.9 int synapseBlkSz

15.51.3.10 int theDev

15.51.3.11 int UIntSz = sizeof(unsigned int) * 8

size of the unsigned int variable type on the local architecture

15.52 helper.h File Reference

```
#include <vector>
```

Classes

- struct inputSpec

Functions

- ostream & operator<< (ostream &os, inputSpec &I)
- void write_para ()
- void single_var_init_fullrange (int n)
- void single_var_reinit (int n, double fac)
- void copy_var (int src, int trg)
- void var_init_fullrange ()
- void var_reinit (double fac)
- void truevar_init ()
- void initexpHH ()
- void truevar_initexpHH ()
- void runexpHH (float t)
- void initI (inputSpec &I)

Variables

- double sigGNa = 0.1
- double sigENa = 10.0
- double sigGK = 0.1
- double sigEK = 10.0
- double sigGl = 0.1
- double sigEl = 10.0
- double sigC = 0.1
- const double limit [7][2]
- double Vexp
- double mexp
- double hexp
- double nexp
- double gNaexp
- double ENaexp
- double gKexp
- double EKexp
- double glexp
- double Elexp
- double Cexp

15.52.1 Function Documentation

15.52.1.1 void copy_var (int *src,* int *trg*)

15.52.1.2 void initexpHH ( )

15.52.1.3 void initl (inputSpec & *I*)

15.52.1.4 ostream& operator<< (ostream & *os,* inputSpec & *I*)

15.52.1.5 void runexpHH (float *t*)

15.52.1.6 void single_var_init_fullrange (int *n*)

15.52.1.7 void single_var_reinit (int *n,* double *fac*)

15.52.1.8 void truevar_init ( )

15.52.1.9 void truevar_initexpHH ( )

15.52.1.10 void var_init_fullrange ( )

15.52.1.11 void var_reinit (double *fac*)

15.52.1.12 void write_para ( )

15.52.2 Variable Documentation

15.52.2.1 double Cexp

15.52.2.2 double EKexp

15.52.2.3 double Elexp

15.52.2.4 double ENaexp

15.52.2.5 double gKexp

15.52.2.6 double glexp

15.52.2.7 double gNaexp

15.52.2.8 double hexp

15.52.2.9 const double limit[7][2]

Initial value:

```
= {{1.0, 200.0},
                          {0.0, 100.0},
                          {1.0, 100.0},
                          {-100.0, -20.0},
                          {1.0, 50.0},
                          {-100.0, -20.0},
                          {1e-1, 10.0}}
```

15.52.2.10 double mexp

15.52.2.11 double nexp

15.52.2.12 double sigC = 0.1

15.52.2.13 double sigEK = 10.0

15.52.2.14 double sigEl = 10.0

15.52.2.15 double sigENa = 10.0

15.52.2.16 double sigGK = 0.1

15.52.2.17 double sigGl = 0.1

15.52.2.18 double sigGNa = 0.1

15.52.2.19 double Vexp

15.53 HHVClamp.cc File Reference

This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques.

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "HHVClampParameters.h"
```

Macros

- #define DT 0.25

  This defines the global time step at which the simulation will run.

Functions

- void modelDefinition (NNmodel &model)

  This function defines the HH model with variable parameters.

Variables

- double myHH_ini [11]
- double * myHH_p = NULL

15.53.1 Detailed Description

This file contains the model definition of HHVClamp model. It is used in both the GeNN code generation and the user side simulation code. The HHVClamp model implements a population of unconnected Hodgkin-Huxley neurons that evolve to mimick a model run on the CPU, using genetic algorithm techniques.

15.53.2 Macro Definition Documentation

15.53.2.1 #define DT 0.25

This defines the global time step at which the simulation will run.

15.53.3 Function Documentation

15.53.3.1 void modelDefinition (NNmodel & *model*)

This function defines the HH model with variable parameters.

15.53.4 Variable Documentation

15.53.4.1 double myHH_ini[11]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207,
  120.0,
  55.0,
  36.0,
  -72.0,
  0.3,
  -50.0,
  1.0
}
```

15.53.4.2 double* myHH_p = NULL

15.54 HHVClampParameters.h File Reference

Macros

- #define NPOP 5000
- #define TOTALT 1000
- #define fixGPU 0

15.54.1 Macro Definition Documentation

15.54.1.1 #define fixGPU 0

15.54.1.2 #define NPOP 5000

15.54.1.3 #define TOTALT 1000

15.55 hr_time.cpp File Reference

This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock.

```
#include <cstdio>
#include "hr_time.h"
```

Macros

- #define HR_TIMER

15.55.1 Detailed Description

This file contains the implementation of the CStopWatch class that provides a simple timing tool based on the system clock.

15.55.2 Macro Definition Documentation

15.55.2.1 #define HR_TIMER

15.56 hr_time.h File Reference

This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock.

```
#include <sys/time.h>
```

Classes

- struct stopWatch
- class CStopWatch

15.56.1 Detailed Description

This header file contains the definition of the CStopWatch class that implements a simple timing tool using the system clock.

15.57 lzh_sim_sparse.cu File Reference

```
#include <iostream>
#include <fstream>
#include "Izh_sparse_sim.h"
#include "../GeNNHelperKrnls.cu"
```

Functions

- int main (int argc, char ∗argv[])

15.57.1 Function Documentation

15.57.1.1 int main (int *argc,* char ∗ *argv[]*)

15.58 lzh_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include <vector>
#include "sizes.h"
```

Macros

- #define DT 1.0

Functions

- void modelDefinition (NNmodel &model)

Variables

- std::vector< unsigned int > neuronPSize
- std::vector< unsigned int > neuronVSize
- std::vector< unsigned int > synapsePSize
- scalar meanInpExc = 5.0
- scalar meanInpInh = 2.0
- double * exclzh_p = NULL
- double * inhIzh_p = NULL
- double IzhExc_ini [6]
- double IzhInh_ini [6]
- double * SynIzh_p = NULL
- double * postExpP = NULL
- double * postSynV = NULL
- double SynIzh_ini [1]

15.58.1 Macro Definition Documentation

15.58.1.1 #define DT 1.0

15.58.2 Function Documentation

15.58.2.1 void modelDefinition (NNmodel & *model*)

15.58.3 Variable Documentation

15.58.3.1 double* exclzh_p = NULL

15.58.3.2 double* inhIzh_p = NULL

15.58.3.3 double IzhExc_ini[6]

Initial value:

```
={

        -65.0,
         0.0,
         0.02,
        0.2,
        -65.0,
         8.0
}
```

15.58.3.4 double IzhInh_ini[6]

Initial value:

```
={

        -65,
         0.0,
         0.02,
        0.25,
        -65.0,
         2.0
}
```

15.58.3.5 scalar meanInpExc = 5.0

15.58.3.6 scalar meanInpInh = 2.0

15.58.3.7 std::vector<unsigned int> neuronPSize

15.58.3.8 std::vector<unsigned int> neuronVSize

15.58.3.9 double* postExpP = NULL

15.58.3.10 double* postSynV = NULL

15.58.3.11 std::vector<unsigned int> synapsePSize

15.58.3.12 double SynIzh_ini[1]

Initial value:

```
= {
    0.0
}
```

15.58.3.13 double* SynIzh_p = NULL

15.59 Izh_sparse_model.cc File Reference

```
#include "Izh_sparse_CODE/runner.cc"
#include "../../lib/include/numlib/randomGen.h"
#include "../../lib/include/numlib/gauss.h"
#include "Izh_sparse_model.h"
```

Macros

- #define _IZH_SPARSE_MODEL_CC_

Variables

- randomGauss RG
- randomGen R

15.59.1 Macro Definition Documentation

15.59.1.1 #define _IZH_SPARSE_MODEL_CC_

15.59.2 Variable Documentation

15.59.2.1 randomGen R

15.59.2.2 randomGauss RG

15.60 Izh_sparse_model.h File Reference

Classes

- class classIzh

15.61 Izh_sparse_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "Izh_sparse.cc"
#include "Izh_sparse_model.cc"
```

Macros

- #define DBG_SIZE 5000
- #define T_REPORT_TME 5000.0
- #define TOTAL_TME 5000.0

Variables

- float t = 0.0f
- unsigned int iT = 0
- CStopWatch timer

15.61.1 Macro Definition Documentation

15.61.1.1 #define DBG_SIZE 5000

15.61.1.2 #define T_REPORT_TME 5000.0

15.61.1.3 #define TOTAL_TME 5000.0

15.61.2 Variable Documentation

15.61.2.1 unsigned int iT = 0

15.61.2.2 float t = 0.0f

15.61.2.3 CStopWatch timer

15.62 map_classol.cc File Reference

```
#include "map_classol.h"
#include "MBody1_CODE/runner.cc"
```

Macros

- #define _MAP_CLASSOL_CC_

  macro for avoiding multiple inclusion during compilation

15.62.1 Macro Definition Documentation

15.62.1.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

15.63 map_classol.cc File Reference

```
#include "map_classol.h"
#include "MBody_delayedSyn_CODE/runner.cc"
```

Macros

- #define _MAP_CLASSOL_CC_

  macro for avoiding multiple inclusion during compilation

15.63.1 Macro Definition Documentation

15.63.1.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

15.64 map_classol.cc File Reference

```
#include "map_classol.h"
#include "MBody_individualID_CODE/runner.cc"
```

Macros

- #define _MAP_CLASSOL_CC_

  macro for avoiding multiple inclusion during compilation

15.64.1 Macro Definition Documentation

15.64.1.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

15.65 map_classol.cc File Reference

```
#include "MBody_userdef_CODE/runner.cc"
#include "sparseUtils.cc"
#include "map_classol.h"
```

Macros

- #define _MAP_CLASSOL_CC_

  macro for avoiding multiple inclusion during compilation

15.65.1 Macro Definition Documentation

15.65.1.1 #define _MAP_CLASSOL_CC_

macro for avoiding multiple inclusion during compilation

15.66 map_classol.h File Reference

Classes

- class classol

  This class cpontains the methods for running the MBody1 example model.

15.67 map_classol.h File Reference

Classes

- class classol

  This class cpontains the methods for running the MBody1 example model.

15.68 map_classol.h File Reference

Classes

- class classol

  This class cpontains the methods for running the MBody1 example model.

15.69 map_classol.h File Reference

Classes

- class classol

  This class cpontains the methods for running the MBody1 example model.

15.70 MBody1.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "../../userproject/include/sizes.h"
```

Macros

- #define DT 0.1

  This defines the global time step at which the simulation will run.

Functions

- void modelDefinition (NNmodel &model)

  This function defines the MBody1 model, and it is a good example of how networks should be defined.

Variables

- double myPOI_p [4]
- double myPOI_ini [4]
- double stdTM_p [7]
- double stdTM_ini [4]

- double myPNKC_p [3]
- double postExpPNKC [2]
- double myPNLHI_p [3]
- double postExpPNLHI [2]
- double myLHIKC_p [4]
- double gLHIKC = 0.006
- double postExpLHIKC [2]
- double myKCDN_p [13]
- double postExpKCDN [2]
- double myDNDN_p [4]
- double gDNDN = 0.01
- double postExpDNDN [2]
- double * postSynV = NULL

15.70.1 Macro Definition Documentation

15.70.1.1 #define DT 0.1

This defines the global time step at which the simulation will run.

15.70.2 Function Documentation

15.70.2.1 void modelDefinition (NNmodel & *model*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

15.70.3 Variable Documentation

15.70.3.1 double gDNDN = 0.01

15.70.3.2 double gLHIKC = 0.006

15.70.3.3 double myDNDN_p[4]

Initial value:

```
= {
  -92.0,
  -30.0,
  8.0,
  50.0
}
```

15.70.3.4 double myKCDN_p[13]

Initial value:

```
= {
  0.0,
  -20.0,
  5.0,
  25.0,
  100.0,
  50000.0,
  100000.0,
  100.0,
  0.06,
  0.03,
  33.33,
  10.0,

  0.00006
}
```

15.70.3.5 double myLHIKC_p[4]

Initial value:

```
= {
  -92.0,
  -40.0,
  3.0,
  50.0
}
```

15.70.3.6 double myPNKC_p[3]

Initial value:

```
= {
  0.0,
  -20.0,
  1.0
}
```

15.70.3.7 double myPNLHI_p[3]

Initial value:

```
= {
  0.0,
  -20.0,
  1.0
}
```

15.70.3.8 double myPOI_ini[4]

Initial value:

```
= {
 -60.0,
  0,
  -10.0,
}
```

15.70.3.9 double myPOI_p[4]

Initial value:

```
= {
  0.1,
  2.5,
  20.0,
  -60.0
}
```

15.70.3.10 double postExpDNDN[2]

Initial value:

```
={
  8.0,
  -92.0
}
```

15.70.3.11 double postExpKCDN[2]

Initial value:

```
={
  5.0,
  0.0
}
```

15.70.3.12 double postExpLHIKC[2]

Initial value:

```
={
  3.0,
  -92.0
}
```

15.70.3.13 double postExpPNKC[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.70.3.14 double postExpPNLHI[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.70.3.15 double* postSynV = NULL

15.70.3.16 double stdTM_ini[4]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207
}
```

15.70.3.17 double stdTM_p[7]

Initial value:

```
= {
  7.15,
  50.0,
  1.43,
  -95.0,
  0.02672,
  -63.563,
  0.143
}
```

15.71 MBody1.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "sizes.h"
```

Macros

- #define DT 0.1

  This defines the global time step at which the simulation will run.

Functions

- void modelDefinition (NNmodel &model)

  This function defines the MBody1 model, and it is a good example of how networks should be defined.

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [10]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

15.71.1 Macro Definition Documentation

15.71.1.1 #define DT 0.1

This defines the global time step at which the simulation will run.

15.71.2 Function Documentation

15.71.2.1 void modelDefinition (NNmodel & *model*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

15.71.3 Variable Documentation

15.71.3.1 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
}
```

15.71.3.2 double myDNDN_p[2]

Initial value:

```
= {
  -30.0,
  50.0
}
```

15.71.3.3 double myKCDN_ini[2]

Initial value:

```
={
  0.01,
  0.01,
}
```

15.71.3.4 double myKCDN_p[10]

Initial value:

```
= {
  50.0,
  50.0,
  50000.0,
  100000.0,
  200.0,
  0.015,
  0.0075,
  33.33,
  10.0,
  0.00006
}
```

15.71.3.5 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

15.71.3.6 double myLHIKC_p[2]

Initial value:

```
= {
  -40.0,
  50.0
}
```

15.71.3.7 double myPNKC_ini[1]

Initial value:

```
= {
  0.01
}
```

15.71.3.8 double* myPNKC_p = NULL

15.71.3.9 double myPNLHI_ini[1]

Initial value:

```
= {
    0.0
}
```

15.71.3.10 double* myPNLHI_p = NULL

15.71.3.11 double myPOI_ini[3]

Initial value:

```
= {
 -60.0,
  0,
  -10.0
}
```

15.71.3.12 double myPOI_p[4]

Initial value:

```
= {
  0.1,
  2.5,
  20.0,
  -60.0
}
```

15.71.3.13 double postExpDNDN[2]

Initial value:

```
={
  2.5,
  -92.0
}
```

15.71.3.14 double postExpKCDN[2]

Initial value:

```
={
  5.0,
  0.0
}
```

15.71.3.15 double postExpLHIKC[2]

Initial value:

```
={
    1.5,
  -92.0
}
```

15.71.3.16 double postExpPNKC[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.71.3.17 double postExpPNLHI[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.71.3.18 double* postSynV = NULL

15.71.3.19 double stdTM_ini[4]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207
}
```

15.71.3.20 double stdTM_p[7]

Initial value:

```
= {
  7.15,
  50.0,
  1.43,
  -95.0,
  0.02672,
  -63.563,
  0.143
}
```

15.72 MBody_delayedSyn.cc File Reference

This file contains the model definition of the mushroom body "MBody_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim).

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "sizes.h"
```

Macros

- #define DT 0.1

  This defines the global time step at which the simulation will run.

Functions

- void modelDefinition (NNmodel &model)

  This function defines the MBody_delayedSyn model, and it is a good example of how networks should be defined.

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]

- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [10]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

15.72.1 Detailed Description

This file contains the model definition of the mushroom body "MBody_delayedSyn" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim).

15.72.2 Macro Definition Documentation

15.72.2.1 #define DT 0.1

This defines the global time step at which the simulation will run.

15.72.3 Function Documentation

15.72.3.1 void modelDefinition (NNmodel & *model*)

This function defines the MBody_delayedSyn model, and it is a good example of how networks should be defined.

15.72.4 Variable Documentation

15.72.4.1 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
}
```

15.72.4.2 double myDNDN_p[2]

Initial value:

```
= {
  -30.0,
  50.0
}
```

15.72.4.3 double myKCDN_ini[2]

Initial value:

```
={
  0.01,
  0.01,
}
```

15.72.4.4 double myKCDN_p[10]

Initial value:

```
= {
  50.0,
  50.0,
  50000.0,
  100000.0,
  200.0,
  0.015,
  0.0075,
  33.33,
  10.0,
  0.00006
}
```

15.72.4.5 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

15.72.4.6 double myLHIKC_p[2]

Initial value:

```
= {
  -40.0,
  50.0
}
```

15.72.4.7 double myPNKC_ini[1]

Initial value:

```
= {
  0.01
}
```

15.72.4.8 double* myPNKC_p = NULL

15.72.4.9 double myPNLHI_ini[1]

Initial value:

```
= {
    0.0
}
```

15.72.4.10 double* myPNLHI_p = NULL

15.72.4.11 double myPOI_ini[3]

Initial value:

```
= {
 -60.0,
  0,
  -10.0
}
```

15.72.4.12 double myPOI_p[4]

Initial value:

```
= {
  0.1,
  2.5,
  20.0,
  -60.0
}
```

15.72.4.13 double postExpDNDN[2]

Initial value:

```
={
  2.5,
  -92.0
}
```

15.72.4.14 double postExpKCDN[2]

Initial value:

```
={
  5.0,
  0.0
}
```

15.72.4.15 double postExpLHIKC[2]

Initial value:

```
={
    1.5,
  -92.0
}
```

15.72.4.16 double postExpPNKC[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.72.4.17 double postExpPNLHI[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.72.4.18 double* postSynV = NULL

15.72.4.19 double stdTM_ini[4]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207
}
```

15.72.4.20 double stdTM_p[7]

Initial value:

```
= {
  7.15,
  50.0,
  1.43,
  -95.0,
  0.02672,
  -63.563,
  0.143
}
```

15.73 MBody_individualID.cc File Reference

This file contains the model definition of the mushroom body "MBody_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC.

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "sizes.h"
```

Macros

- #define DT 0.1

  This defines the global time step at which the simulation will run.

Functions

- void modelDefinition (NNmodel &model)

  This function defines the MBody1 model, and it is a good example of how networks should be defined.

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [11]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL

15.73.1 Detailed Description

This file contains the model definition of the mushroom body "MBody_incividualID" model. It is used in both the GeNN code generation and the user side simulation code (class classol, file classol_sim). It uses INDIVIDUALID for the connections from AL to MB allowing quite large numbers of PN and KC.

15.73.2 Macro Definition Documentation

15.73.2.1 #define DT 0.1

This defines the global time step at which the simulation will run.

15.73.3 Function Documentation

15.73.3.1 void modelDefinition (**NNmodel** & *model*)

This function defines the MBody1 model, and it is a good example of how networks should be defined.

15.73.4 Variable Documentation

15.73.4.1 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
}
```

15.73.4.2 double myDNDN_p[2]

Initial value:

```
= {
  -30.0,
  50.0
}
```

15.73.4.3 double myKCDN_ini[2]

Initial value:

```
={
  0.01,
  0.01,
}
```

15.73.4.4 double myKCDN_p[11]

Initial value:

```
= {
    50.0,
    50.0,
  50000.0,
  100000.0,
  200.0,
  0.015,
  0.0075,
  33.33,
  10.0,
  0.00006
}
```

15.73.4.5 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

15.73.4.6 double myLHIKC_p[2]

Initial value:

```
= {
  -40.0,
  50.0
}
```

15.73.4.7 double myPNKC_ini[1]

Initial value:

```
= {
  gPNKC_GLOBAL
}
```

15.73.4.8 double* myPNKC_p = NULL

15.73.4.9 double myPNLHI_ini[1]

Initial value:

```
= {
    0.0
}
```

15.73.4.10 double* myPNLHI_p = NULL

15.73.4.11 double myPOI_ini[3]

Initial value:

```
= {
 -60.0,
  0,
  -10.0
}
```

15.73.4.12 double myPOI_p[4]

Initial value:

```
= {
  0.1,
  2.5,
  20.0,
  -60.0
}
```

15.73.4.13 double postExpDNDN[2]

Initial value:

```
={
  2.5,
  -92.0
}
```

15.73.4.14 double postExpKCDN[2]

Initial value:

```
={
  5.0,
  0.0
}
```

15.73.4.15 double postExpLHIKC[2]

Initial value:

```
={
    1.5,
  -92.0
}
```

15.73.4.16 double postExpPNKC[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.73.4.17 double postExpPNLHI[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.73.4.18 double* postSynV = NULL

15.73.4.19 double stdTM_ini[4]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207
}
```

15.73.4.20 double stdTM_p[7]

Initial value:

```
= {
  7.15,
  50.0,
  1.43,
  -95.0,
  0.02672,
  -63.563,
  0.143
}
```

15.74 MBody_userdef.cc File Reference

This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol_sim).

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "sizes.h"
```

Classes

- class pwSTDP_userdef

  TODO This class definition may be code-generated in a future release.

Macros

- #define DT 0.1

  This defines the global time step at which the simulation will run.
- #define TIMING

Functions

- void modelDefinition (NNmodel &model)

  This function defines the MBody1 model with user defined synapses.

Variables

- double myPOI_p [4]
- double myPOI_ini [3]
- double stdTM_p [7]
- double stdTM_ini [4]
- double * myPNKC_p = NULL
- double myPNKC_ini [1]
- double postExpPNKC [2]
- double * myPNLHI_p = NULL
- double myPNLHI_ini [1]
- double postExpPNLHI [2]
- double myLHIKC_p [2]
- double myLHIKC_ini [1]
- double postExpLHIKC [2]
- double myKCDN_p [11]
- double myKCDN_ini [2]
- double postExpKCDN [2]
- double myDNDN_p [2]
- double myDNDN_ini [1]
- double postExpDNDN [2]
- double * postSynV = NULL
- double postSynV_EXPDECAY_EVAR [1]
- scalar * gpPNKC = new scalar[_NAL*_NMB]
- scalar * gpKCDN = new scalar[_NMB*_NLB]

15.74.1 Detailed Description

This file contains the model definition of the mushroom body model. tis used in the GeNN code generation and the user side simulation code (class classol, file classol_sim).

15.74.2 Macro Definition Documentation

15.74.2.1 #define DT 0.1

This defines the global time step at which the simulation will run.

15.74.2.2 #define TIMING

15.74.3 Function Documentation

15.74.3.1 void modelDefinition (**NNmodel** & *model*)

This function defines the MBody1 model with user defined synapses.

15.74.4 Variable Documentation

15.74.4.1 **scalar*** gpKCDN = new **scalar**[_NMB*_NLB]

15.74.4.2 **scalar*** gpPNKC = new **scalar**[_NAL*_NMB]

15.74.4.3 double myDNDN_ini[1]

Initial value:

```
={
    5.0/_NLB
}
```

15.74.4.4 double myDNDN_p[2]

Initial value:

```
= {
  -30.0,
  50.0
}
```

15.74.4.5 double myKCDN_ini[2]

Initial value:

```
={
  0.01,
  0.01,
}
```

15.74.4.6 double myKCDN_p[11]

Initial value:

```
= {
  -20.0,
  50.0,
  50.0,
  50000.0,
  100000.0,
```

```
  200.0,
  0.015,
  0.0075,
  33.33,
  10.0,
  0.00006
}
```

15.74.4.7 double myLHIKC_ini[1]

Initial value:

```
= {
    1.0/_NLHI
}
```

15.74.4.8 double myLHIKC_p[2]

Initial value:

```
= {
  -40.0,
  50.0
}
```

15.74.4.9 double myPNKC_ini[1]

Initial value:

```
= {
  0.01
}
```

15.74.4.10 double* myPNKC_p = NULL

15.74.4.11 double myPNLHI_ini[1]

Initial value:

```
= {
    0.0
}
```

15.74.4.12 double* myPNLHI_p = NULL

15.74.4.13 double myPOI_ini[3]

Initial value:

```
= {
 -60.0,
  0,
  -10.0
}
```

15.74.4.14 double myPOI_p[4]

Initial value:

```
= {
  0.1,
  2.5,
  20.0,
  -60.0
}
```

15.74.4.15 double postExpDNDN[2]

Initial value:

```
={
  2.5,
  -92.0
}
```

15.74.4.16 double postExpKCDN[2]

Initial value:

```
={
  5.0,
  0.0
}
```

15.74.4.17 double postExpLHIKC[2]

Initial value:

```
={
  1.5,
  -92.0
}
```

15.74.4.18 double postExpPNKC[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.74.4.19 double postExpPNLHI[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.74.4.20 double* postSynV = NULL

15.74.4.21 double postSynV_EXPDECAY_EVAR[1]

Initial value:

```
= {
0
}
```

15.74.4.22 double stdTM_ini[4]

Initial value:

```
= {
  -60.0,
  0.0529324,
  0.3176767,
  0.5961207
}
```

15.74.4.23 double stdTM_p[7]

Initial value:

```
= {
  7.15,
  50.0,
  1.43,
  -95.0,
  0.02672,
  -63.563,
  0.143
}
```

15.75 Model_Schmuker_2014_classifier.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 0.5
- #define NUM_VR 10
- #define NUM_FEATURES 4
- #define NUM_CLASSES 3
- #define NETWORK_SCALE 10
- #define CLUST_SIZE_AN (NETWORK_SCALE ∗ 6)
- #define CLUST_SIZE_PN (NETWORK_SCALE ∗ 6)
- #define CLUST_SIZE_RN (NETWORK_SCALE ∗ 6)
- #define SYNAPSE_TAU_RNPN 1.0
- #define SYNAPSE_TAU_PNPN 5.5
- #define SYNAPSE_TAU_PNAN 1.0
- #define SYNAPSE_TAU_ANAN 8.0

Functions

- void modelDefinition (NNmodel &model)

15.75.1 Macro Definition Documentation

15.75.1.1 #define CLUST_SIZE_AN (NETWORK_SCALE ∗ 6)

15.75.1.2 #define CLUST_SIZE_PN (NETWORK_SCALE ∗ 6)

15.75.1.3 #define CLUST_SIZE_RN (NETWORK_SCALE ∗ 6)

15.75.1.4 #define DT 0.5

15.75.1.5 #define NETWORK_SCALE 10

15.75.1.6 #define NUM_CLASSES 3

15.75.1.7 #define NUM_FEATURES 4

15.75.1.8 #define NUM_VR 10

15.75.1.9 #define SYNAPSE_TAU_ANAN 8.0

15.75.1.10 #define SYNAPSE_TAU_PNAN 1.0

15.75.1.11 #define SYNAPSE_TAU_PNPN 5.5

15.75.1.12 #define SYNAPSE_TAU_RNPN 1.0

15.75.2 Function Documentation

15.75.2.1 void modelDefinition (NNmodel & *model*)

15.76 modelSpec.cc File Reference

```
#include "utils.h"
```

Macros

- #define _MODELSPEC_CC_

  macro for avoiding multiple inclusion during compilation

Functions

- void initGeNN ()

  Method for GeNN initialisation (by preparing standard models)

15.76.1 Macro Definition Documentation

15.76.1.1 #define _MODELSPEC_CC_

macro for avoiding multiple inclusion during compilation

15.76.2 Function Documentation

15.76.2.1 void initGeNN ( )

Method for GeNN initialisation (by preparing standard models)

15.77 modelSpec.h File Reference

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections.

```
#include <vector>
#include "global.h"
```

Classes

- class dpclass
- struct neuronModel

  class (struct) for specifying a neuron model.
- struct postSynModel

Structure to hold the information that defines a post-synaptic model (a model of how synapses affect post-synaptic neuron variables, classically in the form of a synaptic current). It also allows to define an equation for the dynamics that can be applied to the summed synaptic input variable "insyn".

- class weightUpdateModel

  Structure to hold the information that defines a weightupdate model (a model of how spikes affect synaptic (and/or) (mostly) post-synaptic neuron variables. It also allows to define changes in response to post-synaptic spikes/spike-like events.

- class NNmodel

  Structure to hold the information that defines synapse dynamics (a model of how synapse variables change over time, independent of or in addition to changes when spikes occur).

Macros

- #define _MODELSPEC_H_

  macro for avoiding multiple inclusion during compilation

- #define MAXNRN 7
- #define SYNTYPENO 4
- #define NOINP 0

  Macro attaching the name NOINP (no input) to 0.

- #define CONSTINP 1

  Macro attaching the name CONSTINP (constant input) to 1.

- #define MATINP 2

  Macro attaching the name MATINP (explicit input defined as a matrix) to 2.

- #define INPRULE 3

  Macro attaching the name INPRULE (explicit dynamic input defined as a rule) to 3.

- #define RANDNINP 4

  Macro attaching the name RANDNINP (Random input with Gaussian distribution, calculated real time on the device by the generated code) to 4 (TODO, not implemented yet)

- #define ALLTOALL 0

  Macro attaching the label "ALLTOALL" to connectivity type 0.

- #define DENSE 1

  Macro attaching the label "DENSE" to connectivity type 1.

- #define SPARSE 2

  Macro attaching the label "SPARSE" to connectivity type 2.

- #define INDIVIDUALG 0

  Macro attaching the label "INDIVIDUALG" to method 0 for the definition of synaptic conductances.

- #define GLOBALG 1

  Macro attaching the label "GLOBALG" to method 1 for the definition of synaptic conductances.

- #define INDIVIDUALID 2

  Macro attaching the label "INDIVIDUALID" to method 2 for the definition of synaptic conductances.

- #define NO_DELAY 0

  Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

- #define NOLEARNING 0

  Macro attaching the label "NOLEARNING" to flag 0.

- #define LEARNING 1

  Macro attaching the label "LEARNING" to flag 1.

- #define EXITSYN 0

  Macro attaching the label "EXITSYN" to flag 0 (excitatory synapse)

- #define INHIBSYN 1

  Macro attaching the label "INHIBSYN" to flag 1 (inhibitory synapse)

- #define TRUE 1

  Macro attaching the label "TRUE" to value 1.

- #define FALSE 0

  Macro attaching the label "FALSE" to value 0.
- #define CPU 0

  Macro attaching the label "CPU" to flag 0.
- #define GPU 1

  Macro attaching the label "GPU" to flag 1.
- #define GENN_FLOAT 0

  Macro attaching the label "GENN_FLOAT" to flag 0. Used by NNModel::setPrecision()
- #define GENN_DOUBLE 1

  Macro attaching the label "GENN_DOUBLE" to flag 1. Used by NNModel::setPrecision()
- #define AUTODEVICE -1

  Macro attaching the label AUTODEVICE to flag -1. Used by setGPUDevice.
- #define SPK_THRESH_STDP 0.0f

  Macro defining the spiking threshold for the purposes of STDP.
- #define MAXPOSTSYN 2

Functions

- void initGeNN ()

  Method for GeNN initialisation (by preparing standard models)

Variables

- unsigned int GeNNReady = 0
- unsigned int MAPNEURON

  variable attaching the name "MAPNEURON"
- unsigned int POISSONNEURON

  variable attaching the name "POISSONNEURON"
- unsigned int TRAUBMILES_FAST

  variable attaching the name "TRAUBMILES_FAST"
- unsigned int TRAUBMILES_ALTERNATIVE

  variable attaching the name "TRAUBMILES_ALTERNATIVE"
- unsigned int TRAUBMILES_SAFE

  variable attaching the name "TRAUBMILES_SAFE"
- unsigned int TRAUBMILES

  variable attaching the name "TRAUBMILES"
- unsigned int TRAUBMILES_PSTEP

  variable attaching the name "TRAUBMILES_PSTEP"
- unsigned int IZHIKEVICH

  variable attaching the name "IZHIKEVICH"
- unsigned int IZHIKEVICH_V

  variable attaching the name "IZHIKEVICH_V"
- unsigned int SPIKESOURCE

  variable attaching the name "SPIKESOURCE"
- unsigned int NSYNAPSE

  Variable attaching the name NSYNAPSE to predefined synapse type 0, which is a non-learning synapse.
- unsigned int NGRADSYNAPSE

  Variable attaching the name NGRADSYNAPSE to predefined synapse type 1 which is a graded synapse wrt the presynaptic voltage.
- unsigned int LEARN1SYNAPSE

  Variable attaching the name LEARN1SYNAPSE to the predefined synapse type 2 which is a learning using spike timing; uses a primitive STDP rule for learning.
- unsigned int EXPDECAY
- unsigned int IZHIKEVICH_PS

15.77.1 Detailed Description

Header file that contains the class (struct) definition of neuronModel for defining a neuron model and the class definition of NNmodel for defining a neuronal network model. Part of the code generation and generated code sections.

15.77.2 Macro Definition Documentation

15.77.2.1 #define _MODELSPEC_H_

macro for avoiding multiple inclusion during compilation

15.77.2.2 #define ALLTOALL 0

Macro attaching the label "ALLTOALL" to connectivity type 0.

15.77.2.3 #define AUTODEVICE -1

Macro attaching the label AUTODEVICE to flag -1. Used by setGPUDevice.

15.77.2.4 #define CONSTINP 1

Macro attaching the name CONSTINP (constant input) to 1.

15.77.2.5 #define CPU 0

Macro attaching the label "CPU" to flag 0.

15.77.2.6 #define DENSE 1

Macro attaching the label "DENSE" to connectivity type 1.

15.77.2.7 #define EXITSYN 0

Macro attaching the label "EXITSYN" to flag 0 (excitatory synapse)

15.77.2.8 #define FALSE 0

Macro attaching the label "FALSE" to value 0.

15.77.2.9 #define GENN_DOUBLE 1

Macro attaching the label "GENN_DOUBLE" to flag 1. Used by NNModel::setPrecision()

15.77.2.10 #define GENN_FLOAT 0

Macro attaching the label "GENN_FLOAT" to flag 0. Used by NNModel::setPrecision()

15.77.2.11 #define GLOBALG 1

Macro attaching the label "GLOBALG" to method 1 for the definition of synaptic conductances.

15.77.2.12 #define GPU 1

Macro attaching the label "GPU" to flag 1.

15.77.2.13 #define INDIVIDUALG 0

Macro attaching the label "INDIVIDUALG" to method 0 for the definition of synaptic conductances.

15.77.2.14 #define INDIVIDUALID 2

Macro attaching the label "INDIVIDUALID" to method 2 for the definition of synaptic conductances.

15.77.2.15 #define INHIBSYN 1

Macro attaching the label "INHIBSYN" to flag 1 (inhibitory synapse)

15.77.2.16 #define INPRULE 3

Macro attaching the name INPRULE (explicit dynamic input defined as a rule) to 3.

15.77.2.17 #define LEARNING 1

Macro attaching the label "LEARNING" to flag 1.

15.77.2.18 #define MATINP 2

Macro attaching the name MATINP (explicit input defined as a matrix) to 2.

15.77.2.19 #define MAXNRN 7

15.77.2.20 #define MAXPOSTSYN 2

15.77.2.21 #define NO_DELAY 0

Macro used to indicate no synapse delay for the group (only one queue slot will be generated)

15.77.2.22 #define NOINP 0

Macro attaching the name NOINP (no input) to 0.

15.77.2.23 #define NOLEARNING 0

Macro attaching the label "NOLEARNING" to flag 0.

15.77.2.24 #define RANDNINP 4

Macro attaching the name RANDNINP (Random input with Gaussian distribution, calculated real time on the device by the generated code) to 4 (TODO, not implemented yet)

15.77.2.25 #define SPARSE 2

Macro attaching the label "SPARSE" to connectivity type 2.

15.77.2.26 #define SPK_THRESH_STDP 0.0f

Macro defining the spiking threshold for the purposes of STDP.

15.77.2.27 #define SYNTYPENO 4

15.77.2.28 #define TRUE 1

Macro attaching the label "TRUE" to value 1.

15.77.3 Function Documentation

15.77.3.1 void initGeNN ( )

Method for GeNN initialisation (by preparing standard models)

15.77.4 Variable Documentation

15.77.4.1 unsigned int EXPDECAY

15.77.4.2 unsigned int GeNNReady = 0

15.77.4.3 unsigned int IZHIKEVICH

variable attaching the name "IZHIKEVICH"

15.77.4.4 unsigned int IZHIKEVICH_PS

15.77.4.5 unsigned int IZHIKEVICH_V

variable attaching the name "IZHIKEVICH_V"

15.77.4.6 unsigned int LEARN1SYNAPSE

Variable attaching the name LEARN1SYNAPSE to the predefined synapse type 2 which is a learning using spike timing; uses a primitive STDP rule for learning.

15.77.4.7 unsigned int MAPNEURON

variable attaching the name "MAPNEURON"

15.77.4.8 unsigned int NGRADSYNAPSE

Variable attaching the name NGRADSYNAPSE to predefined synapse type 1 which is a graded synapse wrt the presynaptic voltage.

15.77.4.9 unsigned int NSYNAPSE

Variable attaching the name NSYNAPSE to predefined synapse type 0, which is a non-learning synapse.

15.77.4.10 unsigned int POISSONNEURON

variable attaching the name "POISSONNEURON"

15.77.4.11 unsigned int SPIKESOURCE

variable attaching the name "SPIKESOURCE"

15.77.4.12 unsigned int TRAUBMILES

variable attaching the name "TRAUBMILES"

15.77.4.13 unsigned int TRAUBMILES_ALTERNATIVE

variable attaching the name "TRAUBMILES_ALTERNATIVE"

15.77.4.14 unsigned int TRAUBMILES_FAST

variable attaching the name "TRAUBMILES_FAST"

15.77.4.15 unsigned int TRAUBMILES_PSTEP

variable attaching the name "TRAUBMILES_PSTEP"

15.77.4.16 unsigned int TRAUBMILES_SAFE

variable attaching the name "TRAUBMILES_SAFE"

15.78 OneComp.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "../../userproject/include/sizes.h"
```

Macros

- #define DT 1.0

Functions

- void modelDefinition (NNmodel &model)

Variables

- double exIzh_p [4]
- double exIzh_ini [2]
- double mySyn_p [3]
- double postExp [2]
- double * postSynV = NULL
- float inpIzh1 = 4.0

15.78.1 Macro Definition Documentation

15.78.1.1 #define DT 1.0

15.78.2 Function Documentation

15.78.2.1 void modelDefinition (NNmodel & *model*)

15.78.3 Variable Documentation

15.78.3.1 double exIzh_ini[2]

Initial value:

```
={

        -65,
        -20
}
```

15.78.3.2 double exIzh_p[4]

Initial value:

```
={

        0.02,
        0.2,
        -65,
        6
}
```

15.78.3.3 float inplzh1 = 4.0

15.78.3.4 double mySyn_p[3]

Initial value:

```
= {
  0.0,
  -20.0,
  1.0
}
```

15.78.3.5 double postExp[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.78.3.6 double* postSynV = NULL

15.79 OneComp_model.cc File Reference

```
#include "OneComp_CODE/runner.cc"
```

Macros

- #define _ONECOMP_MODEL_CC_

15.79.1 Macro Definition Documentation

15.79.1.1 #define _ONECOMP_MODEL_CC_

15.80 OneComp_model.h File Reference

```
#include "OneComp.cc"
```

Classes

- class neuronpop

15.81 OneComp_sim.cu File Reference

```
#include "OneComp_sim.h"
```

Functions

- int main (int argc, char *argv[])

15.81.1 Function Documentation

15.81.1.1 int main (int *argc*, char * *argv[]*)

15.82 OneComp_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "OneComp_model.h"
#include "OneComp_model.cc"
```

Macros

- #define DBG_SIZE 10000
- #define T_REPORT_TME 100.0
- #define TOTAL_TME 5000

Variables

- float t = 0.0f
- unsigned int iT = 0
- CStopWatch timer

15.82.1 Macro Definition Documentation

15.82.1.1 #define DBG_SIZE 10000

15.82.1.2 #define T_REPORT_TME 100.0

15.82.1.3 #define TOTAL_TME 5000

15.82.2 Variable Documentation

15.82.2.1 unsigned int iT = 0

15.82.2.2 float t = 0.0f

15.82.2.3 CStopWatch timer

15.83 PoissonIzh-model.cc File Reference

```
#include "PoissonIzh-model.h"
#include "PoissonIzh_CODE/runner.cc"
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define _POISSONIZHMODEL_CC_

15.83.1 Macro Definition Documentation

15.83.1.1 #define _POISSONIZHMODEL_CC_

15.84 PoissonIzh-model.h File Reference

Classes

- class classol

  This class cpontains the methods for running the MBody1 example model.

15.85 PoissonIzh.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
#include "../../userproject/include/sizes.h"
```

Macros

- #define DT 1.0
- #define _FTYPE GENN_FLOAT
- #define scalar float
- #define SCALAR_MIN (float)FLT_MIN
- #define SCALAR_MAX (float)FLT_MAX

Functions

- void modelDefinition (NNmodel &model)

Variables

- double myPOI_p [4]
- double myPOI_ini [4]
- double exIzh_p [4]
- double exIzh_ini [2]
- double mySyn_p [3]
- double mySyn_ini [1]
- double postExp [2]
- double * postSynV = NULL

15.85.1 Macro Definition Documentation

15.85.1.1 #define _FTYPE GENN_FLOAT

15.85.1.2 #define DT 1.0

15.85.1.3 #define scalar float

15.85.1.4 #define SCALAR_MAX (float)FLT_MAX

15.85.1.5 #define SCALAR_MIN (float)FLT_MIN

15.85.2 Function Documentation

15.85.2.1 void modelDefinition (NNmodel & *model*)

15.85.3 Variable Documentation

15.85.3.1 double exIzh_ini[2]

Initial value:

```
={

        -65,
        -20
}
```

15.85.3.2 double exIzh_p[4]

Initial value:

```
={

        0.02,
        0.2,
        -65,
        6
}
```

15.85.3.3 double myPOI_ini[4]

Initial value:

```
= {
 -60.0,
  0,
  -10.0
}
```

15.85.3.4 double myPOI_p[4]

Initial value:

```
= {

  1,
  2.5,
  20.0,
  -60.0
}
```

15.85.3.5 double mySyn_ini[1]

Initial value:

```
={
  0.0
}
```

15.85.3.6 double mySyn_p[3]

Initial value:

```
= {
  0.0,
  -20.0,
  1.0
}
```

15.85.3.7 double postExp[2]

Initial value:

```
={
  1.0,
  0.0
}
```

15.85.3.8 double* postSynV = NULL

15.86 PoissonIzh_sim.cu File Reference

```
#include "PoissonIzh_sim.h"
```

Functions

- int main (int argc, char *argv[])

15.86.1 Function Documentation

15.86.1.1 int main (int *argc,* char * *argv[]*)

15.87 PoissonIzh_sim.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "PoissonIzh.cc"
#include "PoissonIzh-model.h"
#include "PoissonIzh-model.cc"
```

Macros

- #define MYRAND(Y, X) Y = Y * 1103515245 +12345; X= (Y >> 16);
- #define INJECTCURRENT 0
- #define DBG_SIZE 1000
- #define PATTERNNO 100
- #define T_REPORT_TME 1000.0
- #define SYN_OUT_TME 2000.0
- #define TOTAL_TME 5000

Variables

- float t = 0.0f
- unsigned int iT = 0
- scalar InputBaseRate = 2e-02
- CStopWatch timer

15.87.1 Macro Definition Documentation

15.87.1.1 #define DBG_SIZE 1000

15.87.1.2 #define INJECTCURRENT 0

15.87.1.3 #define MYRAND(*Y*, *X*) Y = Y * 1103515245 +12345; X= (Y >> 16);

15.87.1.4 #define PATTERNNO 100

15.87.1.5 #define SYN_OUT_TME 2000.0

15.87.1.6 #define T_REPORT_TME 1000.0

15.87.1.7 #define TOTAL_TME 5000

15.87.2 Variable Documentation

15.87.2.1 scalar InputBaseRate = 2e-02

15.87.2.2 unsigned int iT = 0

15.87.2.3 float t = 0.0f

15.87.2.4 CStopWatch timer

15.88 postVarsInPostLearn.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double neuron_p [1]
- double neuron_p2 [1]
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.88.1 Macro Definition Documentation

15.88.1.1 #define DT 1.0f

15.88.2 Function Documentation

15.88.2.1 void modelDefinition (NNmodel & *model*)

15.88.3 Variable Documentation

15.88.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.88.3.2 double neuron_p[1]

Initial value:

```
={
    1.0
}
```

15.88.3.3 double neuron_p2[1]

Initial value:

```
={
    2.0
}
```

15.88.3.4 double* postSyn_ini = NULL

15.88.3.5 double* postSyn_p = NULL

15.88.3.6 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.88.3.7 double* synapses_p = NULL

15.89 postVarsInPostLearn_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double neuron_p [1]
- double neuron_p2 [1]
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.89.1 Macro Definition Documentation

15.89.1.1 #define DT 1.0f

15.89.2 Function Documentation

15.89.2.1 void modelDefinition (NNmodel & *model*)

15.89.3 Variable Documentation

15.89.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.89.3.2 double neuron_p[1]

Initial value:

```
={
    1.0
}
```

15.89.3.3 double neuron_p2[1]

Initial value:

```
={
    2.0
}
```

15.89.3.4 double* postSyn_ini = NULL

15.89.3.5 double* postSyn_p = NULL

15.89.3.6 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.89.3.7 double* synapses_p = NULL

15.90 postVarsInSimCode.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.90.1 Macro Definition Documentation

15.90.1.1 #define DT 1.0f

15.90.2 Function Documentation

15.90.2.1 void modelDefinition (NNmodel & *model*)

15.90.3 Variable Documentation

15.90.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.90.3.2 double* neuron_p = NULL

15.90.3.3 double* postSyn_ini = NULL

15.90.3.4 double* postSyn_p = NULL

15.90.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.90.3.6 double* synapses_p = NULL

15.91 postVarsInSimCode_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.91.1 Macro Definition Documentation

15.91.1.1 #define DT 1.0f

15.91.2 Function Documentation

15.91.2.1 void modelDefinition (NNmodel & *model*)

15.91.3 Variable Documentation

15.91.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.91.3.2 double* neuron_p = NULL

15.91.3.3 double* postSyn_ini = NULL

15.91.3.4 double* postSyn_p = NULL

15.91.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.91.3.6 double* synapses_p = NULL

15.92 postVarsInSynapseDynamics.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.92.1 Macro Definition Documentation

15.92.1.1 #define DT 1.0f

15.92.2 Function Documentation

15.92.2.1 void modelDefinition (NNmodel & *model*)

15.92.3 Variable Documentation

15.92.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.92.3.2 double* neuron_p = NULL

15.92.3.3 double* postSyn_ini = NULL

15.92.3.4 double* postSyn_p = NULL

15.92.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.92.3.6 double* synapses_p = NULL

15.93 postVarsInSynapseDynamics_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.93.1 Macro Definition Documentation

15.93.1.1 #define DT 1.0f

15.93.2 Function Documentation

15.93.2.1 void modelDefinition (NNmodel & *model*)

15.93.3 Variable Documentation

15.93.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.93.3.2 double* neuron_p = NULL

15.93.3.3 double* postSyn_ini = NULL

15.93.3.4 double* postSyn_p = NULL

15.93.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.93.3.6 double* synapses_p = NULL

15.94 preVarsInPostLearn.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double neuron_p [1]
- double neuron_p2 [1]
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.94.1 Macro Definition Documentation

15.94.1.1 #define DT 1.0f

15.94.2 Function Documentation

15.94.2.1 void modelDefinition (NNmodel & *model*)

15.94.3 Variable Documentation

15.94.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.94.3.2 double neuron_p[1]

Initial value:

```
={
    1.0
}
```

15.94.3.3 double neuron_p2[1]

Initial value:

```
={
    2.0
}
```

15.94.3.4 double* postSyn_ini = NULL

15.94.3.5 double* postSyn_p = NULL

15.94.3.6 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.94.3.7 double* synapses_p = NULL

15.95 preVarsInPostLearn_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double neuron_p [1]
- double neuron_p2 [1]
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.95.1 Macro Definition Documentation

15.95.1.1 #define DT 1.0f

15.95.2 Function Documentation

15.95.2.1 void modelDefinition (NNmodel & *model*)

15.95.3 Variable Documentation

15.95.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.95.3.2 double neuron_p[1]

Initial value:

```
={
    1.0
}
```

15.95.3.3 double neuron_p2[1]

Initial value:

```
={
    2.0
}
```

15.95.3.4 double* postSyn_ini = NULL

15.95.3.5 double* postSyn_p = NULL

15.95.3.6 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.95.3.7 double* synapses_p = NULL

15.96 preVarsInSimCode.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.96.1 Macro Definition Documentation

15.96.1.1 #define DT 1.0f

15.96.2 Function Documentation

15.96.2.1 void modelDefinition (NNmodel & *model*)

15.96.3 Variable Documentation

15.96.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.96.3.2 double* neuron_p = NULL

15.96.3.3 double* postSyn_ini = NULL

15.96.3.4 double* postSyn_p = NULL

15.96.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.96.3.6 double* synapses_p = NULL

15.97 preVarsInSimCode_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.97.1 Macro Definition Documentation

15.97.1.1 #define DT 1.0f

15.97.2 Function Documentation

15.97.2.1 void modelDefinition (NNmodel & *model*)

15.97.3 Variable Documentation

15.97.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.97.3.2 double* neuron_p = NULL

15.97.3.3 double* postSyn_ini = NULL

15.97.3.4 double* postSyn_p = NULL

15.97.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.97.3.6 double* synapses_p = NULL

15.98 preVarsInSynapseDynamics.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.98.1 Macro Definition Documentation

15.98.1.1 #define DT 1.0f

15.98.2 Function Documentation

15.98.2.1 void modelDefinition (NNmodel & *model*)

15.98.3 Variable Documentation

15.98.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.98.3.2 double* neuron_p = NULL

15.98.3.3 double* postSyn_ini = NULL

15.98.3.4 double* postSyn_p = NULL

15.98.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.98.3.6 double* synapses_p = NULL

15.99 preVarsInSynapseDynamics_sparse.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double * neuron_p = NULL
- double neuron_ini [2]
- double * synapses_p = NULL
- double synapses_ini [1]
- double * postSyn_p = NULL
- double * postSyn_ini = NULL

15.99.1 Macro Definition Documentation

15.99.1.1 #define DT 1.0f

15.99.2 Function Documentation

15.99.2.1 void modelDefinition (NNmodel & *model*)

15.99.3 Variable Documentation

15.99.3.1 double neuron_ini[2]

Initial value:

```
= {
    0.0,
    0.0
}
```

15.99.3.2 double* neuron_p = NULL

15.99.3.3 double* postSyn_ini = NULL

15.99.3.4 double* postSyn_p = NULL

15.99.3.5 double synapses_ini[1]

Initial value:

```
= {
    0.0
}
```

15.99.3.6 double* synapses_p = NULL

15.100 randomGen.cc File Reference

Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand().

```
#include "randomGen.h"
```

Macros

- #define RANDOMGEN_CC

  macro for avoiding multiple inclusion during compilation

15.100.1 Detailed Description

Contains the implementation of the ISAAC random number generator class for uniformly distributed random numbers and for a standard random number generator based on the C function rand().

15.100.2 Macro Definition Documentation

15.100.2.1 #define RANDOMGEN_CC

macro for avoiding multiple inclusion during compilation

15.101 randomGen.h File Reference

header file containing the class definition for a uniform random generator based on the ISAAC random number generator

```
#include <time.h>
#include <limits.h>
#include <stdlib.h>
#include "isaac.hpp"
#include <assert.h>
```

Classes

- class randomGen

  Class randomGen which implements the ISAAC random number generator for uniformely distributed random numbers.
- class stdRG

Macros

- #define RANDOMGEN_H

  macro for avoiding multiple inclusion during compilation

15.101.1 Detailed Description

header file containing the class definition for a uniform random generator based on the ISAAC random number generator

15.101.2 Macro Definition Documentation

15.101.2.1 #define RANDOMGEN_H

macro for avoiding multiple inclusion during compilation

15.102 Schmuker2014_classifier.cu File Reference

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <time.h>
#include "Schmuker2014_classifier.h"
#include "Schmuker_2014_classifier_CODE/runner.cc"
#include "sparseUtils.cc"
```

Macros

- #define _SCHMUKER2014_CLASSIFIER_

  macro for avoiding multiple inclusion during compilation

15.102.1 Macro Definition Documentation

15.102.1.1 #define _SCHMUKER2014_CLASSIFIER_

macro for avoiding multiple inclusion during compilation

15.103 Schmuker2014_classifier.h File Reference

Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014".

```
#include "Model_Schmuker_2014_classifier.cc"
```

Classes

- class Schmuker2014_classifier

  This class cpontains the methods for running the Schmuker_2014_classifier example model.

15.103.1 Detailed Description

Header file containing the class definition for the Schmuker2014 classifier, which contains the methods for setting up, initialising, simulating and saving results of a multivariate classifier imspired by the insect olfactory system. See "A neuromorphic network for generic multivariate data classification, Michael Schmuker, Thomas Pfeilc, and Martin Paul Nawrota, 2014".

15.104 simpleBit.h File Reference

Contains three macros that allow simple bit manipulations on an (presumably unsigned) 32 bit integer.

```
#include <cassert>
#include <cmath>
```

Macros

- #define SIMPLEBIT_H

  macro for avoiding multiple inclusion during compilation
- #define B(x, i) ((x) & (0x80000000 >> (i)))

  Extract the bit at the specified position i from x.
- #define setB(x, i) x= ((x) | (0x80000000 >> (i)))

  Set the bit at the specified position i in x to 1.
- #define delB(x, i) x= ((x) & (~(0x80000000 >> (i))))

  Set the bit at the specified position i in x to 0.

15.104.1 Detailed Description

Contains three macros that allow simple bit manipulations on an (presumably unsigned) 32 bit integer.

15.104.2 Macro Definition Documentation

15.104.2.1 #define B(x, i) ((x) & (0x80000000 >> (i)))

Extract the bit at the specified position i from x.

15.104.2.2 #define delB(x, i) x= ((x) & (~(0x80000000 >> (i))))

Set the bit at the specified position i in x to 0.

15.104.2.3 #define setB(x, i) x= ((x) | (0x80000000 >> (i)))

Set the bit at the specified position i in x to 1.

15.104.2.4 #define SIMPLEBIT_H

macro for avoiding multiple inclusion during compilation

15.105 sizes.h File Reference

Macros

- #define _NC1 1

15.105.1 Macro Definition Documentation

15.105.1.1 #define _NC1 1

15.106 sparseUtils.cc File Reference

```
#include <cstdio>
#include <cmath>
```

Macros

- #define sparse_utils_cc

Functions

- template<class DATATYPE >
  unsigned int countEntriesAbove (DATATYPE *Array, int sz, double includeAbove)
- template<class DATATYPE >
  DATATYPE getG (DATATYPE *wuvar, SparseProjection *sparseStruct, int x, int y)
- template<class DATATYPE >
  float getSparseVar (DATATYPE *wuvar, SparseProjection *sparseStruct, int x, int y)
- template<class DATATYPE >
  void setSparseConnectivityFromDense (DATATYPE *wuvar, int preN, int postN, DATATYPE *tmp_gRNPN, SparseProjection *sparseStruct)
- template<class DATATYPE >
  void createSparseConnectivityFromDense (DATATYPE *wuvar, int preN, int postN, DATATYPE *tmp_gR↩NPN, SparseProjection *sparseStruct, bool runTest)
- void createPosttoPreArray (unsigned int preN, unsigned int postN, SparseProjection *sparseStruct)
- void strsearch (string &s, const string trg)

  !!!find var to check if a string is used in a code (atm it is used to create post-to-pre arrays)

15.106.1 Macro Definition Documentation

15.106.1.1 #define sparse_utils_cc

15.106.2 Function Documentation

15.106.2.1 template<class DATATYPE > unsigned int countEntriesAbove (DATATYPE * *Array,* int *sz,* double *includeAbove*)

15.106.2.2 void createPosttoPreArray (unsigned int *preN,* unsigned int *postN,* SparseProjection * *sparseStruct*)

15.106.2.3 template<class DATATYPE > void createSparseConnectivityFromDense (DATATYPE * *wuvar,* int *preN,* int *postN,* DATATYPE * *tmp_gRNPN,* SparseProjection * *sparseStruct,* bool *runTest*)

15.106.2.4 template<class DATATYPE > DATATYPE getG (DATATYPE * *wuvar,* SparseProjection * *sparseStruct,* int *x,* int *y*)

15.106.2.5 template<class DATATYPE > float getSparseVar (DATATYPE * *wuvar,* SparseProjection * *sparseStruct,* int *x,* int *y*)

15.106.2.6 template<class DATATYPE > void setSparseConnectivityFromDense (DATATYPE * *wuvar,* int *preN,* int *postN,* DATATYPE * *tmp_gRNPN,* SparseProjection * *sparseStruct*)

15.106.2.7 void strsearch (string & *s,* const string *trg*)

!!!!find var to check if a string is used in a code (atm it is used to create post-to-pre arrays)

15.107 stringutils.h File Reference

```
#include <string>
```

Macros

- #define __mathFN 56

Functions

- void substitute (string &s, const string trg, const string rep)

  Tool for substituting strings in the neuron code strings or other templates.
- void name_substitutions (string &code, string prefix, vector< string > &names, string postfix=string(""))

  This function performs a list of name substitutions for variables in code snippets.
- void value_substitutions (string &code, vector< string > &names, vector< double > &values)

  This function performs a list of value substitutions for parameters in code snippets.
- void extended_name_substitutions (string &code, string prefix, vector< string > &names, string ext, string postfix=string(""))

  This function performs a list of name substitutions for variables in code snippets where the variables have a prefix/postfix in their names.
- void extended_value_substitutions (string &code, vector< string > &names, string ext, vector< double > &values)

  This function performs a list of value substitutions for parameters in code snippets where the parameters have a prefix/postfix in their names.
- void ensureMathFunctionFtype (string &code, string type)

  This function converts code to contain only explicit single precision (float) function calls (C99 standard)
- void doFinal (string &code, unsigned int i, string type, unsigned int &state)

This function is part of the parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

- string ensureFtype (string oldcode, string type)

  This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

Variables

- string digits = string("0123456789")
- string op = string("+-*/(<>= ,;")+string("\n")+string("\t")
- const char * __dnames [__mathFN]
- const char * __fnames [__mathFN]

15.107.1 Macro Definition Documentation

15.107.1.1 #define __mathFN 56

15.107.2 Function Documentation

15.107.2.1 void doFinal (string & *code*, unsigned int *i*, string *type*, unsigned int & *state*)

This function is part of the parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

15.107.2.2 string ensureFtype (string *oldcode*, string *type*)

This function implements a parser that converts any floating point constant in a code snippet to a floating point constant with an explicit precision (by appending "f" or removing it).

15.107.2.3 void ensureMathFunctionFtype (string & *code*, string *type*)

This function converts code to contain only explicit single precision (float) function calls (C99 standard)

15.107.2.4 void extended_name_substitutions (string & *code*, string *prefix*, vector< string > & *names*, string *ext*, string *postfix* = `string("")`)

This function performs a list of name substitutions for variables in code snippets where the variables have a prefix/postfix in their names.

15.107.2.5 void extended_value_substitutions (string & *code*, vector< string > & *names*, string *ext*, vector< double > & *values*)

This function performs a list of value substitutions for parameters in code snippets where the parameters have a prefix/postfix in their names.

15.107.2.6 void name_substitutions (string & *code*, string *prefix*, vector< string > & *names*, string *postfix* = `string("")`)

This function performs a list of name substitutions for variables in code snippets.

15.107.2.7 void substitute (string & *s*, const string *trg*, const string *rep*)

Tool for substituting strings in the neuron code strings or other templates.

15.107.2.8 void value_substitutions (string & *code*, vector< string > & *names*, vector< double > & *values*)

This function performs a list of value substitutions for parameters in code snippets.

15.107.3 Variable Documentation

15.107.3.1 const char* __dnames[__mathFN]

15.107.3.2 const char* __fnames[__mathFN]

15.107.3.3 string digits = string("0123456789")

15.107.3.4 string op = string("+-*/(<>= ,;")+string("\n")+string("\t")

15.108 SynDelay.cc File Reference

```
#include "modelSpec.h"
#include "modelSpec.cc"
```

Macros

- #define DT 1.0f

Functions

- void modelDefinition (NNmodel &model)

Variables

- double input_p [4]
- double input_ini [2]
- double postExpInp [2]
- double inter_p [4]
- double inter_ini [2]
- double postExpInt [2]
- double output_p [4]
- double output_ini [2]
- double postExpOut [2]
- double * synapses_p = NULL
- double inputInter_ini [1]
- double inputOutput_ini [1]
- double interOutput_ini [1]
- double * postSynV = NULL
- double constInput = 4.0

15.108.1 Macro Definition Documentation

15.108.1.1 #define DT 1.0f

15.108.2 Function Documentation

15.108.2.1 void modelDefinition (NNmodel & *model*)

15.108.3 Variable Documentation

15.108.3.1 double constInput = 4.0

15.108.3.2 double input_ini[2]

Initial value:

```
= {
  -65,
  -20
}
```

15.108.3.3 double input_p[4]

Initial value:

```
= {
  0.02,
  0.2,
  -65,
  6
}
```

15.108.3.4 double inputInter_ini[1]

Initial value:

```
= {
  0.06
}
```

15.108.3.5 double inputOutput_ini[1]

Initial value:

```
= {
  0.03
}
```

15.108.3.6 double inter_ini[2]

Initial value:

```
= {
  -65,
  -20
}
```

15.108.3.7 double inter_p[4]

Initial value:

```
= {
  0.02,
  0.2,
  -65,
  6
}
```

15.108.3.8 double interOutput_ini[1]

Initial value:

```
= {
  0.03
}
```

15.108.3.9 double output_ini[2]

Initial value:

```
= {
  -65,
  -20
}
```

15.108.3.10 double output_p[4]

Initial value:

```
= {
  0.02,
  0.2,
  -65,
  6
}
```

15.108.3.11 double postExpInp[2]

Initial value:

```
= {
  1.0,
  0.0
}
```

15.108.3.12 double postExpInt[2]

Initial value:

```
= {
  1.0,
  0.0
}
```

15.108.3.13 double postExpOut[2]

Initial value:

```
= {
  1.0,
  0.0
}
```

15.108.3.14 double* postSynV = NULL

15.108.3.15 double* synapses_p = NULL

15.109 SynDelaySim.cu File Reference

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "SynDelaySim.h"
#include "SynDelay_CODE/definitions.h"
#include "SynDelay_CODE/runner.cc"
```

Macros

- #define SYNDELAYSIM_CU

Functions

- int main (int argc, char ∗argv[])

15.109.1 Macro Definition Documentation

15.109.1.1 #define SYNDELAYSIM_CU

15.109.2 Function Documentation

15.109.2.1 int main (int *argc,* char ∗ *argv[]*)

15.110 SynDelaySim.h File Reference

Classes

- class SynDelay

Macros

- #define DT 1.0f
- #define TOTAL_TIME 5000.0f
- #define REPORT_TIME 1000.0f

15.110.1 Macro Definition Documentation

15.110.1.1 #define DT 1.0f

15.110.1.2 #define REPORT_TIME 1000.0f

15.110.1.3 #define TOTAL_TIME 5000.0f

15.111 testHelper.h File Reference

Functions

- template<class T >
  T absDiff (T ∗x0, T ∗x1, int n)

15.111.1 Function Documentation

15.111.1.1 template<class T > T absDiff (T ∗ *x0,* T ∗ *x1,* int *n*)

15.112 testHelper.h File Reference

Functions

- template<class T >
  T absDiff (T ∗x0, T ∗x1, int n)

15.112.1 Function Documentation

15.112.1.1 template<class T > T absDiff (T * x0, T * x1, int n)

15.113 testPostVarsInPostLearn.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInPostLearn.h"
#include "postVarsInPostLearn_CODE/definitions.h"
#include "postVarsInPostLearn_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char *argv[])

15.113.1 Macro Definition Documentation

15.113.1.1 #define TESTPREVARSINSIMCODE_CU

15.113.2 Function Documentation

15.113.2.1 int main (int argc, char * argv[])

15.114 testPostVarsInPostLearn.h File Reference

Classes

- class postVarsInPostLearn

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.114.1 Macro Definition Documentation

15.114.1.1 #define DT 0.1f

15.114.1.2 #define REPORT_TIME 1.0f

15.114.1.3 #define TOTAL_TIME 20.0f

15.115 testPostVarsInPostLearn_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInPostLearn_sparse.h"
#include "postVarsInPostLearn_sparse_CODE/definitions.h"
#include "postVarsInPostLearn_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINPOSTLEARN_SPARSE_CU

Functions

- int main (int argc, char ∗argv[])

15.115.1 Macro Definition Documentation

15.115.1.1 #define TESTPREVARSINPOSTLEARN_SPARSE_CU

15.115.2 Function Documentation

15.115.2.1 int main (int *argc,* char ∗ *argv[]*)

15.116 testPostVarsInPostLearn_sparse.h File Reference

Classes

- class postVarsInPostLearn_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.116.1 Macro Definition Documentation

15.116.1.1 #define DT 0.1f

15.116.1.2 #define REPORT_TIME 1.0f

15.116.1.3 #define TOTAL_TIME 20.0f

15.117 testPostVarsInSimCode.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInSimCode.h"
#include "postVarsInSimCode_CODE/definitions.h"
#include "postVarsInSimCode_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char ∗argv[])

15.117.1 Macro Definition Documentation

15.117.1.1 #define TESTPREVARSINSIMCODE_CU

15.117.2 Function Documentation

15.117.2.1 int main (int *argc*, char ∗ *argv[]*)

15.118 testPostVarsInSimCode.h File Reference

Classes

- class postVarsInSimCode

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.118.1 Macro Definition Documentation

15.118.1.1 #define DT 0.1f

15.118.1.2 #define REPORT_TIME 1.0f

15.118.1.3 #define TOTAL_TIME 20.0f

15.119 testPostVarsInSimCode_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInSimCode_sparse.h"
#include "postVarsInSimCode_sparse_CODE/definitions.h"
#include "postVarsInSimCode_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char ∗argv[])

15.119.1 Macro Definition Documentation

15.119.1.1 #define TESTPREVARSINSIMCODE_CU

15.119.2 Function Documentation

15.119.2.1 int main (int *argc*, char ∗ *argv[]*)

15.120 testPostVarsInSimCode_sparse.h File Reference

Classes

- class postVarsInSimCode_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.120.1 Macro Definition Documentation

15.120.1.1 #define DT 0.1f

15.120.1.2 #define REPORT_TIME 1.0f

15.120.1.3 #define TOTAL_TIME 20.0f

15.121 testPostVarsInSynapseDynamics.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInSynapseDynamics.h"
#include "postVarsInSynapseDynamics_CODE/definitions.h"
#include "postVarsInSynapseDynamics_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSYNAPSEDYNAMICS_CU

Functions

- int main (int argc, char ∗argv[])

15.121.1 Macro Definition Documentation

15.121.1.1 #define TESTPREVARSINSYNAPSEDYNAMICS_CU

15.121.2 Function Documentation

15.121.2.1 int main (int *argc*, char ∗ *argv[]*)

15.122 testPostVarsInSynapseDynamics.h File Reference

Classes

- class postVarsInSynapseDynamics

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.122.1 Macro Definition Documentation

15.122.1.1 #define DT 0.1f

15.122.1.2 #define REPORT_TIME 1.0f

15.122.1.3 #define TOTAL_TIME 20.0f

15.123 testPostVarsInSynapseDynamics_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPostVarsInSynapseDynamics_sparse.h"
#include "postVarsInSynapseDynamics_sparse_CODE/definitions.h"
#include "postVarsInSynapseDynamics_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_CU

Functions

- int main (int argc, char ∗argv[])

15.123.1 Macro Definition Documentation

15.123.1.1 #define TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_CU

15.123.2 Function Documentation

15.123.2.1 int main (int *argc,* char ∗ *argv[]*)

15.124 testPostVarsInSynapseDynamics_sparse.h File Reference

Classes

- class postVarsInSynapseDynamics_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.124.1 Macro Definition Documentation

15.124.1.1 #define DT 0.1f

15.124.1.2 #define REPORT_TIME 1.0f

15.124.1.3 #define TOTAL_TIME 20.0f

15.125 testPreVarsInPostLearn.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInPostLearn.h"
#include "preVarsInPostLearn_CODE/definitions.h"
#include "preVarsInPostLearn_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char ∗argv[])

15.125.1 Macro Definition Documentation

15.125.1.1 #define TESTPREVARSINSIMCODE_CU

15.125.2 Function Documentation

15.125.2.1 int main (int *argc*, char ∗ *argv[]*)

15.126 testPreVarsInPostLearn.h File Reference

Classes

- class preVarsInPostLearn

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.126.1 Macro Definition Documentation

15.126.1.1 #define DT 0.1f

15.126.1.2 #define REPORT_TIME 1.0f

15.126.1.3 #define TOTAL_TIME 20.0f

15.127 testPreVarsInPostLearn_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInPostLearn_sparse.h"
#include "preVarsInPostLearn_sparse_CODE/definitions.h"
#include "preVarsInPostLearn_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINPOSTLEARN_SPARSE_CU

Functions

- int main (int argc, char ∗argv[])

15.127.1 Macro Definition Documentation

15.127.1.1 #define TESTPREVARSINPOSTLEARN_SPARSE_CU

15.127.2 Function Documentation

15.127.2.1 int main (int *argc*, char ∗ *argv[]*)

15.128 testPreVarsInPostLearn_sparse.h File Reference

Classes

- class preVarsInPostLearn_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.128.1 Macro Definition Documentation

15.128.1.1 #define DT 0.1f

15.128.1.2 #define REPORT_TIME 1.0f

15.128.1.3 #define TOTAL_TIME 20.0f

15.129 testPreVarsInSimCode.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInSimCode.h"
#include "preVarsInSimCode_CODE/definitions.h"
#include "preVarsInSimCode_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char ∗argv[])

15.129.1 Macro Definition Documentation

15.129.1.1 #define TESTPREVARSINSIMCODE_CU

15.129.2 Function Documentation

15.129.2.1 int main (int *argc,* char ∗ *argv[]*)

15.130 testPreVarsInSimCode.h File Reference

Classes

- class preVarsInSimCode

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.130.1 Macro Definition Documentation

15.130.1.1 #define DT 0.1f

15.130.1.2 #define REPORT_TIME 1.0f

15.130.1.3 #define TOTAL_TIME 20.0f

15.131 testPreVarsInSimCode_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInSimCode_sparse.h"
#include "preVarsInSimCode_sparse_CODE/definitions.h"
#include "preVarsInSimCode_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSIMCODE_CU

Functions

- int main (int argc, char ∗argv[])

15.131.1 Macro Definition Documentation

15.131.1.1 #define TESTPREVARSINSIMCODE_CU

15.131.2 Function Documentation

15.131.2.1 int main (int *argc*, char ∗ *argv[]*)

15.132 testPreVarsInSimCode_sparse.h File Reference

Classes

- class preVarsInSimCode_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.132.1 Macro Definition Documentation

15.132.1.1 #define DT 0.1f

15.132.1.2 #define REPORT_TIME 1.0f

15.132.1.3 #define TOTAL_TIME 20.0f

15.133 testPreVarsInSynapseDynamics.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInSynapseDynamics.h"
#include "preVarsInSynapseDynamics_CODE/definitions.h"
#include "preVarsInSynapseDynamics_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSYNAPSEDYNAMICS_CU

Functions

- int main (int argc, char ∗argv[])

15.133.1 Macro Definition Documentation

15.133.1.1 #define TESTPREVARSINSYNAPSEDYNAMICS_CU

15.133.2 Function Documentation

15.133.2.1 int main (int *argc*, char ∗ *argv[]*)

15.134 testPreVarsInSynapseDynamics.h File Reference

Classes

- class preVarsInSynapseDynamics

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.134.1 Macro Definition Documentation

15.134.1.1 #define DT 0.1f

15.134.1.2 #define REPORT_TIME 1.0f

15.134.1.3 #define TOTAL_TIME 20.0f

15.135 testPreVarsInSynapseDynamics_sparse.cu File Reference

```
#include <cstdlib>
#include <cfloat>
#include <iostream>
#include <fstream>
#include "hr_time.cpp"
#include "utils.h"
#include "testHelper.h"
#include "testPreVarsInSynapseDynamics_sparse.h"
#include "preVarsInSynapseDynamics_sparse_CODE/definitions.h"
#include "preVarsInSynapseDynamics_sparse_CODE/runner.cc"
```

Macros

- #define TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_CU

Functions

- int main (int argc, char ∗argv[])

15.135.1 Macro Definition Documentation

15.135.1.1 #define TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_CU

15.135.2 Function Documentation

15.135.2.1 int main (int *argc*, char ∗ *argv[]*)

15.136 testPreVarsInSynapseDynamics_sparse.h File Reference

Classes

- class preVarsInSynapseDynamics_sparse

Macros

- #define DT 0.1f
- #define TOTAL_TIME 20.0f
- #define REPORT_TIME 1.0f

15.136.1 Macro Definition Documentation

15.136.1.1 #define DT 0.1f

15.136.1.2 #define REPORT_TIME 1.0f

15.136.1.3 #define TOTAL_TIME 20.0f

15.137 toString.h File Reference

Contains a template function for string conversion from const char∗ to C++ string.

```
#include <string>
#include <sstream>
```

Macros

- #define _TOSTRING_H_

  macro for avoiding multiple inclusion during compilation

- #define tS(X) toString(X)

  Macro providing the abbreviated syntax tS() instead of toString().

Functions

- template<class T >
  std::string toString (T t)

  template function for string conversion from const char to C++ string*

- template<>
  std::string toString (float t)

- template<>
  std::string toString (double t)

15.137.1 Detailed Description

Contains a template function for string conversion from const char* to C++ string.

15.137.2 Macro Definition Documentation

15.137.2.1 #define _TOSTRING_H_

macro for avoiding multiple inclusion during compilation

15.137.2.2 #define tS(*X*) toString(X)

Macro providing the abbreviated syntax tS() instead of toString().

15.137.3 Function Documentation

15.137.3.1 template<class T > std::string toString (T *t*)

template function for string conversion from const char* to C++ string

15.137.3.2 template<> std::string toString (float *t*)

15.137.3.3 template<> std::string toString (double *t*)

15.138 utils.h File Reference

This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models.

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <memory>
#include <fstream>
#include <cmath>
#include <cuda_runtime.h>
#include "modelSpec.h"
#include "toString.h"
#include "stringutils.h"
#include "extra_neurons.h"
#include "extra_postsynapses.h"
#include "extra_weightupdates.h"
#include "numlib/simpleBit.h"
```

Classes

- class rulkovdp

  Class defining the dependent parameters of teh Rulkov map neuron.
- class expDecayDp

  Class defining the dependent parameter for exponential decay.
- class pwSTDP

  TODO This class definition may be code-generated in a future release.

Macros

- #define _UTILS_H_

  macro for avoiding multiple inclusion during compilation
- #define CHECK_CUDA_ERRORS(call)

  Macro for wrapping cuda runtime function calls and catching any errors that may be thrown.

Functions

- void gennError (string error)

  Function called upon the detection of an error. Outputs an error message and then exits.
- void gennError (const char *error)

  Function called upon the detection of an error. Outputs an error message and then exits.
- void writeHeader (ostream &os)

  Function to write the comment header denoting file authorship and contact details into the generated code.
- unsigned int theSize (string type)

  Tool for determining the size of variable types on the current architecture.
- void prepareStandardModels ()

  Function that defines standard neuron models.
- void preparePostSynModels ()

  Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.
- void prepareWeightUpdateModels ()

  Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

Variables

- vector< neuronModel > nModels

  Global C++ vector containing all neuron model descriptions.
- vector< postSynModel > postSynModels

  Global C++ vector containing all post-synaptic update model descriptions.
- vector< weightUpdateModel > weightUpdateModels

  Global C++ vector containing all weightupdate model descriptions.

15.138.1 Detailed Description

This file contains standard utility functions provide within the NVIDIA CUDA software development toolkit (SDK). The remainder of the file contains a function that defines the standard neuron models.

15.138.2 Macro Definition Documentation

15.138.2.1 #define _UTILS_H_

macro for avoiding multiple inclusion during compilation

15.138.2.2 #define CHECK_CUDA_ERRORS(*call*)

Value:

```
{                                                                           \
  cudaError_t error = call;                                                 \
  if (error != cudaSuccess)                                                 \
  {                                                                         \
    fprintf(stderr, "%s: %i: cuda error %i: %s\n",                          \
            __FILE__, __LINE__, (int)error, cudaGetErrorString(error));     \
    exit(EXIT_FAILURE);                                                     \
  }                                                                         \
}
```

Macro for wrapping cuda runtime function calls and catching any errors that may be thrown.

15.138.3 Function Documentation

15.138.3.1 void gennError (string *error*)

Function called upon the detection of an error. Outputs an error message and then exits.

15.138.3.2 void gennError (const char * *error*)

Function called upon the detection of an error. Outputs an error message and then exits.

15.138.3.3 void preparePostSynModels ( )

Function that prepares the standard post-synaptic models, including their variables, parameters, dependent parameters and code strings.

15.138.3.4 void prepareStandardModels ( )

Function that defines standard neuron models.

The neuron models are defined and added to the C++ vector nModels that is holding all neuron model descriptions. User defined neuron models can be appended to this vector later in (a) separate function(s).

15.138.3.5 void prepareWeightUpdateModels ( )

Function that prepares the standard (pre) synaptic models, including their variables, parameters, dependent parameters and code strings.

15.138.3.6 unsigned int theSize (string *type*)

Tool for determining the size of variable types on the current architecture.

15.138.3.7 void writeHeader (ostream & *os*)

Function to write the comment header denoting file authorship and contact details into the generated code.

15.138.4 Variable Documentation

15.138.4.1 vector<neuronModel> nModels

Global C++ vector containing all neuron model descriptions.

15.138.4.2 vector<postSynModel> postSynModels

Global C++ vector containing all post-synaptic update model descriptions.

15.138.4.3 vector<weightUpdateModel> weightUpdateModels

Global C++ vector containing all weightupdate model descriptions.

15.139 VClampGA.cu File Reference

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU.

```
#include "VClampGA.h"
```

Functions

- int main (int argc, char ∗argv[])

  This function is the entry point for running the project.

15.139.1 Detailed Description

Main entry point for the GeNN project demonstrating realtime fitting of a neuron with a GA running mostly on the GPU.

15.139.2 Function Documentation

15.139.2.1 int main (int *argc,* char ∗ *argv[]*)

This function is the entry point for running the project.

15.140 VClampGA.h File Reference

```
#include <cassert>
#include "hr_time.cpp"
#include "utils.h"
#include <cuda_runtime.h>
#include "HHVClamp.cc"
#include "HHVClamp_CODE/runner.cc"
#include "../../../lib/include/numlib/randomGen.h"
#include "../../../lib/include/numlib/gauss.h"
#include "helper.h"
#include "GA.cc"
```

Macros

- #define RAND(Y, X) Y = Y ∗ 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767

Variables

- randomGen R
- randomGauss RG
- double t = 0.0f
- unsigned int iT = 0
- CStopWatch timer

15.140.1 Macro Definition Documentation

15.140.1.1 #define RAND(*Y, X*) Y = Y ∗ 1103515245 +12345;X= (unsigned int)(Y >> 16) & 32767

15.140.2 Variable Documentation

15.140.2.1 unsigned int iT = 0

15.140.2.2 randomGen R

15.140.2.3 randomGauss RG

15.140.2.4 double t = 0.0f

15.140.2.5 CStopWatch timer

References

[1] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. 7, 8, 9, 20

[2] Thomas Nowotny, Ramón Huerta, Henry DI Abarbanel, and Mikhail I Rabinovich. Self-organization in the olfactory system: one shot odor recognition in insects. *Biological cybernetics*, 93(6):436–446, 2005. 9, 10, 17, 25

[3] Nikolai F Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65(4):041922, 2002. 17

[4] R. D. Traub and R. Miles. *Neural Networks of the Hippocampus*. Cambridge University Press, New York, 1991. 9, 19, 26, 28

Index

_FTYPE
PoissonIzh.cc, 177
_GLOBAL_H_
global.h, 138
_IZH_SPARSE_MODEL_CC_
Izh_sparse_model.cc, 145
_MAP_CLASSOL_CC_
MBody1_project/model/map_classol.cc, 146
MBody_delayedSyn_project/model/map_classol.↩
cc, 147
MBody_individualID_project/model/map_classol.↩
cc, 147
MBody_userdef_project/model/map_classol.cc, 147
_MODELSPEC_CC_
modelSpec.cc, 168
_MODELSPEC_H_
modelSpec.h, 171
_NC1
sizes.h, 196
_ONECOMP_MODEL_CC_
OneComp_model.cc, 175
_POISSONIZHMODEL_CC_
PoissonIzh-model.cc, 177
_SCHMUKER2014_CLASSIFIER_
Schmuker2014_classifier.cu, 195
_TOSTRING_H_
toString.h, 215
_UTILS_H_
utils.h, 217
__CODE_HELPER_CC
CodeHelper.cc, 110
__dnames
stringutils.h, 199
__fnames
stringutils.h, 199
__mathFN
stringutils.h, 198
∼NNmodel
NNmodel, 69
∼Schmuker2014_classifier
Schmuker2014_classifier, 97
∼SynDelay
SynDelay, 101
∼classIzh
classIzh, 48
∼classol
classol, 53
∼neuronpop
neuronpop, 65
∼postVarsInPostLearn
postVarsInPostLearn, 81
∼postVarsInPostLearn_sparse
postVarsInPostLearn_sparse, 82
∼postVarsInSimCode
postVarsInSimCode, 83
∼postVarsInSimCode_sparse
postVarsInSimCode_sparse, 83
∼postVarsInSynapseDynamics
postVarsInSynapseDynamics, 84
∼postVarsInSynapseDynamics_sparse
postVarsInSynapseDynamics_sparse, 85
∼preVarsInPostLearn
preVarsInPostLearn, 86
∼preVarsInPostLearn_sparse
preVarsInPostLearn_sparse, 86
∼preVarsInSimCode
preVarsInSimCode, 87
∼preVarsInSimCode_sparse
preVarsInSimCode_sparse, 88
∼preVarsInSynapseDynamics
preVarsInSynapseDynamics, 89
∼preVarsInSynapseDynamics_sparse
preVarsInSynapseDynamics_sparse, 89
∼randomGauss
randomGauss, 92
∼randomGen
randomGen, 93
∼stdRG
stdRG, 100
00_MainPage.dox, 104
01_Installation.dox, 104
02_Quickstart.dox, 104
03_Examples.dox, 104
09_ReleaseNotes_v2.dox, 104
10_UserManual.dox, 104
11_Tutorial.dox, 104
12_Tutorial.dox, 104
13_UserGuide.dox, 104
14_Credits.dox, 104

ALLTOALL
modelSpec.h, 171
AUTODEVICE
modelSpec.h, 171
absDiff
postsynapticVariables/testHelper.h, 202
presynapticVariables/testHelper.h, 203
activateDirectInput
NNmodel, 69
addInputRate
Schmuker2014_classifier, 97
addNeuronPopulation
NNmodel, 69, 70
addSynapsePopulation
NNmodel, 70, 72, 73
allocate_device_mem_input
classIzh, 48
classol, 53
neuronpop, 65

allocate_device_mem_patterns
classIzh, 48
classol, 53
neuronpop, 65
allocateHostAndDeviceMemory
Schmuker2014_classifier, 97
applyInputToClassifier
experiment.cu, 112
applyLearningRuleSynapses
Schmuker2014_classifier, 97
asGoodAsZero
global.h, 138

B
simpleBit.h, 196
baseV
inputSpec, 62
baserates
classol, 58
BlkSz
GeNNHelperKrnls.cu, 137
braces
CodeHelper, 59

CACHE_DIR
experiment.h, 113
CB
CodeHelper.cc, 110
CHECK_CUDA_ERRORS
utils.h, 217
CLUST_SIZE_AN
Model_Schmuker_2014_classifier.cc, 167
CLUST_SIZE_PN
Model_Schmuker_2014_classifier.cc, 167
CLUST_SIZE_RN
Model_Schmuker_2014_classifier.cc, 167
CONNECTIVITY_AN_AN
experiment.h, 113
CONNECTIVITY_PN_AN
experiment.h, 113
CONNECTIVITY_PN_PN
experiment.h, 113
CONNECTIVITY_RN_PN
experiment.h, 114
CONSTINP
modelSpec.h, 171
CPU
modelSpec.h, 171
CStopWatch, 59
CStopWatch, 60
getElapsedTime, 60
startTimer, 60
stopTimer, 60
cacheDir
Schmuker2014_classifier, 99
calculateCurrentWindowWinner
Schmuker2014_classifier, 97
calculateDerivedParameter
dpclass, 60
expDecayDp, 61
pwSTDP, 90
pwSTDP_userdef, 91
rulkovdp, 94
calculateOverallWinner
Schmuker2014_classifier, 97
calculateVrResponse
Schmuker2014_classifier, 97
calculateWinner
Schmuker2014_classifier, 97
Cexp
helper.h, 140
checkContents
Schmuker2014_classifier, 97
checkSizes
NNmodel, 74
chooseDevice
generateALL.cc, 132
chooseGPUDevice
NNmodel, 75
classIzh, 47
∼classIzh, 48
allocate_device_mem_input, 48
allocate_device_mem_patterns, 48
classIzh, 48
copy_device_mem_input, 48
create_input_values, 48
d_input1, 49
d_input2, 49
free_device_mem, 48
gen_alltoall_syns, 48
getSpikeNumbersFromGPU, 49
getSpikesFromGPU, 49
init, 49
initializeAllVars, 49
input1, 49
input2, 49
model, 49
output_params, 49
output_spikes, 49
output_state, 49
randomizeVar, 49
randomizeVarSq, 49
read_input_values, 49
read_sparsesyns_par, 49
run, 49
setInput, 49
sum_spikes, 49
sumPExc, 49
sumPInh, 49
write_input_to_file, 49
classLabel
Schmuker2014_classifier, 99
classifier
experiment.cu, 112
classol, 50
∼classol, 53
allocate_device_mem_input, 53

allocate_device_mem_patterns, 53
baserates, 58
classol, 53
d_baserates, 58
d_pattern, 58
free_device_mem, 53
generate_baserates, 53
get_kcdnsyns, 53, 54
getSpikeNumbersFromGPU, 54
getSpikesFromGPU, 54
init, 54
model, 58
offset, 58
output_spikes, 54, 55
output_state, 55
p_pattern, 58
pattern, 58
read_PNIzh1syns, 55
read_input_patterns, 55
read_kcdnsyns, 55
read_pnkcsyns, 55, 56
read_pnlhisyns, 56
read_sparsesyns_par, 56
run, 56, 57
runCPU, 57
runGPU, 57
size_g, 58
sum_spikes, 57
sumDN, 58
sumIzh1, 58
sumKC, 58
sumLHI, 58
sumPN, 58
theRates, 58
write_kcdnsyns, 57
write_pnkcsyns, 57, 58
write_pnlhisyns, 58
classol_sim.cu, 104, 105
classol_sim.h, 106–109
clear
extra_neurons.h, 115
lib/include/extra_postsynapses.h, 116
SpineML_to_GeNN/extra_postsynapses.h, 117
clearDownDevice
Schmuker2014_classifier, 97
clearedDownDevice
Schmuker2014_classifier, 99
closeBrace
CodeHelper, 59
clusterSpikeCountAN
Schmuker2014_classifier, 99
CodeHelper, 58
braces, 59
closeBrace, 59
CodeHelper, 59
endl, 59
openBrace, 59
setVerbose, 59
verbose, 59
CodeHelper.cc, 110
__CODE_HELPER_CC, 110
CB, 110
ENDL, 110
OB, 110
SAVEP, 110
compareErrTupel
GA.cc, 118
constInput
SynDelay.cc, 199
convertToRateCode
Schmuker2014_classifier, 97
copy_device_mem_input
classIzh, 48
neuronpop, 65
copy_var
helper.h, 140
correctClass
Schmuker2014_classifier, 99
countAN
Schmuker2014_classifier, 99
countEntriesAbove
sparseUtils.cc, 197
countPN
Schmuker2014_classifier, 99
countPNAN
Schmuker2014_classifier, 99
countRN
Schmuker2014_classifier, 99
create_input_values
classIzh, 48
neuronpop, 65
createDirectory
experiment.cu, 112
createPosttoPreArray
sparseUtils.cc, 197
createSparseConnectivityFromDense
sparseUtils.cc, 197
createWTAConnectivity
Schmuker2014_classifier, 97

D_MAX_RANDOM_NUM
experiment.h, 114
d_baserates
classol, 58
d_input1
classIzh, 49
neuronpop, 65
d_input2
classIzh, 49
d_inputRates
Schmuker2014_classifier, 99
d_maxRandomNumber
Schmuker2014_classifier, 99
d_pattern
classol, 58
DATASET_NAME
experiment.h, 114

DBG_SIZE
Izh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 109
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
DENSE
modelSpec.h, 171
DT
HHVClamp.cc, 141
HHVclampGA_project/model/MBody1.cc, 149
Izh_sparse.cc, 144
MBody1_project/model/MBody1.cc, 152
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
Model_Schmuker_2014_classifier.cc, 167
OneComp.cc, 174
PoissonIzh.cc, 177
postVarsInPostLearn.cc, 180
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 187
preVarsInPostLearn_sparse.cc, 188
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
SynDelay.cc, 199
SynDelaySim.h, 202
testPostVarsInPostLearn.h, 203
testPostVarsInPostLearn_sparse.h, 204
testPostVarsInSimCode.h, 205
testPostVarsInSimCode_sparse.h, 206
testPostVarsInSynapseDynamics.h, 207
testPostVarsInSynapseDynamics_sparse.h, 208
testPreVarsInPostLearn.h, 209
testPreVarsInPostLearn_sparse.h, 210
testPreVarsInSimCode.h, 211
testPreVarsInSimCode_sparse.h, 212
testPreVarsInSynapseDynamics.h, 213
testPreVarsInSynapseDynamics_sparse.h, 214
data_type
Schmuker2014_classifier, 97
data_type_double
Schmuker2014_classifier, 97
data_type_float
Schmuker2014_classifier, 97
data_type_int
Schmuker2014_classifier, 97
data_type_uint
Schmuker2014_classifier, 97
datasetName
Schmuker2014_classifier, 99
delB
simpleBit.h, 196
deviceCount
global.h, 138
deviceProp
global.h, 138
digits
ensureFtype.h, 111
stringutils.h, 199
directoryExists
experiment.cu, 112
divi
experiment.h, 114
dnp
NNmodel, 75
doFinal
ensureFtype.h, 111
stringutils.h, 198
dpNames
neuronModel, 63
postSynModel, 80
weightUpdateModel, 103
dpclass, 60
calculateDerivedParameter, 60
dpclass, 60
dps
neuronModel, 63
postSynModel, 80
weightUpdateModel, 103
dpsp
NNmodel, 75
dsp_w
NNmodel, 75

EKexp
helper.h, 140
ENDL
CodeHelper.cc, 110
ENaexp
helper.h, 140
EXITSYN
modelSpec.h, 171
EXPDECAY
modelSpec.h, 173
Elexp
helper.h, 140
endl
CodeHelper, 59
ensureFtype
ensureFtype.h, 111
stringutils.h, 198
ensureFtype.h, 111
digits, 111
doFinal, 111
ensureFtype, 111
op, 111

ensureMathFunctionFtype
stringutils.h, 198
err
errTupel, 61
errTupel, 61
err, 61
id, 61
evntThreshold
weightUpdateModel, 103
exIzh_ini
OneComp.cc, 174
PoissonIzh.cc, 178
exIzh_p
OneComp.cc, 174
PoissonIzh.cc, 178
excIzh_p
Izh_sparse.cc, 144
expDecay
expDecayDp, 61
expDecayDp, 61
calculateDerivedParameter, 61
expDecay, 61
experiment.cu, 111
applyInputToClassifier, 112
classifier, 112
createDirectory, 112
directoryExists, 112
getAverage, 112
getStdDev, 112
getUniqueRunId, 112
main, 112
outputRunParameters, 112
Parameter, 112
printTextFile, 112
RAND, 112
S_ISDIR, 112
setDefaultParamValues, 112
vectorContains, 112
experiment.h, 113
CACHE_DIR, 113
CONNECTIVITY_AN_AN, 113
CONNECTIVITY_PN_AN, 113
CONNECTIVITY_PN_PN, 113
CONNECTIVITY_RN_PN, 114
D_MAX_RANDOM_NUM, 114
DATASET_NAME, 114
divi, 114
FLAG_RUN_ON_CPU, 114
GLOBAL_WEIGHT_SCALING, 114
MAX_FIRING_RATE_HZ, 114
MAX_WEIGHT_PN_AN, 114
MIN_FIRING_RATE_HZ, 114
MIN_WEIGHT_PN_AN, 114
N_FOLDING, 114
OUTPUT_DIR, 114
PLASTICITY_INTERVAL_MS, 114
RECORDING_TIME_MS, 114
RECORDINGS_DIR, 114
REPEAT_LEARNING_SET, 114
SPIKING_ACTIVITY_THRESHOLD_HZ, 114
TOTAL_RECORDINGS, 114
timer, 114
UINT, 114
VR_DATA_FILENAME, 114
WEIGHT_DELTA_PN_AN, 114
WEIGHT_RN_PN, 114
WEIGHT_WTA_AN_AN, 114
WEIGHT_WTA_PN_PN, 114
extended_name_substitutions
stringutils.h, 198
extended_value_substitutions
stringutils.h, 198
extra_neurons.h, 114
clear, 115
push_back, 115
simCode, 115
extra_postsynapses.h, 116, 117
extra_weightupdates.h, 118
extraGlobalNeuronKernelParameterTypes
neuronModel, 63
extraGlobalNeuronKernelParameters
neuronModel, 63
extraGlobalSynapseKernelParameterTypes
weightUpdateModel, 103
extraGlobalSynapseKernelParameters
weightUpdateModel, 103

FALSE
modelSpec.h, 171
FLAG_RUN_ON_CPU
experiment.h, 114
final
NNmodel, 76
finalize
NNmodel, 74
findMaxMinSampleDistances
Schmuker2014_classifier, 97
fixGPU
HHVClampParameters.h, 142
free_device_mem
classIzh, 48
classol, 53
ftype
NNmodel, 76

g_ee
gen_syns_sparse_izhModel.cc, 125
g_ei
gen_syns_sparse_izhModel.cc, 125
g_ie
gen_syns_sparse_izhModel.cc, 125
g_ii
gen_syns_sparse_izhModel.cc, 125
GA.cc, 118
compareErrTupel, 118
procreatePop, 118
GAUSS_CC

gauss.cc, 119
GAUSS_H
gauss.h, 119
gDNDN
HHVclampGA_project/model/MBody1.cc, 149
GENN_DOUBLE
modelSpec.h, 171
GENN_FLOAT
modelSpec.h, 171
gKexp
helper.h, 140
gLHIKC
HHVclampGA_project/model/MBody1.cc, 149
GLOBAL_WEIGHT_SCALING
experiment.h, 114
GLOBALG
modelSpec.h, 171
gNaexp
helper.h, 140
GPU
modelSpec.h, 171
garray
gen_syns_sparse_izhModel.cc, 125
garray_ee
gen_syns_sparse_izhModel.cc, 125
garray_ei
gen_syns_sparse_izhModel.cc, 125
garray_ie
gen_syns_sparse_izhModel.cc, 125
garray_ii
gen_syns_sparse_izhModel.cc, 125
gauss.cc, 118
GAUSS_CC, 119
gauss.h, 119
GAUSS_H, 119
GeNNHelperKrnls.cu, 136
BlkSz, 137
generate_random_gpuInput_xorwow, 137
setup_kernel, 137
xorwow_setup, 137
GeNNReady
modelSpec.h, 173
gen_alltoall_syns
classIzh, 48
gen_input_structured.cc, 119
main, 120
R, 120
gen_kcdn_syns.cc, 120
main, 120
R, 120
RG, 120
gen_kcdn_syns_fixto10K.cc, 120
main, 121
R, 121
R2, 121
RG, 121
gen_pnkc_syns.cc, 121
main, 122
R, 122
RG, 122
gen_pnkc_syns_indivID.cc, 122
main, 122
R, 122
RG, 122
gen_pnlhi_syns.cc, 122
main, 123
gen_syns_sparse.cc, 123
main, 123
R, 123
RG, 124
gen_syns_sparse_izhModel.cc, 124
g_ee, 125
g_ei, 125
g_ie, 125
g_ii, 125
garray, 125
garray_ee, 125
garray_ei, 125
garray_ie, 125
garray_ii, 125
gsyn, 125
ind, 125
ind_ee, 125
ind_ei, 125
ind_ie, 125
ind_ii, 125
indInG_ee, 125
indInG_ei, 125
indInG_ie, 125
indInG_ii, 125
main, 125
printVector, 125
R, 125
Rind, 125
genNeuronFunction
generateCPU.cc, 134
genNeuronKernel
generateKernels.cc, 135
genRunner
generateRunner.cc, 136
genRunnerGPU
generateRunner.cc, 136
genSynapseFunction
generateCPU.cc, 134
genSynapseKernel
generateKernels.cc, 135
generate_baserates
classol, 53
generate_inputrates_dataset
Schmuker2014_classifier, 97
generate_model_runner
generateALL.cc, 132
generate_or_load_inputrates_dataset
Schmuker2014_classifier, 97
generate_process_presynaptic_events_code
generateKernels.cc, 135

generate_process_presynaptic_events_code_CPU
generateCPU.cc, 133
generate_random_gpuInput_xorwow
GeNNHelperKrnls.cu, 137
generate_run.cc, 126–131
generateALL.cc, 132
chooseDevice, 132
generate_model_runner, 132
main, 133
generateCPU.cc, 133
genNeuronFunction, 134
genSynapseFunction, 134
generate_process_presynaptic_events_code_C↩
PU, 133
generateKernels.cc, 134
genNeuronKernel, 135
genSynapseKernel, 135
generate_process_presynaptic_events_code, 135
hlp, 135
isGrpVarNeeded, 135
generateRunner.cc, 136
genRunner, 136
genRunnerGPU, 136
generateSimulatedTimeSeriesData
Schmuker2014_classifier, 97
gennError
utils.h, 217
get_kcdnsyns
classol, 53, 54
getAverage
experiment.cu, 112
getClassCluster
Schmuker2014_classifier, 97
getClusterIndex
Schmuker2014_classifier, 97
getElapsedTime
CStopWatch, 60
getG
sparseUtils.cc, 197
getManhattanDistance
Schmuker2014_classifier, 97
getRand0to1
Schmuker2014_classifier, 97
getRecordingFilename
Schmuker2014_classifier, 98
getSampleDistance
Schmuker2014_classifier, 98
getSparseVar
sparseUtils.cc, 197
getSpikeNumbersFromGPU
classIzh, 49
classol, 54
neuronpop, 65
Schmuker2014_classifier, 98
getSpikesFromGPU
classIzh, 49
classol, 54
neuronpop, 65
Schmuker2014_classifier, 98
getStdDev
experiment.cu, 112
getUniqueRunId
experiment.cu, 112
glexp
helper.h, 140
global.h, 137
_GLOBAL_H_, 138
asGoodAsZero, 138
deviceCount, 138
deviceProp, 138
hostCount, 138
learnBlkSz, 138
logUIntSz, 138
neuronBlkSz, 138
optimiseBlockSize, 138
synapseBlkSz, 138
theDev, 139
UIntSz, 139
globalInp
NNmodel, 76
gpKCDN
MBody_userdef.cc, 164
gpPNKC
MBody_userdef.cc, 164
gsyn
gen_syns_sparse_izhModel.cc, 125

HHVClamp.cc, 141
DT, 141
modelDefinition, 141
myHH_ini, 142
myHH_p, 142
HHVClampParameters.h, 142
fixGPU, 142
NPOP, 142
TOTALT, 142
HHVclampGA_project/generate_run.cc
main, 126
HHVclampGA_project/model/MBody1.cc
DT, 149
gDNDN, 149
gLHIKC, 149
modelDefinition, 149
myDNDN_p, 149
myKCDN_p, 149
myLHIKC_p, 149
myPNKC_p, 150
myPNLHI_p, 150
myPOI_ini, 150
myPOI_p, 150
postExpDNDN, 150
postExpKCDN, 150
postExpLHIKC, 150
postExpPNKC, 151
postExpPNLHI, 151
postSynV, 151
stdTM_ini, 151

stdTM_p, 151
HR_TIMER
hr_time.cpp, 142
helper.h, 139
Cexp, 140
copy_var, 140
EKexp, 140
ENaexp, 140
Elexp, 140
gKexp, 140
gNaexp, 140
glexp, 140
hexp, 140
initI, 140
initexpHH, 140
limit, 140
mexp, 140
nexp, 140
operator<<, 140
runexpHH, 140
sigC, 140
sigEK, 140
sigENa, 141
sigEl, 140
sigGK, 141
sigGNa, 141
sigGl, 141
single_var_init_fullrange, 140
single_var_reinit, 140
truevar_init, 140
truevar_initexpHH, 140
var_init_fullrange, 140
var_reinit, 140
Vexp, 141
write_para, 140
hexp
helper.h, 140
hlp
generateKernels.cc, 135
hostCount
global.h, 138
hr_time.cpp, 142
HR_TIMER, 142
hr_time.h, 143

INDIVIDUALG
modelSpec.h, 171
INDIVIDUALID
modelSpec.h, 171
INHIBSYN
modelSpec.h, 172
INJECTCURRENT
PoissonIzh_sim.h, 180
INPRULE
modelSpec.h, 172
iT
Izh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 108
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
VClampGA.h, 219
IZHIKEVICH
modelSpec.h, 173
IZHIKEVICH_PS
modelSpec.h, 173
IZHIKEVICH_V
modelSpec.h, 173
id
errTupel, 61
inSyn
NNmodel, 76
ind
gen_syns_sparse_izhModel.cc, 125
ind_ee
gen_syns_sparse_izhModel.cc, 125
ind_ei
gen_syns_sparse_izhModel.cc, 125
ind_ie
gen_syns_sparse_izhModel.cc, 125
ind_ii
gen_syns_sparse_izhModel.cc, 125
indInG_ee
gen_syns_sparse_izhModel.cc, 125
indInG_ei
gen_syns_sparse_izhModel.cc, 125
indInG_ie
gen_syns_sparse_izhModel.cc, 125
indInG_ii
gen_syns_sparse_izhModel.cc, 125
individualSpikeCountPN
Schmuker2014_classifier, 99
inhIzh_p
Izh_sparse.cc, 144
init
classIzh, 49
classol, 54
neuronpop, 65
init_neurons
postVarsInPostLearn, 81
postVarsInPostLearn_sparse, 82
postVarsInSimCode, 83
postVarsInSimCode_sparse, 83
postVarsInSynapseDynamics, 84
postVarsInSynapseDynamics_sparse, 85
preVarsInPostLearn, 86
preVarsInPostLearn_sparse, 86
preVarsInSimCode, 87
preVarsInSimCode_sparse, 88
preVarsInSynapseDynamics, 89
preVarsInSynapseDynamics_sparse, 89
init_synapses

postVarsInPostLearn, 81
postVarsInPostLearn_sparse, 82
postVarsInSimCode, 83
postVarsInSimCode_sparse, 83
postVarsInSynapseDynamics, 84
postVarsInSynapseDynamics_sparse, 85
preVarsInPostLearn, 86
preVarsInPostLearn_sparse, 86
preVarsInSimCode, 87
preVarsInSimCode_sparse, 88
preVarsInSynapseDynamics, 89
preVarsInSynapseDynamics_sparse, 89
initGeNN
modelSpec.cc, 168
modelSpec.h, 172
initI
helper.h, 140
initLearnGrps
NNmodel, 74
initexpHH
helper.h, 140
initialiseInputData
Schmuker2014_classifier, 98
initialiseWeights_DENSE_PN_AN
Schmuker2014_classifier, 98
initialiseWeights_SPARSE_RN_PN
Schmuker2014_classifier, 98
initialiseWeights_WTA_AN_AN
Schmuker2014_classifier, 98
initialiseWeights_WTA_PN_PN
Schmuker2014_classifier, 98
initializeAllVars
classIzh, 49
inpIzh1
OneComp.cc, 174
input1
classIzh, 49
neuronpop, 65
input2
classIzh, 49
input_ini
SynDelay.cc, 199
input_p
SynDelay.cc, 200
InputBaseRate
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
PoissonIzh_sim.h, 180
inputInter_ini
SynDelay.cc, 200
inputOutput_ini
SynDelay.cc, 200
inputRates
Schmuker2014_classifier, 99
inputRatesSize
Schmuker2014_classifier, 99
inputSpec, 62
baseV, 62
N, 62
st, 62
t, 62
V, 62
inter_ini
SynDelay.cc, 200
inter_p
SynDelay.cc, 200
interOutput_ini
SynDelay.cc, 200
ip0
rulkovdp, 94
ip1
rulkovdp, 94
ip2
rulkovdp, 94
isGrpVarNeeded
generateKernels.cc, 135
Izh_sim_sparse.cu, 143
main, 143
Izh_sparse.cc, 143
DT, 144
excIzh_p, 144
inhIzh_p, 144
IzhExc_ini, 144
IzhInh_ini, 144
meanInpExc, 144
meanInpInh, 145
modelDefinition, 144
neuronPSize, 145
neuronVSize, 145
postExpP, 145
postSynV, 145
SynIzh_ini, 145
SynIzh_p, 145
synapsePSize, 145
Izh_sparse_model.cc, 145
_IZH_SPARSE_MODEL_CC_, 145
R, 145
RG, 145
Izh_sparse_model.h, 145
Izh_sparse_project/generate_run.cc
main, 127
openFileGetMax, 127
tS, 127
toLower, 127
toString, 127
toUpper, 127
Izh_sparse_sim.h, 146
DBG_SIZE, 146
iT, 146
t, 146
T_REPORT_TME, 146
TOTAL_TME, 146

timer, 146
IzhExc_ini
Izh_sparse.cc, 144
IzhInh_ini
Izh_sparse.cc, 144

LEARN1SYNAPSE
modelSpec.h, 173
LEARNING
modelSpec.h, 172
learnBlkSz
global.h, 138
lib/include/extra_postsynapses.h
clear, 116
postSynDecay, 116
postSyntoCurrent, 116
push_back, 116
lim0
pwSTDP, 90
pwSTDP_userdef, 91
lim1
pwSTDP, 90
pwSTDP_userdef, 91
limit
helper.h, 140
load_VR_data
Schmuker2014_classifier, 98
loadArrayFromTextFile
Schmuker2014_classifier, 98
loadClassLabels
Schmuker2014_classifier, 98
log
Schmuker2014_classifier, 99
logUIntSz
global.h, 138
lrnGroups
NNmodel, 76
lrnSynGrp
NNmodel, 76

MAPNEURON
modelSpec.h, 173
MATINP
modelSpec.h, 172
MAX_FIRING_RATE_HZ
experiment.h, 114
MAX_WEIGHT_PN_AN
experiment.h, 114
MAXNRN
modelSpec.h, 172
MAXPOSTSYN
modelSpec.h, 172
MBody1.cc, 148, 151
MBody1_project/generate_run.cc
main, 128
tS, 127
toLower, 128
toString, 128
toUpper, 128
MBody1_project/model/MBody1.cc
DT, 152
modelDefinition, 152
myDNDN_ini, 152
myDNDN_p, 152
myKCDN_ini, 153
myKCDN_p, 153
myLHIKC_ini, 153
myLHIKC_p, 153
myPNKC_ini, 153
myPNKC_p, 153
myPNLHI_ini, 153
myPNLHI_p, 153
myPOI_ini, 154
myPOI_p, 154
postExpDNDN, 154
postExpKCDN, 154
postExpLHIKC, 154
postExpPNKC, 154
postExpPNLHI, 154
postSynV, 154
stdTM_ini, 155
stdTM_p, 155
MBody1_project/model/classol_sim.cu
main, 104
MBody1_project/model/classol_sim.h
DBG_SIZE, 106
iT, 106
InputBaseRate, 106
MYRAND, 106
PAT_TIME, 106
PATFTIME, 106
PATTERNNO, 106
patFireTime, 106
patSetTime, 106
SYN_OUT_TME, 106
t, 107
T_REPORT_TME, 106
TOTAL_TME, 106
timer, 107
MBody1_project/model/map_classol.cc
_MAP_CLASSOL_CC_, 146
MBody_delayedSyn.cc, 155
DT, 156
modelDefinition, 156
myDNDN_ini, 156
myDNDN_p, 156
myKCDN_ini, 156
myKCDN_p, 156
myLHIKC_ini, 157
myLHIKC_p, 157
myPNKC_ini, 157
myPNKC_p, 157
myPNLHI_ini, 157
myPNLHI_p, 157
myPOI_ini, 157
myPOI_p, 157
postExpDNDN, 158

postExpKCDN, 158
postExpLHIKC, 158
postExpPNKC, 158
postExpPNLHI, 158
postSynV, 158
stdTM_ini, 158
stdTM_p, 158
MBody_delayedSyn_project/generate_run.cc
main, 128
tS, 128
toLower, 128
toString, 129
toUpper, 129
MBody_delayedSyn_project/model/classol_sim.cu
main, 105
MBody_delayedSyn_project/model/classol_sim.h
DBG_SIZE, 107
iT, 108
InputBaseRate, 107
MYRAND, 107
PAT_TIME, 107
PATFTIME, 107
PATTERNNO, 107
patFireTime, 108
patSetTime, 108
SYN_OUT_TME, 107
t, 108
T_REPORT_TME, 107
TOTAL_TME, 107
timer, 108
MBody_delayedSyn_project/model/map_classol.cc
_MAP_CLASSOL_CC_, 147
MBody_individualID.cc, 159
DT, 160
modelDefinition, 160
myDNDN_ini, 160
myDNDN_p, 160
myKCDN_ini, 160
myKCDN_p, 160
myLHIKC_ini, 160
myLHIKC_p, 161
myPNKC_ini, 161
myPNKC_p, 161
myPNLHI_ini, 161
myPNLHI_p, 161
myPOI_ini, 161
myPOI_p, 161
postExpDNDN, 161
postExpKCDN, 161
postExpLHIKC, 162
postExpPNKC, 162
postExpPNLHI, 162
postSynV, 162
stdTM_ini, 162
stdTM_p, 162
MBody_individualID_project/generate_run.cc
main, 129
tS, 129
toLower, 129
toString, 129
toUpper, 129
MBody_individualID_project/model/classol_sim.cu
main, 105
MBody_individualID_project/model/classol_sim.h
DBG_SIZE, 108
iT, 109
InputBaseRate, 109
MYRAND, 108
PAT_TIME, 108
PATFTIME, 108
PATTERNNO, 108
patFireTime, 109
patSetTime, 109
SYN_OUT_TME, 108
t, 109
T_REPORT_TME, 108
TOTAL_TME, 109
timer, 109
MBody_individualID_project/model/map_classol.cc
_MAP_CLASSOL_CC_, 147
MBody_userdef.cc, 163
DT, 164
gpKCDN, 164
gpPNKC, 164
modelDefinition, 164
myDNDN_ini, 164
myDNDN_p, 164
myKCDN_ini, 164
myKCDN_p, 164
myLHIKC_ini, 165
myLHIKC_p, 165
myPNKC_ini, 165
myPNKC_p, 165
myPNLHI_ini, 165
myPNLHI_p, 165
myPOI_ini, 165
myPOI_p, 165
postExpDNDN, 165
postExpKCDN, 166
postExpLHIKC, 166
postExpPNKC, 166
postExpPNLHI, 166
postSynV, 166
postSynV_EXPDECAY_EVAR, 166
stdTM_ini, 166
stdTM_p, 166
TIMING, 164
MBody_userdef_project/generate_run.cc
main, 130
tS, 130
toLower, 130
toString, 130
toUpper, 130
MBody_userdef_project/model/classol_sim.cu
main, 105
MBody_userdef_project/model/classol_sim.h

DBG_SIZE, 109
iT, 110
InputBaseRate, 110
MYRAND, 109
PAT_TIME, 109
PATFTIME, 109
PATTERNNO, 110
patFireTime, 110
patSetTime, 110
SYN_OUT_TME, 110
t, 110
T_REPORT_TME, 110
TOTAL_TME, 110
timer, 110
MBody_userdef_project/model/map_classol.cc
_MAP_CLASSOL_CC_, 147
MIN_FIRING_RATE_HZ
experiment.h, 114
MIN_WEIGHT_PN_AN
experiment.h, 114
MYRAND
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 109
PoissonIzh_sim.h, 180
main
experiment.cu, 112
gen_input_structured.cc, 120
gen_kcdn_syns.cc, 120
gen_kcdn_syns_fixto10K.cc, 121
gen_pnkc_syns.cc, 122
gen_pnkc_syns_indivID.cc, 122
gen_pnlhi_syns.cc, 123
gen_syns_sparse.cc, 123
gen_syns_sparse_izhModel.cc, 125
generateALL.cc, 133
HHVclampGA_project/generate_run.cc, 126
Izh_sim_sparse.cu, 143
Izh_sparse_project/generate_run.cc, 127
MBody1_project/generate_run.cc, 128
MBody1_project/model/classol_sim.cu, 104
MBody_delayedSyn_project/generate_run.cc, 128
MBody_delayedSyn_project/model/classol_sim.cu, 105
MBody_individualID_project/generate_run.cc, 129
MBody_individualID_project/model/classol_sim.cu, 105
MBody_userdef_project/generate_run.cc, 130
MBody_userdef_project/model/classol_sim.cu, 105
OneComp_project/generate_run.cc, 131
OneComp_sim.cu, 176
PoissonIzh_project/generate_run.cc, 131
PoissonIzh_sim.cu, 179
SynDelaySim.cu, 202
testPostVarsInPostLearn.cu, 203
testPostVarsInPostLearn_sparse.cu, 204
testPostVarsInSimCode.cu, 205
testPostVarsInSimCode_sparse.cu, 206
testPostVarsInSynapseDynamics.cu, 207
testPostVarsInSynapseDynamics_sparse.cu, 208
testPreVarsInPostLearn.cu, 209
testPreVarsInPostLearn_sparse.cu, 210
testPreVarsInSimCode.cu, 211
testPreVarsInSimCode_sparse.cu, 212
testPreVarsInSynapseDynamics.cu, 213
testPreVarsInSynapseDynamics_sparse.cu, 214
VClampGA.cu, 218
map_classol.cc, 146, 147
map_classol.h, 148
maxConn
NNmodel, 76
meanInpExc
Izh_sparse.cc, 144
meanInpInh
Izh_sparse.cc, 145
mexp
helper.h, 140
model
classIzh, 49
classol, 58
neuronpop, 65
Schmuker2014_classifier, 99
Model_Schmuker_2014_classifier.cc, 167
CLUST_SIZE_AN, 167
CLUST_SIZE_PN, 167
CLUST_SIZE_RN, 167
DT, 167
modelDefinition, 168
NETWORK_SCALE, 167
NUM_CLASSES, 167
NUM_FEATURES, 167
NUM_VR, 167
SYNAPSE_TAU_ANAN, 167
SYNAPSE_TAU_PNAN, 167
SYNAPSE_TAU_PNPN, 168
SYNAPSE_TAU_RNPN, 168
modelDefinition
HHVClamp.cc, 141
HHVclampGA_project/model/MBody1.cc, 149
Izh_sparse.cc, 144
MBody1_project/model/MBody1.cc, 152
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
Model_Schmuker_2014_classifier.cc, 168
OneComp.cc, 174
PoissonIzh.cc, 177
postVarsInPostLearn.cc, 180
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186

preVarsInPostLearn.cc, 187
preVarsInPostLearn_sparse.cc, 188
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
SynDelay.cc, 199
modelSpec.cc, 168
_MODELSPEC_CC_, 168
initGeNN, 168
modelSpec.h, 168
_MODELSPEC_H_, 171
ALLTOALL, 171
AUTODEVICE, 171
CONSTINP, 171
CPU, 171
DENSE, 171
EXITSYN, 171
EXPDECAY, 173
FALSE, 171
GENN_DOUBLE, 171
GENN_FLOAT, 171
GLOBALG, 171
GPU, 171
GeNNReady, 173
INDIVIDUALG, 171
INDIVIDUALID, 171
INHIBSYN, 172
INPRULE, 172
IZHIKEVICH, 173
IZHIKEVICH_PS, 173
IZHIKEVICH_V, 173
initGeNN, 172
LEARN1SYNAPSE, 173
LEARNING, 172
MAPNEURON, 173
MATINP, 172
MAXNRN, 172
MAXPOSTSYN, 172
NGRADSYNAPSE, 173
NO_DELAY, 172
NOINP, 172
NOLEARNING, 172
NSYNAPSE, 173
POISSONNEURON, 173
RANDNINP, 172
SPARSE, 172
SPIKESOURCE, 173
SPK_THRESH_STDP, 172
SYNTYPENO, 172
TRAUBMILES, 173
TRAUBMILES_ALTERNATIVE, 173
TRAUBMILES_FAST, 173
TRAUBMILES_PSTEP, 173
TRAUBMILES_SAFE, 173
TRUE, 172
myDNDN_ini
MBody1_project/model/MBody1.cc, 152
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
myDNDN_p
HHVclampGA_project/model/MBody1.cc, 149
MBody1_project/model/MBody1.cc, 152
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
myHH_ini
HHVClamp.cc, 142
myHH_p
HHVClamp.cc, 142
myKCDN_ini
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
myKCDN_p
HHVclampGA_project/model/MBody1.cc, 149
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 156
MBody_individualID.cc, 160
MBody_userdef.cc, 164
myLHIKC_ini
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 160
MBody_userdef.cc, 165
myLHIKC_p
HHVclampGA_project/model/MBody1.cc, 149
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
myPNKC_ini
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
myPNKC_p
HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
myPNLHI_ini
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
myPNLHI_p
HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 153
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
myPOI_ini

HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
PoissonIzh.cc, 178
myPOI_p
HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 157
MBody_individualID.cc, 161
MBody_userdef.cc, 165
PoissonIzh.cc, 178
mySyn_ini
PoissonIzh.cc, 178
mySyn_p
OneComp.cc, 175
PoissonIzh.cc, 178

N
inputSpec, 62
n
randomGauss, 92
randomGen, 93
stdRG, 101
N_FOLDING
experiment.h, 114
NETWORK_SCALE
Model_Schmuker_2014_classifier.cc, 167
NGRADSYNAPSE
modelSpec.h, 173
nModels
utils.h, 218
NNmodel, 66
∼NNmodel, 69
activateDirectInput, 69
addNeuronPopulation, 69, 70
addSynapsePopulation, 70, 72, 73
checkSizes, 74
chooseGPUDevice, 75
dnp, 75
dpsp, 75
dsp_w, 75
final, 76
finalize, 74
ftype, 76
globalInp, 76
inSyn, 76
initLearnGrps, 74
lrnGroups, 76
lrnSynGrp, 76
maxConn, 76
NNmodel, 69
name, 76
needSt, 76
needSynapseDelay, 76
neuronDelaySlots, 76
neuronDeviceID, 76
neuronGrpN, 76
neuronHostID, 76
neuronIni, 76
neuronN, 77
neuronName, 77
neuronNeedSpkEvnt, 77
neuronNeedSt, 77
neuronNeedTrueSpk, 77
neuronPara, 77
neuronPostSyn, 77
neuronSpkEvntCondition, 77
neuronType, 77
neuronVarNeedQueue, 77
outSyn, 77
padSumLearnN, 77
padSumNeuronN, 77
padSumSynapseKrnl, 77
postSynIni, 78
postSynapsePara, 77
postSynapseType, 77
RNtype, 78
receivesInputCurrent, 78
seed, 78
setConstInp, 74
setGPUDevice, 74
setMaxConn, 74
setName, 74
setNeuronClusterIndex, 74
setPrecision, 75
setSeed, 75
setSynapseClusterIndex, 75
setSynapseG, 75
setTiming, 75
sumNeuronN, 78
synapseConnType, 78
synapseDelay, 78
synapseDeviceID, 78
synapseGType, 78
synapseGrpN, 78
synapseHostID, 78
synapseInSynNo, 78
synapseIni, 78
synapseName, 78
synapseOutSynNo, 78
synapsePara, 78
synapseSource, 79
synapseSpkEvntVars, 79
synapseTarget, 79
synapseType, 79
synapseUsesPostLearning, 79
synapseUsesSpikeEvents, 79
synapseUsesTrueSpikes, 79
timing, 79
NO_DELAY
modelSpec.h, 172
NOINP
modelSpec.h, 172
NOLEARNING
modelSpec.h, 172

NPOP
HHVClampParameters.h, 142
NSYNAPSE
modelSpec.h, 173
NUM_CLASSES
Model_Schmuker_2014_classifier.cc, 167
NUM_FEATURES
Model_Schmuker_2014_classifier.cc, 167
NUM_VR
Model_Schmuker_2014_classifier.cc, 167
name
NNmodel, 76
Parameter, 79
name_substitutions
stringutils.h, 198
needPostSt
neuronModel, 63
weightUpdateModel, 103
needPreSt
neuronModel, 63
weightUpdateModel, 103
needSt
NNmodel, 76
needSynapseDelay
NNmodel, 76
neuron_ini
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 187
preVarsInPostLearn_sparse.cc, 188
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
neuron_p
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 187
preVarsInPostLearn_sparse.cc, 189
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
neuron_p2
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
preVarsInPostLearn.cc, 187
preVarsInPostLearn_sparse.cc, 189
neuronBlkSz
global.h, 138
neuronDelaySlots
NNmodel, 76
neuronDeviceID
NNmodel, 76
neuronGrpN
NNmodel, 76
neuronHostID
NNmodel, 76
neuronIni
NNmodel, 76
neuronModel, 62
dpNames, 63
dps, 63
extraGlobalNeuronKernelParameterTypes, 63
extraGlobalNeuronKernelParameters, 63
needPostSt, 63
needPreSt, 63
pNames, 63
resetCode, 63
simCode, 63
thresholdConditionCode, 64
tmpVarNames, 64
tmpVarTypes, 64
varNames, 64
varTypes, 64
neuronN
NNmodel, 77
neuronName
NNmodel, 77
neuronNeedSpkEvnt
NNmodel, 77
neuronNeedSt
NNmodel, 77
neuronNeedTrueSpk
NNmodel, 77
neuronPSize
Izh_sparse.cc, 145
neuronPara
NNmodel, 77
neuronPostSyn
NNmodel, 77
neuronSpkEvntCondition
NNmodel, 77
neuronType
NNmodel, 77
neuronVSize
Izh_sparse.cc, 145
neuronVarNeedQueue
NNmodel, 77
neuronpop, 64
∼neuronpop, 65
allocate_device_mem_input, 65
allocate_device_mem_patterns, 65
copy_device_mem_input, 65
create_input_values, 65
d_input1, 65
getSpikeNumbersFromGPU, 65
getSpikesFromGPU, 65

init, 65
input1, 65
model, 65
neuronpop, 65
output_spikes, 65
output_state, 65
read_input_values, 65
run, 65
sum_spikes, 65
sumIzh1, 65
write_input_to_file, 65
nexp
helper.h, 140
nlong
stdRG, 101

OB
CodeHelper.cc, 110
OUTPUT_DIR
experiment.h, 114
off0
pwSTDP, 90
pwSTDP_userdef, 91
off1
pwSTDP, 90
pwSTDP_userdef, 91
off2
pwSTDP, 90
pwSTDP_userdef, 91
offset
classol, 58
OneComp.cc, 174
DT, 174
exIzh_ini, 174
exIzh_p, 174
inpIzh1, 174
modelDefinition, 174
mySyn_p, 175
postExp, 175
postSynV, 175
OneComp_model.cc, 175
_ONECOMP_MODEL_CC_, 175
OneComp_model.h, 175
OneComp_project/generate_run.cc
main, 131
toString, 131
OneComp_sim.cu, 175
main, 176
OneComp_sim.h, 176
DBG_SIZE, 176
iT, 176
t, 176
T_REPORT_TME, 176
TOTAL_TME, 176
timer, 176
op
ensureFtype.h, 111
stringutils.h, 199
openBrace
CodeHelper, 59
openFileGetMax
Izh_sparse_project/generate_run.cc, 127
openRecordingFile
Schmuker2014_classifier, 98
operator<<
helper.h, 140
optimiseBlockSize
global.h, 138
outSyn
NNmodel, 77
output_ini
SynDelay.cc, 200
output_p
SynDelay.cc, 201
output_params
classIzh, 49
output_spikes
classIzh, 49
classol, 54, 55
neuronpop, 65
output_state
classIzh, 49
classol, 55
neuronpop, 65
outputDir
Schmuker2014_classifier, 99
outputRunParameters
experiment.cu, 112
outputSpikes
Schmuker2014_classifier, 98
overallWinnerSpikeCountAN
Schmuker2014_classifier, 99

p_pattern
classol, 58
PAT_TIME
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 109
PATFTIME
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 109
PATTERNNO
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 110
PoissonIzh_sim.h, 180
PLASTICITY_INTERVAL_MS

experiment.h, 114
pNames
neuronModel, 63
postSynModel, 80
weightUpdateModel, 103
POISSONNEURON
modelSpec.h, 173
padSumLearnN
NNmodel, 77
padSumNeuronN
NNmodel, 77
padSumSynapseKrnl
NNmodel, 77
param_CONNECTIVITY_AN_AN
Schmuker2014_classifier, 99
param_CONNECTIVITY_PN_AN
Schmuker2014_classifier, 99
param_CONNECTIVITY_PN_PN
Schmuker2014_classifier, 99
param_CONNECTIVITY_RN_PN
Schmuker2014_classifier, 99
param_GLOBAL_WEIGHT_SCALING
Schmuker2014_classifier, 99
param_MAX_FIRING_RATE_HZ
Schmuker2014_classifier, 99
param_MAX_WEIGHT_PN_AN
Schmuker2014_classifier, 99
param_MIN_FIRING_RATE_HZ
Schmuker2014_classifier, 99
param_MIN_WEIGHT_PN_AN
Schmuker2014_classifier, 99
param_PLASTICITY_INTERVAL_MS
Schmuker2014_classifier, 99
param_SPIKING_ACTIVITY_THRESHOLD_HZ
Schmuker2014_classifier, 99
param_WEIGHT_DELTA_PN_AN
Schmuker2014_classifier, 99
param_WEIGHT_RN_PN
Schmuker2014_classifier, 100
param_WEIGHT_WTA_AN_AN
Schmuker2014_classifier, 100
param_WEIGHT_WTA_PN_PN
Schmuker2014_classifier, 100
Parameter, 79
experiment.cu, 112
name, 79
value, 79
patFireTime
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 108
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
patSetTime
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 108
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
pattern
classol, 58
plasticWeights
Schmuker2014_classifier, 100
PoissonIzh-model.cc, 176
_POISSONIZHMODEL_CC_, 177
PoissonIzh-model.h, 177
PoissonIzh.cc, 177
_FTYPE, 177
DT, 177
exIzh_ini, 178
exIzh_p, 178
modelDefinition, 177
myPOI_ini, 178
myPOI_p, 178
mySyn_ini, 178
mySyn_p, 178
postExp, 178
postSynV, 179
SCALAR_MAX, 177
SCALAR_MIN, 177
scalar, 177
PoissonIzh_project/generate_run.cc
main, 131
toString, 131
PoissonIzh_sim.cu, 179
main, 179
PoissonIzh_sim.h, 179
DBG_SIZE, 180
INJECTCURRENT, 180
iT, 180
InputBaseRate, 180
MYRAND, 180
PATTERNNO, 180
SYN_OUT_TME, 180
t, 180
T_REPORT_TME, 180
TOTAL_TME, 180
timer, 180
populateDeviceMemory
Schmuker2014_classifier, 98
postExp
OneComp.cc, 175
PoissonIzh.cc, 178
postExpDNDN
HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 161
MBody_userdef.cc, 165
postExpInp
SynDelay.cc, 201
postExpInt
SynDelay.cc, 201
postExpKCDN

HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 161
MBody_userdef.cc, 166
postExpLHIKC
HHVclampGA_project/model/MBody1.cc, 150
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
postExpOut
SynDelay.cc, 201
postExpP
Izh_sparse.cc, 145
postExpPNKC
HHVclampGA_project/model/MBody1.cc, 151
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
postExpPNLHI
HHVclampGA_project/model/MBody1.cc, 151
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
postSyn_ini
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 188
preVarsInPostLearn_sparse.cc, 189
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
postSyn_p
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 188
preVarsInPostLearn_sparse.cc, 189
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
postSynDecay
lib/include/extra_postsynapses.h, 116
postSynModel, 80
SpineML_to_GeNN/extra_postsynapses.h, 117
postSynIni
NNmodel, 78
postSynModel, 80
dpNames, 80
dps, 80
pNames, 80
postSynDecay, 80
postSyntoCurrent, 80
varNames, 80
varTypes, 80
postSynModels
utils.h, 218
postSynV
HHVclampGA_project/model/MBody1.cc, 151
Izh_sparse.cc, 145
MBody1_project/model/MBody1.cc, 154
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
OneComp.cc, 175
PoissonIzh.cc, 179
SynDelay.cc, 201
postSynV_EXPDECAY_EVAR
MBody_userdef.cc, 166
postSynapsePara
NNmodel, 77
postSynapseType
NNmodel, 77
postSyntoCurrent
lib/include/extra_postsynapses.h, 116
postSynModel, 80
SpineML_to_GeNN/extra_postsynapses.h, 117
postVarsInPostLearn, 81
~postVarsInPostLearn, 81
init_neurons, 81
init_synapses, 81
postVarsInPostLearn, 81
run, 81
theW, 81
postVarsInPostLearn.cc, 180
DT, 180
modelDefinition, 180
neuron_ini, 181
neuron_p, 181
neuron_p2, 181
postSyn_ini, 181
postSyn_p, 181
synapses_ini, 181
synapses_p, 181
postVarsInPostLearn_sparse, 81
~postVarsInPostLearn_sparse, 82
init_neurons, 82
init_synapses, 82
postVarsInPostLearn_sparse, 82
run, 82
theW, 82
postVarsInPostLearn_sparse.cc, 181
DT, 182
modelDefinition, 182

neuron_ini, 182
neuron_p, 182
neuron_p2, 182
postSyn_ini, 182
postSyn_p, 182
synapses_ini, 182
synapses_p, 182
postVarsInSimCode, 82
~postVarsInSimCode, 83
init_neurons, 83
init_synapses, 83
postVarsInSimCode, 83
run, 83
theW, 83
postVarsInSimCode.cc, 183
DT, 183
modelDefinition, 183
neuron_ini, 183
neuron_p, 183
postSyn_ini, 183
postSyn_p, 183
synapses_ini, 183
synapses_p, 183
postVarsInSimCode_sparse, 83
~postVarsInSimCode_sparse, 83
init_neurons, 83
init_synapses, 83
postVarsInSimCode_sparse, 83
run, 83
theW, 83
postVarsInSimCode_sparse.cc, 184
DT, 184
modelDefinition, 184
neuron_ini, 184
neuron_p, 184
postSyn_ini, 184
postSyn_p, 184
synapses_ini, 184
synapses_p, 184
postVarsInSynapseDynamics, 84
~postVarsInSynapseDynamics, 84
init_neurons, 84
init_synapses, 84
postVarsInSynapseDynamics, 84
run, 84
theW, 84
postVarsInSynapseDynamics.cc, 185
DT, 185
modelDefinition, 185
neuron_ini, 185
neuron_p, 185
postSyn_ini, 185
postSyn_p, 185
synapses_ini, 185
synapses_p, 185
postVarsInSynapseDynamics_sparse, 84
~postVarsInSynapseDynamics_sparse, 85
init_neurons, 85
init_synapses, 85
postVarsInSynapseDynamics_sparse, 85
run, 85
theW, 85
postVarsInSynapseDynamics_sparse.cc, 186
DT, 186
modelDefinition, 186
neuron_ini, 186
neuron_p, 186
postSyn_ini, 186
postSyn_p, 186
synapses_ini, 186
synapses_p, 186
postsynapticVariables/testHelper.h
absDiff, 202
preVarsInPostLearn, 85
~preVarsInPostLearn, 86
init_neurons, 86
init_synapses, 86
preVarsInPostLearn, 86
run, 86
theW, 86
preVarsInPostLearn.cc, 187
DT, 187
modelDefinition, 187
neuron_ini, 187
neuron_p, 187
neuron_p2, 187
postSyn_ini, 188
postSyn_p, 188
synapses_ini, 188
synapses_p, 188
preVarsInPostLearn_sparse, 86
~preVarsInPostLearn_sparse, 86
init_neurons, 86
init_synapses, 86
preVarsInPostLearn_sparse, 86
run, 86
theW, 86
preVarsInPostLearn_sparse.cc, 188
DT, 188
modelDefinition, 188
neuron_ini, 188
neuron_p, 189
neuron_p2, 189
postSyn_ini, 189
postSyn_p, 189
synapses_ini, 189
synapses_p, 189
preVarsInSimCode, 87
~preVarsInSimCode, 87
init_neurons, 87
init_synapses, 87
preVarsInSimCode, 87
run, 87
theW, 87
preVarsInSimCode.cc, 189
DT, 190

modelDefinition, 190
neuron_ini, 190
neuron_p, 190
postSyn_ini, 190
postSyn_p, 190
synapses_ini, 190
synapses_p, 190
preVarsInSimCode_sparse, 87
~preVarsInSimCode_sparse, 88
init_neurons, 88
init_synapses, 88
preVarsInSimCode_sparse, 88
run, 88
theW, 88
preVarsInSimCode_sparse.cc, 190
DT, 191
modelDefinition, 191
neuron_ini, 191
neuron_p, 191
postSyn_ini, 191
postSyn_p, 191
synapses_ini, 191
synapses_p, 191
preVarsInSynapseDynamics, 88
~preVarsInSynapseDynamics, 89
init_neurons, 89
init_synapses, 89
preVarsInSynapseDynamics, 89
run, 89
theW, 89
preVarsInSynapseDynamics.cc, 191
DT, 192
modelDefinition, 192
neuron_ini, 192
neuron_p, 192
postSyn_ini, 192
postSyn_p, 192
synapses_ini, 192
synapses_p, 192
preVarsInSynapseDynamics_sparse, 89
~preVarsInSynapseDynamics_sparse, 89
init_neurons, 89
init_synapses, 89
preVarsInSynapseDynamics_sparse, 89
run, 89
theW, 89
preVarsInSynapseDynamics_sparse.cc, 192
DT, 193
modelDefinition, 193
neuron_ini, 193
neuron_p, 193
postSyn_ini, 193
postSyn_p, 193
synapses_ini, 193
synapses_p, 193
preparePostSynModels
utils.h, 217
prepareStandardModels
utils.h, 217
prepareWeightUpdateModels
utils.h, 217
presynapticVariables/testHelper.h
absDiff, 203
printSeparator
Schmuker2014_classifier, 98
printTextFile
experiment.cu, 112
printVector
gen_syns_sparse_izhModel.cc, 125
procreatePop
GA.cc, 118
ps
SpineML_to_GeNN/extra_postsynapses.h, 118
push_back
extra_neurons.h, 115
lib/include/extra_postsynapses.h, 116
SpineML_to_GeNN/extra_postsynapses.h, 117
pwSTDP, 90
calculateDerivedParameter, 90
lim0, 90
lim1, 90
off0, 90
off1, 90
off2, 90
slope0, 90
slope1, 90
pwSTDP_userdef, 91
calculateDerivedParameter, 91
lim0, 91
lim1, 91
off0, 91
off1, 91
off2, 91
slope0, 91
slope1, 91

R
gen_input_structured.cc, 120
gen_kcdn_syns.cc, 120
gen_kcdn_syns_fixto10K.cc, 121
gen_pnkc_syns.cc, 122
gen_pnkc_syns_indivID.cc, 122
gen_syns_sparse.cc, 123
gen_syns_sparse_izhModel.cc, 125
Izh_sparse_model.cc, 145
VClampGA.h, 219
R2
gen_kcdn_syns_fixto10K.cc, 121
RAND
experiment.cu, 112
VClampGA.h, 219
RANDNINP
modelSpec.h, 172
RANDOMGEN_CC
randomGen.cc, 193
RANDOMGEN_H
randomGen.h, 194

RECORDING_TIME_MS
experiment.h, 114
RECORDINGS_DIR
experiment.h, 114
REPEAT_LEARNING_SET
experiment.h, 114
REPORT_TIME
SynDelaySim.h, 202
testPostVarsInPostLearn.h, 203
testPostVarsInPostLearn_sparse.h, 204
testPostVarsInSimCode.h, 205
testPostVarsInSimCode_sparse.h, 206
testPostVarsInSynapseDynamics.h, 207
testPostVarsInSynapseDynamics_sparse.h, 208
testPreVarsInPostLearn.h, 209
testPreVarsInPostLearn_sparse.h, 210
testPreVarsInSimCode.h, 211
testPreVarsInSimCode_sparse.h, 212
testPreVarsInSynapseDynamics.h, 213
testPreVarsInSynapseDynamics_sparse.h, 214
RG
gen_kcdn_syns.cc, 120
gen_kcdn_syns_fixto10K.cc, 121
gen_pnkc_syns.cc, 122
gen_pnkc_syns_indivID.cc, 122
gen_syns_sparse.cc, 124
Izh_sparse_model.cc, 145
VClampGA.h, 219
RNtype
NNmodel, 78
randomEventOccurred
Schmuker2014_classifier, 98
randomGauss, 92
∼randomGauss, 92
n, 92
randomGauss, 92
srand, 92
randomGen, 93
∼randomGen, 93
n, 93
randomGen, 93
srand, 93
randomGen.cc, 193
RANDOMGEN_CC, 193
randomGen.h, 194
RANDOMGEN_H, 194
randomizeVar
classIzh, 49
randomizeVarSq
classIzh, 49
read_PNIzh1syns
classol, 55
read_input_patterns
classol, 55
read_input_values
classIzh, 49
neuronpop, 65
read_kcdnsyns
classol, 55
read_pnkcsyns
classol, 55, 56
read_pnlhisyns
classol, 56
read_sparsesyns_par
classIzh, 49
classol, 56
receivesInputCurrent
NNmodel, 78
recordingsDir
Schmuker2014_classifier, 100
resetClusterSpikeCountAN
Schmuker2014_classifier, 98
resetCode
neuronModel, 63
resetDevice
Schmuker2014_classifier, 98
resetIndividualSpikeCountPN
Schmuker2014_classifier, 98
resetOverallWinner
Schmuker2014_classifier, 98
Rind
gen_syns_sparse_izhModel.cc, 125
rulkovdp, 94
calculateDerivedParameter, 94
ip0, 94
ip1, 94
ip2, 94
run
classIzh, 49
classol, 56, 57
neuronpop, 65
postVarsInPostLearn, 81
postVarsInPostLearn_sparse, 82
postVarsInSimCode, 83
postVarsInSimCode_sparse, 83
postVarsInSynapseDynamics, 84
postVarsInSynapseDynamics_sparse, 85
preVarsInPostLearn, 86
preVarsInPostLearn_sparse, 86
preVarsInSimCode, 87
preVarsInSimCode_sparse, 88
preVarsInSynapseDynamics, 89
preVarsInSynapseDynamics_sparse, 89
Schmuker2014_classifier, 98
SynDelay, 101
runCPU
classol, 57
runGPU
classol, 57
runexpHH
helper.h, 140

S_ISDIR
experiment.cu, 112
SAVEP
CodeHelper.cc, 110
SCALAR_MAX

PoissonIzh.cc, 177
SCALAR_MIN
PoissonIzh.cc, 177
SIMPLEBIT_H
simpleBit.h, 196
SPARSE
modelSpec.h, 172
SPIKESOURCE
modelSpec.h, 173
SPIKING_ACTIVITY_THRESHOLD_HZ
experiment.h, 114
SPK_THRESH_STDP
modelSpec.h, 172
SYN_OUT_TME
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 110
PoissonIzh_sim.h, 180
SYNAPSE_TAU_ANAN
Model_Schmuker_2014_classifier.cc, 167
SYNAPSE_TAU_PNAN
Model_Schmuker_2014_classifier.cc, 167
SYNAPSE_TAU_PNPN
Model_Schmuker_2014_classifier.cc, 168
SYNAPSE_TAU_RNPN
Model_Schmuker_2014_classifier.cc, 168
SYNDELAYSIM_CU
SynDelaySim.cu, 202
SYNTYPENO
modelSpec.h, 172
sampleDistance
Schmuker2014_classifier, 100
scalar
PoissonIzh.cc, 177
Schmuker2014_classifier, 94
∼Schmuker2014_classifier, 97
addInputRate, 97
allocateHostAndDeviceMemory, 97
applyLearningRuleSynapses, 97
cacheDir, 99
calculateCurrentWindowWinner, 97
calculateOverallWinner, 97
calculateVrResponse, 97
calculateWinner, 97
checkContents, 97
classLabel, 99
clearDownDevice, 97
clearedDownDevice, 99
clusterSpikeCountAN, 99
convertToRateCode, 97
correctClass, 99
countAN, 99
countPN, 99
countPNAN, 99
countRN, 99
createWTAConnectivity, 97
d_inputRates, 99
d_maxRandomNumber, 99
data_type, 97
data_type_double, 97
data_type_float, 97
data_type_int, 97
data_type_uint, 97
datasetName, 99
findMaxMinSampleDistances, 97
generate_inputrates_dataset, 97
generate_or_load_inputrates_dataset, 97
generateSimulatedTimeSeriesData, 97
getClassCluster, 97
getClusterIndex, 97
getManhattanDistance, 97
getRand0to1, 97
getRecordingFilename, 98
getSampleDistance, 98
getSpikeNumbersFromGPU, 98
getSpikesFromGPU, 98
individualSpikeCountPN, 99
initialiseInputData, 98
initialiseWeights_DENSE_PN_AN, 98
initialiseWeights_SPARSE_RN_PN, 98
initialiseWeights_WTA_AN_AN, 98
initialiseWeights_WTA_PN_PN, 98
inputRates, 99
inputRatesSize, 99
load_VR_data, 98
loadArrayFromTextFile, 98
loadClassLabels, 98
log, 99
model, 99
openRecordingFile, 98
outputDir, 99
outputSpikes, 98
overallWinnerSpikeCountAN, 99
param_CONNECTIVITY_AN_AN, 99
param_CONNECTIVITY_PN_AN, 99
param_CONNECTIVITY_PN_PN, 99
param_CONNECTIVITY_RN_PN, 99
param_GLOBAL_WEIGHT_SCALING, 99
param_MAX_FIRING_RATE_HZ, 99
param_MAX_WEIGHT_PN_AN, 99
param_MIN_FIRING_RATE_HZ, 99
param_MIN_WEIGHT_PN_AN, 99
param_PLASTICITY_INTERVAL_MS, 99
param_SPIKING_ACTIVITY_THRESHOLD_HZ, 99
param_WEIGHT_DELTA_PN_AN, 99
param_WEIGHT_RN_PN, 100
param_WEIGHT_WTA_AN_AN, 100
param_WEIGHT_WTA_PN_PN, 100
plasticWeights, 100
populateDeviceMemory, 98
printSeparator, 98
randomEventOccurred, 98

recordingsDir, 100
resetClusterSpikeCountAN, 98
resetDevice, 98
resetIndividualSpikeCountPN, 98
resetOverallWinner, 98
run, 98
sampleDistance, 100
Schmuker2014_classifier, 97
setCorrectClass, 98
setMaxMinSampleDistances, 98
startLog, 98
t, 100
timestepsPerRecording, 100
uniqueRunId, 100
update_input_data_on_device, 98
updateClusterSpikeCountAN, 98
updateIndividualSpikeCountPN, 98
updateWeights_PN_AN, 98
updateWeights_PN_AN_on_device, 98
vrData, 100
winningClass, 100
Schmuker2014_classifier.cu, 194
_SCHMUKER2014_CLASSIFIER_, 195
Schmuker2014_classifier.h, 195
seed
NNmodel, 78
setB
simpleBit.h, 196
setConstInp
NNmodel, 74
setCorrectClass
Schmuker2014_classifier, 98
setDefaultParamValues
experiment.cu, 112
setGPUDevice
NNmodel, 74
setInput
classIzh, 49
setMaxConn
NNmodel, 74
setMaxMinSampleDistances
Schmuker2014_classifier, 98
setName
NNmodel, 74
setNeuronClusterIndex
NNmodel, 74
setPrecision
NNmodel, 75
setSeed
NNmodel, 75
setSparseConnectivityFromDense
sparseUtils.cc, 197
setSynapseClusterIndex
NNmodel, 75
setSynapseG
NNmodel, 75
setTiming
NNmodel, 75
setVerbose
CodeHelper, 59
setup_kernel
GeNNHelperKrnls.cu, 137
sigC
helper.h, 140
sigEK
helper.h, 140
sigENa
helper.h, 141
sigEl
helper.h, 140
sigGK
helper.h, 141
sigGNa
helper.h, 141
sigGl
helper.h, 141
simCode
extra_neurons.h, 115
neuronModel, 63
weightUpdateModel, 103
simCodeEvnt
weightUpdateModel, 103
simLearnPost
weightUpdateModel, 103
simpleBit.h, 195
B, 196
delB, 196
SIMPLEBIT_H, 196
setB, 196
single_var_init_fullrange
helper.h, 140
single_var_reinit
helper.h, 140
size_g
classol, 58
sizes.h, 196
_NC1, 196
slope0
pwSTDP, 90
pwSTDP_userdef, 91
slope1
pwSTDP, 90
pwSTDP_userdef, 91
sparse_utils_cc
sparseUtils.cc, 197
sparseUtils.cc, 196
countEntriesAbove, 197
createPosttoPreArray, 197
createSparseConnectivityFromDense, 197
getG, 197
getSparseVar, 197
setSparseConnectivityFromDense, 197
sparse_utils_cc, 197
strsearch, 197
SpineML_to_GeNN/extra_postsynapses.h
clear, 117

postSynDecay, 117
postSyntoCurrent, 117
ps, 118
push_back, 117
srand
randomGauss, 92
randomGen, 93
st
inputSpec, 62
start
stopWatch, 101
startLog
Schmuker2014_classifier, 98
startTimer
CStopWatch, 60
stdRG, 100
~stdRG, 100
n, 101
nlong, 101
stdRG, 100
stdTM_ini
HHVclampGA_project/model/MBody1.cc, 151
MBody1_project/model/MBody1.cc, 155
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
stdTM_p
HHVclampGA_project/model/MBody1.cc, 151
MBody1_project/model/MBody1.cc, 155
MBody_delayedSyn.cc, 158
MBody_individualID.cc, 162
MBody_userdef.cc, 166
stop
stopWatch, 101
stopTimer
CStopWatch, 60
stopWatch, 101
start, 101
stop, 101
stringutils.h, 197
__dnames, 199
__fnames, 199
__mathFN, 198
digits, 199
doFinal, 198
ensureFtype, 198
ensureMathFunctionFtype, 198
extended_name_substitutions, 198
extended_value_substitutions, 198
name_substitutions, 198
op, 199
substitute, 198
value_substitutions, 198
strsearch
sparseUtils.cc, 197
substitute
stringutils.h, 198
sum_spikes
classIzh, 49
classol, 57
neuronpop, 65
sumDN
classol, 58
sumIzh1
classol, 58
neuronpop, 65
sumKC
classol, 58
sumLHI
classol, 58
sumNeuronN
NNmodel, 78
sumPExc
classIzh, 49
sumPInh
classIzh, 49
sumPN
classol, 58
SynDelay, 101
~SynDelay, 101
run, 101
SynDelay, 101
SynDelay.cc, 199
constInput, 199
DT, 199
input_ini, 199
input_p, 200
inputInter_ini, 200
inputOutput_ini, 200
inter_ini, 200
inter_p, 200
interOutput_ini, 200
modelDefinition, 199
output_ini, 200
output_p, 201
postExpInp, 201
postExpInt, 201
postExpOut, 201
postSynV, 201
synapses_p, 201
SynDelaySim.cu, 201
main, 202
SYNDELAYSIM_CU, 202
SynDelaySim.h, 202
DT, 202
REPORT_TIME, 202
TOTAL_TIME, 202
SynIzh_ini
Izh_sparse.cc, 145
SynIzh_p
Izh_sparse.cc, 145
synapseBlkSz
global.h, 138
synapseConnType
NNmodel, 78
synapseDelay

NNmodel, 78
synapseDeviceID
NNmodel, 78
synapseDynamics
weightUpdateModel, 103
synapseGType
NNmodel, 78
synapseGrpN
NNmodel, 78
synapseHostID
NNmodel, 78
synapseInSynNo
NNmodel, 78
synapseIni
NNmodel, 78
synapseName
NNmodel, 78
synapseOutSynNo
NNmodel, 78
synapsePSize
Izh_sparse.cc, 145
synapsePara
NNmodel, 78
synapseSource
NNmodel, 79
synapseSpkEvntVars
NNmodel, 79
synapseTarget
NNmodel, 79
synapseType
NNmodel, 79
synapseUsesPostLearning
NNmodel, 79
synapseUsesSpikeEvents
NNmodel, 79
synapseUsesTrueSpikes
NNmodel, 79
synapses_ini
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 188
preVarsInPostLearn_sparse.cc, 189
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
synapses_p
postVarsInPostLearn.cc, 181
postVarsInPostLearn_sparse.cc, 182
postVarsInSimCode.cc, 183
postVarsInSimCode_sparse.cc, 184
postVarsInSynapseDynamics.cc, 185
postVarsInSynapseDynamics_sparse.cc, 186
preVarsInPostLearn.cc, 188
preVarsInPostLearn_sparse.cc, 189
preVarsInSimCode.cc, 190
preVarsInSimCode_sparse.cc, 191
preVarsInSynapseDynamics.cc, 192
preVarsInSynapseDynamics_sparse.cc, 193
SynDelay.cc, 201

t
inputSpec, 62
Izh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 107
MBody_delayedSyn_project/model/classol_sim.h, 108
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
Schmuker2014_classifier, 100
VClampGA.h, 219
T_REPORT_TME
Izh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 108
MBody_userdef_project/model/classol_sim.h, 110
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
TESTPREVARSINPOSTLEARN_SPARSE_CU
testPostVarsInPostLearn_sparse.cu, 204
testPreVarsInPostLearn_sparse.cu, 210
TESTPREVARSINSIMCODE_CU
testPostVarsInPostLearn.cu, 203
testPostVarsInSimCode.cu, 205
testPostVarsInSimCode_sparse.cu, 206
testPreVarsInPostLearn.cu, 209
testPreVarsInSimCode.cu, 211
testPreVarsInSimCode_sparse.cu, 212
TESTPREVARSINSYNAPSEDYNAMICS_CU
testPostVarsInSynapseDynamics.cu, 207
testPreVarsInSynapseDynamics.cu, 213
TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_↩ CU
testPostVarsInSynapseDynamics_sparse.cu, 208
testPreVarsInSynapseDynamics_sparse.cu, 214
TIMING
MBody_userdef.cc, 164
TOTAL_RECORDINGS
experiment.h, 114
TOTAL_TIME
SynDelaySim.h, 202
testPostVarsInPostLearn.h, 203
testPostVarsInPostLearn_sparse.h, 204
testPostVarsInSimCode.h, 205
testPostVarsInSimCode_sparse.h, 206
testPostVarsInSynapseDynamics.h, 207
testPostVarsInSynapseDynamics_sparse.h, 208

testPreVarsInPostLearn.h, 209
testPreVarsInPostLearn_sparse.h, 210
testPreVarsInSimCode.h, 211
testPreVarsInSimCode_sparse.h, 212
testPreVarsInSynapseDynamics.h, 213
testPreVarsInSynapseDynamics_sparse.h, 214
TOTAL_TME
lzh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 106
MBody_delayedSyn_project/model/classol_sim.h, 107
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
TOTALT
HHVClampParameters.h, 142
TRAUBMILES
modelSpec.h, 173
TRAUBMILES_ALTERNATIVE
modelSpec.h, 173
TRAUBMILES_FAST
modelSpec.h, 173
TRAUBMILES_PSTEP
modelSpec.h, 173
TRAUBMILES_SAFE
modelSpec.h, 173
TRUE
modelSpec.h, 172
tS
lzh_sparse_project/generate_run.cc, 127
MBody1_project/generate_run.cc, 127
MBody_delayedSyn_project/generate_run.cc, 128
MBody_individualID_project/generate_run.cc, 129
MBody_userdef_project/generate_run.cc, 130
toString.h, 215
testHelper.h, 202
testPostVarsInPostLearn.cu, 203
main, 203
TESTPREVARSINSIMCODE_CU, 203
testPostVarsInPostLearn.h, 203
DT, 203
REPORT_TIME, 203
TOTAL_TIME, 203
testPostVarsInPostLearn_sparse.cu, 204
main, 204
TESTPREVARSINPOSTLEARN_SPARSE_CU, 204
testPostVarsInPostLearn_sparse.h, 204
DT, 204
REPORT_TIME, 204
TOTAL_TIME, 204
testPostVarsInSimCode.cu, 205
main, 205
TESTPREVARSINSIMCODE_CU, 205
testPostVarsInSimCode.h, 205
DT, 205
REPORT_TIME, 205
TOTAL_TIME, 205
testPostVarsInSimCode_sparse.cu, 206
main, 206
TESTPREVARSINSIMCODE_CU, 206
testPostVarsInSimCode_sparse.h, 206
DT, 206
REPORT_TIME, 206
TOTAL_TIME, 206
testPostVarsInSynapseDynamics.cu, 207
main, 207
TESTPREVARSINSYNAPSEDYNAMICS_CU, 207
testPostVarsInSynapseDynamics.h, 207
DT, 207
REPORT_TIME, 207
TOTAL_TIME, 207
testPostVarsInSynapseDynamics_sparse.cu, 208
main, 208
TESTPREVARSINSYNAPSEDYNAMICS_SPARSE_CU, 208
testPostVarsInSynapseDynamics_sparse.h, 208
DT, 208
REPORT_TIME, 208
TOTAL_TIME, 208
testPreVarsInPostLearn.cu, 209
main, 209
TESTPREVARSINSIMCODE_CU, 209
testPreVarsInPostLearn.h, 209
DT, 209
REPORT_TIME, 209
TOTAL_TIME, 209
testPreVarsInPostLearn_sparse.cu, 210
main, 210
TESTPREVARSINPOSTLEARN_SPARSE_CU, 210
testPreVarsInPostLearn_sparse.h, 210
DT, 210
REPORT_TIME, 210
TOTAL_TIME, 210
testPreVarsInSimCode.cu, 211
main, 211
TESTPREVARSINSIMCODE_CU, 211
testPreVarsInSimCode.h, 211
DT, 211
REPORT_TIME, 211
TOTAL_TIME, 211
testPreVarsInSimCode_sparse.cu, 212
main, 212
TESTPREVARSINSIMCODE_CU, 212
testPreVarsInSimCode_sparse.h, 212
DT, 212
REPORT_TIME, 212
TOTAL_TIME, 212
testPreVarsInSynapseDynamics.cu, 213
main, 213
TESTPREVARSINSYNAPSEDYNAMICS_CU, 213
testPreVarsInSynapseDynamics.h, 213
DT, 213

REPORT_TIME, 213
TOTAL_TIME, 213
testPreVarsInSynapseDynamics_sparse.cu, 214
main, 214
TESTPREVARSINSYNAPSEDYNAMICS_SPAR↩
SE_CU, 214
testPreVarsInSynapseDynamics_sparse.h, 214
DT, 214
REPORT_TIME, 214
TOTAL_TIME, 214
theDev
global.h, 139
theRates
classol, 58
theSize
utils.h, 218
theW
postVarsInPostLearn, 81
postVarsInPostLearn_sparse, 82
postVarsInSimCode, 83
postVarsInSimCode_sparse, 83
postVarsInSynapseDynamics, 84
postVarsInSynapseDynamics_sparse, 85
preVarsInPostLearn, 86
preVarsInPostLearn_sparse, 86
preVarsInSimCode, 87
preVarsInSimCode_sparse, 88
preVarsInSynapseDynamics, 89
preVarsInSynapseDynamics_sparse, 89
thresholdConditionCode
neuronModel, 64
timer
experiment.h, 114
Izh_sparse_sim.h, 146
MBody1_project/model/classol_sim.h, 107
MBody_delayedSyn_project/model/classol_sim.h, 108
MBody_individualID_project/model/classol_sim.h, 109
MBody_userdef_project/model/classol_sim.h, 110
OneComp_sim.h, 176
PoissonIzh_sim.h, 180
VClampGA.h, 219
timestepsPerRecording
Schmuker2014_classifier, 100
timing
NNmodel, 79
tmpVarNames
neuronModel, 64
tmpVarTypes
neuronModel, 64
toLower
Izh_sparse_project/generate_run.cc, 127
MBody1_project/generate_run.cc, 128
MBody_delayedSyn_project/generate_run.cc, 128
MBody_individualID_project/generate_run.cc, 129
MBody_userdef_project/generate_run.cc, 130
toString
Izh_sparse_project/generate_run.cc, 127
MBody1_project/generate_run.cc, 128
MBody_delayedSyn_project/generate_run.cc, 129
MBody_individualID_project/generate_run.cc, 129
MBody_userdef_project/generate_run.cc, 130
OneComp_project/generate_run.cc, 131
PoissonIzh_project/generate_run.cc, 131
toString.h, 215
toString.h, 214
_TOSTRING_H_, 215
tS, 215
toString, 215
toUpper
Izh_sparse_project/generate_run.cc, 127
MBody1_project/generate_run.cc, 128
MBody_delayedSyn_project/generate_run.cc, 129
MBody_individualID_project/generate_run.cc, 129
MBody_userdef_project/generate_run.cc, 130
truevar_init
helper.h, 140
truevar_initexpHH
helper.h, 140

UINT
experiment.h, 114
UIntSz
global.h, 139
uniqueRunId
Schmuker2014_classifier, 100
update_input_data_on_device
Schmuker2014_classifier, 98
updateClusterSpikeCountAN
Schmuker2014_classifier, 98
updateIndividualSpikeCountPN
Schmuker2014_classifier, 98
updateWeights_PN_AN
Schmuker2014_classifier, 98
updateWeights_PN_AN_on_device
Schmuker2014_classifier, 98
utils.h, 215
_UTILS_H_, 217
CHECK_CUDA_ERRORS, 217
gennError, 217
nModels, 218
postSynModels, 218
preparePostSynModels, 217
prepareStandardModels, 217
prepareWeightUpdateModels, 217
theSize, 218
weightUpdateModels, 218
writeHeader, 218

V
inputSpec, 62
VClampGA.cu, 218
main, 218
VClampGA.h, 219
iT, 219
R, 219

RAND, 219
RG, 219
t, 219
timer, 219
VR_DATA_FILENAME
experiment.h, 114
value
Parameter, 79
value_substitutions
stringutils.h, 198
var_init_fullrange
helper.h, 140
var_reinit
helper.h, 140
varNames
neuronModel, 64
postSynModel, 80
weightUpdateModel, 103
varTypes
neuronModel, 64
postSynModel, 80
weightUpdateModel, 104
vectorContains
experiment.cu, 112
verbose
CodeHelper, 59
Vexp
helper.h, 141
vrData
Schmuker2014_classifier, 100

WEIGHT_DELTA_PN_AN
experiment.h, 114
WEIGHT_RN_PN
experiment.h, 114
WEIGHT_WTA_AN_AN
experiment.h, 114
WEIGHT_WTA_PN_PN
experiment.h, 114
weightUpdateModel, 102
dpNames, 103
dps, 103
evntThreshold, 103
extraGlobalSynapseKernelParameterTypes, 103
extraGlobalSynapseKernelParameters, 103
needPostSt, 103
needPreSt, 103
pNames, 103
simCode, 103
simCodeEvnt, 103
simLearnPost, 103
synapseDynamics, 103
varNames, 103
varTypes, 104
weightUpdateModel, 103
weightUpdateModels
utils.h, 218
winningClass
Schmuker2014_classifier, 100
write_input_to_file
classIzh, 49
neuronpop, 65
write_kcdnsyns
classol, 57
write_para
helper.h, 140
write_pnkcsyns
classol, 57, 58
write_pnlhisyns
classol, 58
writeHeader
utils.h, 218

xorwow_setup
GeNNHelperKrnls.cu, 137