SpiNNFrontEndCommonEnd Documentation

Jan 29, 2020

Contents

These pages document the python code for the SpiNNFrontEndCommon module which is part of the SpiNNaker Project.

This code depends on SpiNNUtils, SpiNNMachine, SpiNNStorageHandlers, SpiNNMan, PACMAN, DataSpecification (Combined_documentation).

Contents:

CHAPTER 1

spinn_front_end_common package

1.1 Subpackages

1.1.1 spinn_front_end_common.abstract_models package

1.1.1.1 Subpackages

spinn_front_end_common.abstract_models.impl package

Submodules

spinn_front_end_common.abstract_models.impl.machine_data_specable_vertex module

class `spinn_front_end_common.abstract_models.impl.machine_data_specable_vertex.`**MachineDataS**

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

generate_data_specification(**args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (`data_specification.DataSpecificationGenerator`) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

generate_machine_data_specification(*spec*, *placement*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps module

class spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps.**NeedsNMachine**

Bases: object

A class that uses the number of machine time steps

set_n_machine_time_steps (*arg*)

spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl module

class spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.**Provide**

Bases: *spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.AbstractProvidesKeyToAtomMapping*

routing_key_partition_atom_mapping (**args*, ***kwargs*)

returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

Module contents

class spinn_front_end_common.abstract_models.impl.**MachineDataSpecableVertex** (**args*, ***kwargs*)

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

generate_data_specification (**args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

generate_machine_data_specification (*spec*, *placement*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

class spinn_front_end_common.abstract_models.impl.**NeedsNMachineTimeSteps** (**args*, ***kwargs*)

Bases: object

A class that uses the number of machine time steps

set_n_machine_time_steps (*arg*)

class spinn_front_end_common.abstract_models.impl.**ProvidesKeyToAtomMappingImpl**

Bases: *spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.AbstractProvidesKeyToAtomMapping*

routing_key_partition_atom_mapping(**args*, ***kwargs*)
returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

1.1.1.2 Submodules

1.1.1.3 spinn_front_end_common.abstract_models.abstract_changable_after_run module

class spinn_front_end_common.abstract_models.abstract_changable_after_run.**AbstractChangable**
Bases: object

An item that can be changed after a call to run, the changes to which might or might not require mapping

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

1.1.1.4 spinn_front_end_common.abstract_models.abstract_generates_data_specification module

class spinn_front_end_common.abstract_models.abstract_generates_data_specification.**Abstract**
Bases: object

generate_data_specification(*spec*, *placement*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

1.1.1.5 spinn_front_end_common.abstract_models.abstract_has_associated_binary module

class spinn_front_end_common.abstract_models.abstract_has_associated_binary.**AbstractHasAsso**
Bases: object

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

1.1.1.6 spinn_front_end_common.abstract_models.abstract_machine_allocation_controller module

class spinn_front_end_common.abstract_models.abstract_machine_allocation_controller.**Abstrac**

Bases: object

An object that controls the allocation of a machine

close()

Indicate that the use of the machine is complete

extend_allocation(*new_total_run_time*)

Extend the allocation of the machine from the original run time

Parameters **new_total_run_time** – The total run time that is now required starting from when the machine was first allocated

1.1.1.7 spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constrai

Bases: object

A vertex that can provide constraints for its incoming edge partitions

get_incoming_partition_constraints(*partition*)

Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

1.1.1.8 spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping module

class spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.**Abstract**

Bases: object

interface to provide a mapping between routing key partitions and atom ids

routing_key_partition_atom_mapping(*routing_info*, *partition*)

returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

1.1.1.9 spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition module

class spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition.**Abstrac**

Bases: `object`

Allows a vertex to provide the number of keys for a partition of edges, rather than relying on the number of atoms in the pre-vertex

get_n_keys_for_partition (*partition*, *graph_mapper*)

Get the number of keys required by the given partition of edges

Parameters

- **partition** (`pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition`) – An partition that comes out of this vertex
- **graph_mapper** (`pacman.model.graph.graph_mapper.GraphMapper`) – A mapper between the graphs

Returns A list of constraints

Return type list of `pacman.model.constraints.abstract_constraint.AbstractConstraint`

1.1.1.10 spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constrai

Bases: `object`

A vertex that can provide constraints for its outgoing edge partitions

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of `pacman.model.constraints.abstract_constraint.AbstractConstraint`

1.1.1.11 spinn_front_end_common.abstract_models.abstract_recordable module

class spinn_front_end_common.abstract_models.abstract_recordable.**AbstractRecordable**

Bases: `object`

Indicates that an object might record some data in to SDRAM

is_recording ()

Deduce if the recorder is actually recording

1.1.1.12 spinn_front_end_common.abstract_models.abstract_rewrites_data_specification module

class spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.**Abstract**

Bases: `object`

Indicates an object that allows data to be changed after run, and so can rewrite the data specification

mark_regions_reloaded()
Indicate that the regions have been reloaded

regenerate_data_specification(*spec*, *placement*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

1.1.1.13 spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex module

class spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.**Abs**
Bases: object

A vertex which wants to commands to be sent to it as multicast packets at fixed points in the simulation

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

1.1.1.14 spinn_front_end_common.abstract_models.abstract_supports_database_injection module

class spinn_front_end_common.abstract_models.abstract_supports_database_injection.**AbstractS**
Bases: object

Marks a machine vertex as supporting injection of information via a database running on the controlling host.

is_in_injection_mode
Whether this vertex is actually in injection mode.

1.1.1.15 spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices module

class spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.**Abstra**
Bases: object

A vertex with a dependent vertices, which should be connected to this vertex by an edge directly to each of them

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

1.1.1.16 Module contents

class spinn_front_end_common.abstract_models.**AbstractChangableAfterRun**
Bases: object

An item that can be changed after a call to run, the changes to which might or might not require mapping

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

class spinn_front_end_common.abstract_models.**AbstractGeneratesDataSpecification**
Bases: object

generate_data_specification(*spec*, *placement*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

class spinn_front_end_common.abstract_models.**AbstractHasAssociatedBinary**
Bases: object

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

class spinn_front_end_common.abstract_models.**AbstractMachineAllocationController**
Bases: object

An object that controls the allocation of a machine

close()
Indicate that the use of the machine is complete

extend_allocation(*new_total_run_time*)
Extend the allocation of the machine from the original run time

Parameters new_total_run_time – The total run time that is now required starting from when the machine was first allocated

class spinn_front_end_common.abstract_models.**AbstractProvidesIncomingPartitionConstraints**
Bases: object

A vertex that can provide constraints for its incoming edge partitions

get_incoming_partition_constraints(*partition*)
Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

class spinn_front_end_common.abstract_models.**AbstractProvidesKeyToAtomMapping**

Bases: object

interface to provide a mapping between routing key partitions and atom ids

routing_key_partition_atom_mapping(*routing_info*, *partition*)

returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

class spinn_front_end_common.abstract_models.**AbstractProvidesNKeysForPartition**

Bases: object

Allows a vertex to provide the number of keys for a partition of edges, rather than relying on the number of atoms in the pre-vertex

get_n_keys_for_partition(*partition*, *graph_mapper*)

Get the number of keys required by the given partition of edges

Parameters

- **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that comes out of this vertex
- **graph_mapper** (pacman.model.graph.graph_mapper.GraphMapper) – A mapper between the graphs

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

class spinn_front_end_common.abstract_models.**AbstractProvidesOutgoingPartitionConstraints**

Bases: object

A vertex that can provide constraints for its outgoing edge partitions

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

class spinn_front_end_common.abstract_models.**AbstractRecordable**

Bases: object

Indicates that an object might record some data in to SDRAM

is_recording()
Deduce if the recorder is actually recording

class spinn_front_end_common.abstract_models.**AbstractRewritesDataSpecification**
Bases: `object`

Indicates an object that allows data to be changed after run, and so can rewrite the data specification

mark_regions_reloaded()
Indicate that the regions have been reloaded

regenerate_data_specification(*spec*, *placement*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

class spinn_front_end_common.abstract_models.**AbstractSendMeMulticastCommandsVertex**
Bases: `object`

A vertex which wants to commands to be sent to it as multicast packets at fixed points in the simulation

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spinn_front_end_common.abstract_models.**AbstractSupportsDatabaseInjection**
Bases: `object`

Marks a machine vertex as supporting injection of information via a database running on the controlling host.

is_in_injection_mode
Whether this vertex is actually in injection mode.

class spinn_front_end_common.abstract_models.**AbstractVertexWithEdgeToDependentVertices**
Bases: `object`

A vertex with a dependent vertices, which should be connected to this vertex by an edge directly to each of them

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

1.1.2 spinn_front_end_common.common_model_binaries package

1.1.2.1 Module contents

This module contains no python code

1.1.3 spinn_front_end_common.interface package

1.1.3.1 Subpackages

spinn_front_end_common.interface.buffer_management package

Subpackages

spinn_front_end_common.interface.buffer_management.buffer_models package

Submodules

spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host module

class `spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_bu`

Bases: `object`

Indicates that this object can receive buffers

get_minimum_buffer_sdram_usage()

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids()

Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)

Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host module

class `spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffe`

Bases: `object`

Interface to an object that sends buffers of keys to be transmitted at given timestamps in the simulation

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Gets the max possible size of a buffered region

Parameters **region** (*int*) – the region to find the max possible size of

Returns the max possible size of the buffered region

Return type int

get_next_key(*region*)
Get the next key in the given region

Parameters **region** (*int*) – The region to get the next key from

Returns The next key, or None if there are no more keys

Return type int

get_next_timestamp(*region*)
Get the next timestamp at which there are still keys to be sent for the given region

Parameters **region** (*int*) – The region to get the timestamp for

Returns The timestamp of the next available keys

Return type int

get_region_buffer_size(*region*)
Get the size of the buffer to be used in SDRAM on the machine for the region in bytes

Parameters **region** (*int*) – The region to get the buffer size of

Returns The size of the buffer space in bytes

Return type int

get_regions()
Get the set of regions for which there are keys to be sent

Returns Iterable of region ids

Return type iterable of int

is_empty(*region*)
Return true if there are no spikes to be buffered for the specified region

Parameters **region** (*int*) – The region to get the next key from

Returns True if there are no keys to send for the region, False otherwise

Return type bool

is_next_key(*region*, *timestamp*)
Determine if there are still keys to be sent at the given timestamp for the given region

Parameters

- **region** (*int*) – The region to determine if there are keys for
- **timestamp** (*int*) – The timestamp to determine if there are more keys for

Returns True if there are more keys to send for the parameters, False otherwise

Return type bool

is_next_timestamp(*region*)
Determine if there is another timestamp with data to be sent

Parameters **region** (*int*) – The region to determine if there is more data for

Returns True if there is more data, False otherwise

Return type int

rewind(*region*)

Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_ir module

class spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_h

Bases: *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host.AbstractSendsBuffersFromHost*

Implementation of the AbstractSendsBuffersFromHost which uses an existing set of buffers for the details

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Return the max possible size of a buffered region

Parameters **region** – the region to find the max possible size of

Returns the max possible size of the buffered region

get_next_key(*region*)
Get the next key for a given region

Parameters **region** – the region to get the next key from

get_next_timestamp(*region*)
Return the next time stamp available in the buffered region

Parameters **region** – the region id which is being asked

Returns the next time stamp

get_region_buffer_size(*region*)
Return the size of a given regions buffer

Parameters **region** – the region to find the size of

Returns the size of the buffer

get_regions()
Return the regions which has buffers to send

is_empty(*region*)
Check if a region is empty

Parameters **region** – the region id to check

Returns bool

is_next_key (*region*, *timestamp*)
Check if there is more keys to transmit for a given region in a given timestamp

Parameters

- **region** – the region id to check
- **timestamp** – the timestamp to check

Returns bool

is_next_timestamp (*region*)
Check if there are more time stamps which need transmitting

Parameters **region** – the region to check

Returns boolean

rewind (*region*)
Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (`int`) – The region to rewind

send_buffers

Module contents

class spinn_front_end_common.interface.buffer_management.buffer_models.**AbstractReceiveBuffe**
Bases: `object`

Indicates that this object can receive buffers

get_minimum_buffer_sdram_usage ()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space (*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids ()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address (*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

class spinn_front_end_common.interface.buffer_management.buffer_models.**AbstractSendsBuffers**

Bases: object

Interface to an object that sends buffers of keys to be transmitted at given timestamps in the simulation

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Gets the max possible size of a buffered region

Parameters **region** (*int*) – the region to find the max possible size of

Returns the max possible size of the buffered region

Return type int

get_next_key(*region*)
Get the next key in the given region

Parameters **region** (*int*) – The region to get the next key from

Returns The next key, or None if there are no more keys

Return type int

get_next_timestamp(*region*)
Get the next timestamp at which there are still keys to be sent for the given region

Parameters **region** (*int*) – The region to get the timestamp for

Returns The timestamp of the next available keys

Return type int

get_region_buffer_size(*region*)
Get the size of the buffer to be used in SDRAM on the machine for the region in bytes

Parameters **region** (*int*) – The region to get the buffer size of

Returns The size of the buffer space in bytes

Return type int

get_regions()
Get the set of regions for which there are keys to be sent

Returns Iterable of region ids

Return type iterable of int

is_empty(*region*)
Return true if there are no spikes to be buffered for the specified region

Parameters **region** (*int*) – The region to get the next key from

Returns True if there are no keys to send for the region, False otherwise

Return type bool

is_next_key(*region*, *timestamp*)
Determine if there are still keys to be sent at the given timestamp for the given region

Parameters

- **region** (*int*) – The region to determine if there are keys for
- **timestamp** (*int*) – The timestamp to determine if there are more keys for

Returns True if there are more keys to send for the parameters, False otherwise

Return type bool

is_next_timestamp(*region*)
Determine if there is another timestamp with data to be sent

Parameters **region** (`int`) – The region to determine if there is more data for

Returns True if there is more data, False otherwise

Return type int

rewind(*region*)

Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (`int`) – The region to rewind

class spinn_front_end_common.interface.buffer_management.buffer_models.**SendsBuffersFromHost**
Bases: *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host.AbstractSendsBuffersFromHost*

Implementation of the AbstractSendsBuffersFromHost which uses an existing set of buffers for the details

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Return the max possible size of a buffered region

Parameters **region** – the region to find the max possible size of

Returns the max possible size of the buffered region

get_next_key(*region*)
Get the next key for a given region

Parameters **region** – the region to get the next key from

get_next_timestamp(*region*)
Return the next time stamp available in the buffered region

Parameters **region** – the region id which is being asked

Returns the next time stamp

get_region_buffer_size(*region*)
Return the size of a given regions buffer

Parameters **region** – the region to find the size of

Returns the size of the buffer

get_regions()
Return the regions which has buffers to send

is_empty(*region*)
Check if a region is empty

Parameters **region** – the region id to check

Returns bool

is_next_key(*region*, *timestamp*)
Check if there is more keys to transmit for a given region in a given timestamp

Parameters

- **region** – the region id to check
- **timestamp** – the timestamp to check

Returns bool

is_next_timestamp(*region*)
Check if there are more time stamps which need transmitting

Parameters **region** – the region to check

Returns boolean

rewind(*region*)
Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

send_buffers

spinn_front_end_common.interface.buffer_management.storage_objects package

Submodules

spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving
Bases: object

Stores the information received through the buffering output technique from the SpiNNaker system. The data kept includes the last sent packet and last received packet, their correspondent sequence numbers, the data retrieved, a flag to identify if the data from a core has been flushed and the final state of the buffering output state machine

Parameters **store_to_file** (*bool*) – A boolean to identify if the data will be stored in memory using a byte array or in a temporary file on the disk

clear(*x*, *y*, *p*, *region_id*)
clears the data from a given data region (only clears things associated with a given data recording region).

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord
- **region_id** – the recording region id to clear data from

Return type None

flushing_data_from_region(*x*, *y*, *p*, *region*, *data*)
Store flushed data from a region of a core on a chip, and mark it as being flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip

- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

get_end_buffering_sequence_number (*x, y, p*)
Get the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The last sequence number

Return type int

get_end_buffering_state (*x, y, p, region*)
Get the end state of the buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The end state

get_region_data (*x, y, p, region*)
Get the data stored for a given region of a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns an array contained all the data received during the, simulation, and a flag indicating if any data was missing

Return type (bytearray, bool)

get_region_data_pointer (*x, y, p, region*)
Get the data received during the simulation for a region of a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns all the data received during the simulation, and a flag indicating if any data was lost

Return type (`spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage`, bool)

is_data_from_region_flushed(*x*, *y*, *p*, *region*)
Check if the data region has been flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns True if the region has been flushed. False otherwise

Return type bool

is_end_buffering_sequence_number_stored(*x*, *y*, *p*)
Determine if the last sequence number has been retrieved

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the number has been retrieved

Return type bool

is_end_buffering_state_recovered(*x*, *y*, *p*, *region*)
Determine if the end state has been stored

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the state has been stored

last_received_packet_from_core(*x*, *y*, *p*)
Get the last packet received for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns SpinnakerRequestReadData packet received

Return type `spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData`

last_sent_packet_to_core(*x*, *y*, *p*)
Retrieve the last packet sent to a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip

- **p** (*int*) – Core within the specified chip

Returns last HostDataRead packet sent

Return type spinnman.messages.eieio.command_messages.host_data_read.HostDataRead

last_sequence_no_for_core (*x*, *y*, *p*)
Get the last sequence number for a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last sequence number used

Return type int

resume ()
Resets states so that it can behave in a resumed mode

store_data_in_region_buffer (*x*, *y*, *p*, *region*, *data*)
Store some information in the correspondent buffer class for a specific chip, core and region

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

store_end_buffering_sequence_number (*x*, *y*, *p*, *sequence*)
Store the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence** (*int*) – The last sequence number

store_end_buffering_state (*x*, *y*, *p*, *region*, *state*)
Store the end state of buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **state** – The end state

store_last_received_packet_from_core (*x*, *y*, *p*, *packet*)
Store the most recent packet received from SpiNNaker for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData) – SpinnakerRequestReadData packet received

store_last_sent_packet_to_core (*x*, *y*, *p*, *packet*)
Store the last packet sent to the given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead) – last HostDataRead packet sent

update_sequence_no_for_core (*x*, *y*, *p*, *sequence_no*)
Set the last sequence number used

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence_no** (*int*) – last sequence number used

Return type None

spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_

Bases: object

A set of keys to be sent at given timestamps for a given region of data. Note that keys must be added in timestamp order or else an exception will be raised

add_key (*timestamp*, *key*)
Add a key to be sent at a given time

Parameters

- **timestamp** (*int*) – The time at which the key is to be sent
- **key** (*int*) – The key to send

add_keys (*timestamp*, *keys*)
Add a set of keys to be sent at the given time

Parameters

- **timestamp** (*int*) – The time at which the keys are to be sent

- **keys** (*iterable of int*) – The keys to send

buffer_size
property method for getting the max size of this buffer

clear()
Clears the buffer

current_timestamp
Get the current timestamp in the iterator

get_n_bytes(*n_keys*)
Get the number of bytes used by a given number of keys

Parameters **n_keys** (*int*) – The number of keys

get_n_keys(*timestamp*)
Get the number of keys for a given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

is_next_key(*timestamp*)
Determine if there is another key for the given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

Return type bool

is_next_timestamp
Determines if the region is empty

Returns True if the region is empty, false otherwise

Return type bool

max_buffer_size_possible
Get the max possible size of a buffer from this region

max_packets_in_timestamp
The maximum number of packets in any time stamp

n_timestamps
The number of timestamps available

Return type int

next_key
The next key to be sent

Return type int

next_timestamp
The next timestamp of the data to be sent, or None if no more data

Return type int or None

rewind()
Rewind the buffer to initial position.

timestamps
The timestamps for which there are keys

Return type iterable of int

total_region_size
Get the max size of this region

spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque

Bases: object

A tracker of buffers sent / to send for a region

Parameters

- **region** (*int*) – The region being managed
- **sent_stop_message** (*bool*) – True if the stop message has been sent
- **n_sequences_per_tranmission** (*int*) – The number of sequences allowed in each transmission set

add_message_to_send(*message*)
Add a message to send. The message is converted to a sequenced message.

Parameters **message** (spinnman.messages.eieio.abstract_messages.AbstractEIEIOMessage) – The message to be added

is_empty()
Determine if there are no messages

Return type int

is_full
Determine if the number of messages sent is at the limit for the sequencing system

Return type bool

messages
The messages that have been added to the set

Return type iterable of spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData

send_stop_message()
Send a message to indicate the end of all the messages

update_last_received_sequence_number(*last_received_sequence_no*)
Updates the last received sequence number. If the sequence number is within the valid window, packets before the sequence number within the window are removed, and the last received sequence number is updated, thus moving the window for the next call. If the sequence number is not within the valid window, it is assumed to be invalid and so is ignored.

Parameters **last_received_sequence_no** (*int*) – The new sequence number

Returns True if update went ahead, False if it was ignored

Return type bool

spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state module

class spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_sta

Bases: object

Stores information related to a single channel output buffering state, as it is retrieved at the end of a simulation on the SpiNNaker system.

Parameters

- **start_address** – start buffering area memory address (32 bits)
- **current_write** – address where data was last written (32 bits)
- **current_read** – address where data was last read (32 bits)
- **end_address** – The address of first byte after the buffer (32 bits)
- **region_id** – The id of the region (8 bits)
- **missing_info** – True if the region overflowed during the simulation (8 bits)
- **last_buffer_operation** – Last operation performed on the buffer - read or write (8 bits)

ChannelBufferStateSize = 24

static create_from_bytearray(*data*)

current_read

current_write

end_address

is_state_updated

last_buffer_operation

missing_info

region_id

set_update_completed()

static size_of_channel_state()

start_address

update_last_operation(*operation*)

update_read_pointer(*read_ptr*)

spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state module

class spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_stat

Bases: object

Stores the buffering state at the end of a simulation

Parameters

- **buffering_out_fsm_state** – Final sequence number received
- **list_channel_buffer_state** – a list of channel state, where each channel is stored in a ChannelBufferState object

buffering_out_fsm_state

channel_buffer_state(*i*)

get_missing_info_for_region(*region_id*)

get_state_for_region(*region_id*)

static size_of_region(*n_regions_to_record*)

Module contents

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedReceiving**

Bases: object

Stores the information received through the buffering output technique from the SpiNNaker system. The data kept includes the last sent packet and last received packet, their correspondent sequence numbers, the data retrieved, a flag to identify if the data from a core has been flushed and the final state of the buffering output state machine

Parameters **store_to_file** (*bool*) – A boolean to identify if the data will be stored in memory using a byte array or in a temporary file on the disk

clear(*x*, *y*, *p*, *region_id*)

clears the data from a given data region (only clears things associated with a given data recording region).

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord
- **region_id** – the recording region id to clear data from

Return type None

flushing_data_from_region(*x*, *y*, *p*, *region*, *data*)

Store flushed data from a region of a core on a chip, and mark it as being flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

get_end_buffering_sequence_number (*x*, *y*, *p*)
Get the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The last sequence number

Return type int

get_end_buffering_state (*x*, *y*, *p*, *region*)
Get the end state of the buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The end state

get_region_data (*x*, *y*, *p*, *region*)
Get the data stored for a given region of a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns an array contained all the data received during the, simulation, and a flag indicating if any data was missing

Return type (bytearray, bool)

get_region_data_pointer (*x*, *y*, *p*, *region*)
Get the data received during the simulation for a region of a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns all the data received during the simulation, and a flag indicating if any data was lost

Return type (`spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage`, bool)

is_data_from_region_flushed(*x*, *y*, *p*, *region*)
Check if the data region has been flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns True if the region has been flushed. False otherwise

Return type bool

is_end_buffering_sequence_number_stored(*x*, *y*, *p*)
Determine if the last sequence number has been retrieved

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the number has been retrieved

Return type bool

is_end_buffering_state_recovered(*x*, *y*, *p*, *region*)
Determine if the end state has been stored

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the state has been stored

last_received_packet_from_core(*x*, *y*, *p*)
Get the last packet received for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns SpinnakerRequestReadData packet received

Return type `spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData`

last_sent_packet_to_core(*x*, *y*, *p*)
Retrieve the last packet sent to a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip

- **p** (*int*) – Core within the specified chip

Returns last HostDataRead packet sent

Return type spinnman.messages.eieio.command_messages.host_data_read.HostDataRead

last_sequence_no_for_core (*x*, *y*, *p*)
Get the last sequence number for a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last sequence number used

Return type int

resume ()
Resets states so that it can behave in a resumed mode

store_data_in_region_buffer (*x*, *y*, *p*, *region*, *data*)
Store some information in the correspondent buffer class for a specific chip, core and region

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

store_end_buffering_sequence_number (*x*, *y*, *p*, *sequence*)
Store the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence** (*int*) – The last sequence number

store_end_buffering_state (*x*, *y*, *p*, *region*, *state*)
Store the end state of buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **state** – The end state

store_last_received_packet_from_core (*x*, *y*, *p*, *packet*)
Store the most recent packet received from SpiNNaker for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData) – SpinnakerRequestReadData packet received

store_last_sent_packet_to_core (*x*, *y*, *p*, *packet*)
Store the last packet sent to the given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead) – last HostDataRead packet sent

update_sequence_no_for_core (*x*, *y*, *p*, *sequence_no*)
Set the last sequence number used

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence_no** (*int*) – last sequence number used

Return type None

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedSendingReg**
Bases: object

A set of keys to be sent at given timestamps for a given region of data. Note that keys must be added in timestamp order or else an exception will be raised

add_key (*timestamp*, *key*)
Add a key to be sent at a given time

Parameters

- **timestamp** (*int*) – The time at which the key is to be sent
- **key** (*int*) – The key to send

add_keys (*timestamp*, *keys*)
Add a set of keys to be sent at the given time

Parameters

- **timestamp** (*int*) – The time at which the keys are to be sent
- **keys** (*iterable of int*) – The keys to send

buffer_size
property method for getting the max size of this buffer

clear()
Clears the buffer

current_timestamp
Get the current timestamp in the iterator

get_n_bytes(*n_keys*)
Get the number of bytes used by a given number of keys

Parameters **n_keys** (*int*) – The number of keys

get_n_keys(*timestamp*)
Get the number of keys for a given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

is_next_key(*timestamp*)
Determine if there is another key for the given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

Return type bool

is_next_timestamp
Determines if the region is empty

Returns True if the region is empty, false otherwise

Return type bool

max_buffer_size_possible
Get the max possible size of a buffer from this region

max_packets_in_timestamp
The maximum number of packets in any time stamp

n_timestamps
The number of timestamps available

Return type int

next_key
The next key to be sent

Return type int

next_timestamp
The next timestamp of the data to be sent, or None if no more data

Return type int or None

rewind()
Rewind the buffer to initial position.

timestamps
The timestamps for which there are keys

Return type iterable of int

total_region_size
Get the max size of this region

class spinn_front_end_common.interface.buffer_management.storage_objects.**BuffersSentDeque**(

Bases: object

A tracker of buffers sent / to send for a region

Parameters

- **region** (*int*) – The region being managed
- **sent_stop_message** (*bool*) – True if the stop message has been sent
- **n_sequences_per_tranmission** (*int*) – The number of sequences allowed in each transmission set

add_message_to_send(*message*)

Add a message to send. The message is converted to a sequenced message.

Parameters **message** (spinnman.messages.eieio.abstract_messages.AbstractEIEIOMessage) – The message to be added

is_empty()

Determine if there are no messages

Return type int

is_full

Determine if the number of messages sent is at the limit for the sequencing system

Return type bool

messages

The messages that have been added to the set

Return type iterable of spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData

send_stop_message()

Send a message to indicate the end of all the messages

update_last_received_sequence_number(*last_received_sequence_no*)

Updates the last received sequence number. If the sequence number is within the valid window, packets before the sequence number within the window are removed, and the last received sequence number is updated, thus moving the window for the next call. If the sequence number is not within the valid window, it is assumed to be invalid and so is ignored.

Parameters **last_received_sequence_no** (*int*) – The new sequence number

Returns True if update went ahead, False if it was ignored

Return type bool

class spinn_front_end_common.interface.buffer_management.storage_objects.**ChannelBufferState**

Bases: object

Stores information related to a single channel output buffering state, as it is retrieved at the end of a simulation on the SpiNNaker system.

Parameters

- **start_address** – start buffering area memory address (32 bits)
- **current_write** – address where data was last written (32 bits)
- **current_read** – address where data was last read (32 bits)
- **end_address** – The address of first byte after the buffer (32 bits)
- **region_id** – The id of the region (8 bits)
- **missing_info** – True if the region overflowed during the simulation (8 bits)
- **last_buffer_operation** – Last operation performed on the buffer - read or write (8 bits)

ChannelBufferStateSize = 24

static **create_from_bytearray**(*data*)

current_read

current_write

end_address

is_state_updated

last_buffer_operation

missing_info

region_id

set_update_completed()

static **size_of_channel_state**()

start_address

update_last_operation(*operation*)

update_read_pointer(*read_ptr*)

class spinn_front_end_common.interface.buffer_management.storage_objects.**EndBufferingState**

Bases: object

Stores the buffering state at the end of a simulation

Parameters

- **buffering_out_fsm_state** – Final sequence number received
- **list_channel_buffer_state** – a list of channel state, where each channel is stored in a ChannelBufferState object

buffering_out_fsm_state

channel_buffer_state(*i*)

get_missing_info_for_region(*region_id*)

get_state_for_region(*region_id*)

static **size_of_region**(*n_regions_to_record*)

Submodules

spinn_front_end_common.interface.buffer_management.buffer_manager module

class spinn_front_end_common.interface.buffer_management.buffer_manager.**BufferManager**(*placem
tags,
transce
store_t*

Bases: object

Manager of send buffers

Parameters

- **placements** (pacman.model.placements.Placements) – The placements of the vertices
- **tags** (pacman.model.tags.Tags) – The tags assigned to the vertices
- **transceiver** (spinnman.transceiver.Transceiver) – The transceiver to use for sending and receiving information
- **store_to_file** (*bool*) – True if the data should be temporarily stored in a file instead of in RAM (default uses RAM)

add_receiving_vertex(*vertex*)
Add a vertex into the managed list for vertices which require buffers to be received from them during runtime

add_sender_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be sent to them during runtime

Parameters vertex (spinnaker.pyNN.models.abstract_models.buffer_models.AbstractSendsBuffersFromHost) – the vertex to be managed

clear_recorded_data(*x, y, p, recording_region_id*)
Removes the recorded data stored in memory.

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord
- **recording_region_id** – the recording region id

get_data_for_vertex(*placement, recording_region_id*)
Get a pointer to the data container for all the data retrieved during the simulation from a specific region area of a core

Parameters

- **placement** (*pacman.model.placements.Placement*) – the placement to get the data from
- **recording_region_id** (*int*) – desired recording data region

Returns pointer to a class which inherits from AbstractBufferedDataStorage

Return type `spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage`

load_initial_buffers()
Load the initial buffers for the senders using mem writes

receive_buffer_command_message(*packet*)
Handle an EIEIO command message for the buffers

Parameters **packet** (`spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage`) – The eieio message received

reload_buffer_files
The file paths for each buffered region for each sender vertex

reset()
Resets the buffered regions to start transmitting from the beginning of its expected regions and clears the buffered out data files

resume()
Resets any data structures needed before starting running again

sender_vertices
The vertices which are buffered

stop()
Indicates that the simulation has finished, so no further outstanding requests need to be processed

spinn_front_end_common.interface.buffer_management.recording_utilities module

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_last_sequence_nu**

Read the last sequence number from the data

Parameters

- **placement** – The placement from which to read the sequence number
- **transceiver** – The transceiver to use to read the sequence number
- **recording_data_address** – The address of the recording data from which to read the number

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_minimum_buffer_s**

Get the minimum buffer SDRAM

Parameters

- **buffered_sdram_per_timestep** (*list of int*) – The maximum number of bytes to use per timestep of recording, per recorded region. Disabled regions can specify 0.
- **n_machine_time_steps** (*int*) – The number of machine time steps for the simulation. Can be None if use_auto_pause_and_resume is True

- **minimum_sdram_for_buffering** (*int*) – The minimum SDRAM to reserve per recorded region for buffering

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_n_timesteps_in_b**

Get the number of time steps of data that can be stored in a given buffers space

Parameters

- **buffer_space** (*int*) – The space that will hold the data
- **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used by each region per timestep

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recorded_region_**

Get the ids of regions where recording is enabled

Parameters **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used by each region per timestep, where 0 indicates a disabled region

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recorded_region_**

Get the size of each recording region to be passed in to get_recording_header_array

Parameters

- **n_machine_time_steps** (*int*) – The duration of the simulation segment in time steps
- **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used per timestep in bytes per region
- **maximum_sdram_for_buffering** (*None or list of int*) – The maximum size of each buffer, or None if no maximum

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_data_s**

Get the size of the recorded data to be reserved

Parameters **recorded_region_sizes** – A list of sizes of each region to be recorded. A size of 0 is acceptable.

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_header**

Get data to be written for the recording header

Parameters

- **recorded_region_sizes** – A list of sizes of each region to be recorded. A size of 0 is acceptable.

- **time_between_triggers** – The minimum time between requesting reads of any region
- **buffer_size_before_request** – The amount of buffer to fill before a read request is sent
- **ip_tags** – A list of ip tags to extract the buffer tag from
- **buffering_tag** – The tag to use for buffering requests

Returns An array of values to be written as the header

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_heade**

Get the size of the data to be written for the recording header

Parameters **n_recorded_regions** – The number of regions to be recorded

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_regio**

Get the size of each recording region to be passed in to get_recording_resources, based on the details of the simulation

Parameters

- **buffered_sdram_per_timestep** (*list of int*) – The maximum number of bytes to use per timestep of recording, per recorded region. Disabled regions can specify 0.
- **n_machine_time_steps** (*int*) – The number of machine time steps for the simulation. Can be None if use_auto_pause_and_resume is True
- **minimum_sdram_for_buffering** (*int*) – The minimum SDRAM to reserve per recorded region for buffering
- **maximum_sdram_for_buffering** (*None or list of int*) – The maximum size of each buffer, or None if no maximum
- **use_auto_pause_and_resume** (*bool*) – True if automatic pause and resume is to be used for buffering

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_resou**

Get the resources for recording

Parameters

- **region_sizes** (*list of int*) – A list of the sizes of each region. A size of 0 is acceptable to indicate an empty region
- **buffering_ip_address** (*str*) – The ip address to receive buffering messages on, or None if buffering is not in use
- **buffering_port** (*int*) – The port to receive buffering messages on, or None if a port is to be assigned
- **notification_tag** (*int*) – The tag to send buffering messages with, or None to use a default tag

Return type `pacman.model.resources.ResourceContainer`

`spinn_front_end_common.interface.buffer_management.recording_utilities.`**`get_region_pointer`**(*p*
t
r
i
r
g

Get a pointer to a recording region

Parameters

- **placement** – The placement from which to read the pointer
- **transceiver** – The transceiver to use to read the pointer
- **recording_data_address** – The address of the recording data from which to read the pointer
- **region** – The index of the region to get the pointer of

Return type int

Module contents

class `spinn_front_end_common.interface.buffer_management.`**`BufferManager`**(*placements*, *tags*, *transceiver*, *store_to_file=False*)

Bases: `object`

Manager of send buffers

Parameters

- **placements** (`pacman.model.placements.Placements`) – The placements of the vertices
- **tags** (`pacman.model.tags.Tags`) – The tags assigned to the vertices
- **transceiver** (`spinnman.transceiver.Transceiver`) – The transceiver to use for sending and receiving information
- **store_to_file** (*bool*) – True if the data should be temporarily stored in a file instead of in RAM (default uses RAM)

add_receiving_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be received from them during runtime

add_sender_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be sent to them during runtime

Parameters **vertex** (`spinnaker.pyNN.models.abstract_models.buffer_models.AbstractSendsBuffersFromHost`) – the vertex to be managed

clear_recorded_data (*x*, *y*, *p*, *recording_region_id*)
Removes the recorded data stored in memory.

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord
- **recording_region_id** – the recording region id

get_data_for_vertex (*placement*, *recording_region_id*)
Get a pointer to the data container for all the data retrieved during the simulation from a specific region area of a core

Parameters

- **placement** (`pacman.model.placements.Placement`) – the placement to get the data from
- **recording_region_id** (*int*) – desired recording data region

Returns pointer to a class which inherits from AbstractBufferedDataStorage

Return type `spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage`

load_initial_buffers ()
Load the initial buffers for the senders using mem writes

receive_buffer_command_message (*packet*)
Handle an EIEIO command message for the buffers

Parameters **packet** (`spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage`) – The eieio message received

reload_buffer_files
The file paths for each buffered region for each sender vertex

reset ()
Resets the buffered regions to start transmitting from the beginning of its expected regions and clears the buffered out data files

resume ()
Resets any data structures needed before starting running again

sender_vertices
The vertices which are buffered

stop ()
Indicates that the simulation has finished, so no further outstanding requests need to be processed

spinn_front_end_common.interface.interface_functions package

Submodules

spinn_front_end_common.interface.interface_functions.application_finisher module

class `spinn_front_end_common.interface.interface_functions.application_finisher.`**`Applicatio`**

Bases: `object`

spinn_front_end_common.interface.interface_functions.application_runner module

class `spinn_front_end_common.interface.interface_functions.application_runner.`**`ApplicationRu`**

Bases: `object`

Ensures all cores are initialised correctly, ran, and completed successfully.

spinn_front_end_common.interface.interface_functions.buffer_extractor module

class `spinn_front_end_common.interface.interface_functions.buffer_extractor.`**`BufferExtracto`**

Bases: `object`

Extracts data in between runs

spinn_front_end_common.interface.interface_functions.buffer_manager_creator module

class `spinn_front_end_common.interface.interface_functions.buffer_manager_creator.`**`BufferMa`**

Bases: `object`

spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer module

class `spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer.`**`ChipIOBufCle`**

Bases: `object`

Updates the runtime of an application running on a spinnaker machine

spinn_front_end_common.interface.interface_functions.chip_iobuf_extractor module

class `spinn_front_end_common.interface.interface_functions.chip_iobuf_extractor.`**`ChipIOBufE`**

Bases: `object`

Extract iobuf buffers from the machine, and separates lines based on their prefix

spinn_front_end_common.interface.interface_functions.chip_provenance_updater module

class `spinn_front_end_common.interface.interface_functions.chip_provenance_updater.`**`ChipPro`**

Bases: `object`

Forces all cores to generate provenance data, and then exit

spinn_front_end_common.interface.interface_functions.chip_runtime_updater module

class spinn_front_end_common.interface.interface_functions.chip_runtime_updater.**ChipRuntime**

Bases: object

Updates the runtime of an application running on a spinnaker machine

spinn_front_end_common.interface.interface_functions.database_interface module

class spinn_front_end_common.interface.interface_functions.database_interface.**DatabaseInter**

Bases: object

Writes a database of the graph(s) and other information

database_file_path

needs_database

spinn_front_end_common.interface.interface_functions.dsg_region_reloader module

class spinn_front_end_common.interface.interface_functions.dsg_region_reloader.**DSGRegionRel**

Bases: object

Regenerates Data Specifications

spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper module

class spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper.**EdgeToNKey**

Bases: object

Works out the number of keys needed for each edge

spinn_front_end_common.interface.interface_functions.graph_binary_gatherer module

class spinn_front_end_common.interface.interface_functions.graph_binary_gatherer.**GraphBinar**

Bases: object

Extracts binaries to be executed

spinn_front_end_common.interface.interface_functions.graph_data_specification_writer module

class spinn_front_end_common.interface.interface_functions.graph_data_specification_writer

Bases: object

Executes data specification generation

spinn_front_end_common.interface.interface_functions.graph_measurer module

class spinn_front_end_common.interface.interface_functions.graph_measurer.**GraphMeasurer**

Bases: object

Works out how many chips a machine graph needs

spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer.**GraphI**

Bases: object

spinn_front_end_common.interface.interface_functions.hbp_allocator module

class spinn_front_end_common.interface.interface_functions.hbp_allocator.**HBPAllocator**

Bases: object

Request a machine from the HBP remote access server that will fit a number of chips

spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator module

class spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator.**HBPMa**

Bases: object

Generates the width and height of the maximum machine a given HBP server can generate

spinn_front_end_common.interface.interface_functions.host_execute_data_specification module

class spinn_front_end_common.interface.interface_functions.host_execute_data_specification

Bases: object

Executes the host based data specification

spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_graphs module

class spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_g

Bases: object

function to add chip power monitors into a given graph

spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gatherers module

class spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gat

Bases: object

Add edges from the recorded vertices to the local Live PacketGatherers

spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_graphs module

class spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_

Bases: object

function to add LPG's as required into a given graph

spinn_front_end_common.interface.interface_functions.load_executable_images module

class spinn_front_end_common.interface.interface_functions.load_executable_images.**LoadExecu**

Bases: object

spinn_front_end_common.interface.interface_functions.locate_executable_start_type module

class spinn_front_end_common.interface.interface_functions.locate_executable_start_type.**Lo**

Bases: object

spinn_front_end_common.interface.interface_functions.machine_execute_data_specification module

class spinn_front_end_common.interface.interface_functions.machine_execute_data_specificat

Bases: object

Executes the machine based data specification

spinnaker_based_data_specification_execution(*write_memory_map_report*, *dsg_targets*, *transceiver*, *app_id*)

Parameters

- **write_memory_map_report** –
- **dsg_targets** –
- **transceiver** –
- **app_id** –

Returns True

Return type bool

spinn_front_end_common.interface.interface_functions.machine_generator module

class spinn_front_end_common.interface.interface_functions.machine_generator.**MachineGenerat**

Bases: object

Interface to make a transceiver and a spinn_machine object

spinn_front_end_common.interface.interface_functions.notification_protocol module

class spinn_front_end_common.interface.interface_functions.notification_protocol.**Notificat**

Bases: object

The notification protocol for external device interaction

send_read_notification(*database_directory*)

Send the read notifications via the notification protocol

Parameters **database_directory** – the path to the database

Return type None:

send_start_resume_notification()
Send the start notifications via the notification protocol

Return type None:

send_stop_pause_notification()
Send the stop or pause notifications via the notification protocol

Return type None:

stop()
Ends the notification protocol

Return type None:

wait_for_confirmation()
Waits for devices to confirm they have read the database via the notification protocol

Return type None:

spinn_front_end_common.interface.interface_functions.placements_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.placements_provenance_gatherer.**I**
Bases: object

spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_chip_power_monitor module

class spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_chip_
Bases: object

Adds chip power monitor resources as required for a machine

spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_live_packet_gatherers module

class spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_live_
Bases: object

Adds Live Packet Gatherer resources as required for a machine

spinn_front_end_common.interface.interface_functions.profile_data_gatherer module

class spinn_front_end_common.interface.interface_functions.profile_data_gatherer.**ProfileDat**
Bases: object

spinn_front_end_common.interface.interface_functions.provenance_json_writer module

class spinn_front_end_common.interface.interface_functions.provenance_json_writer.**Provenanc**
Bases: object

Write provenance data into JSON

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '

spinn_front_end_common.interface.interface_functions.provenance_xml_writer module

class spinn_front_end_common.interface.interface_functions.provenance_xml_writer.**Provenance**

Bases: object

Write provenance data into XML

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '

spinn_front_end_common.interface.interface_functions.router_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.router_provenance_gatherer.**Route**

Bases: object

RouterProvenanceGatherer: gathers diagnostics from the routers.

spinn_front_end_common.interface.interface_functions.routing_setup module

class spinn_front_end_common.interface.interface_functions.routing_setup.**RoutingSetup**

Bases: object

spinn_front_end_common.interface.interface_functions.routing_table_loader module

class spinn_front_end_common.interface.interface_functions.routing_table_loader.**RoutingTabl**

Bases: object

spinn_front_end_common.interface.interface_functions.spalloc_allocator module

class spinn_front_end_common.interface.interface_functions.spalloc_allocator.**SpallocAllocat**

Bases: object

Request a machine from a SPALLOC server that will fit the given number of chips

spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator module

class spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator.**Sp**

Bases: object

Generates the width and height of the maximum machine a given allocation server can generate

spinn_front_end_common.interface.interface_functions.tags_loader module

class spinn_front_end_common.interface.interface_functions.tags_loader.**TagsLoader**

Bases: object

Loads tags onto the machine

static load_iptags(*iptags*, *transceiver*, *progress_bar*)

Loads all the iptags individually.

Parameters

- **iptags** – the iptags to be loaded.
- **transceiver** – the transceiver object

Return type None

static load_reverse_iptags (*reverse_ip_tags*, *transceiver*, *progress_bar*)
Loads all the reverse iptags individually.

Parameters

- **reverse_ip_tags** – the reverse iptags to be loaded
- **transceiver** – the transceiver object

Return type None

spinn_front_end_common.interface.interface_functions.tdma_agenda_builder module

class spinn_front_end_common.interface.interface_functions.tdma_agenda_builder.**TDMAAgendaBu**
Bases: object

algorithm that builds an agenda for transmissions. It uses a TDMA (Time division multiple access) system and graph colouring to deduce the agenda set up. Ensures parallel transmissions so that the destination should never be overloaded

spinn_front_end_common.interface.interface_functions.virtual_machine_generator module

class spinn_front_end_common.interface.interface_functions.virtual_machine_generator.**Virtua**
Bases: object

Module contents

class spinn_front_end_common.interface.interface_functions.**ApplicationFinisher**
Bases: object

class spinn_front_end_common.interface.interface_functions.**ApplicationRunner**
Bases: object

Ensures all cores are initialised correctly, ran, and completed successfully.

class spinn_front_end_common.interface.interface_functions.**BufferExtractor**
Bases: object

Extracts data in between runs

class spinn_front_end_common.interface.interface_functions.**BufferManagerCreator**
Bases: object

class spinn_front_end_common.interface.interface_functions.**ChipIOBufClearer**
Bases: object

Updates the runtime of an application running on a spinnaker machine

class spinn_front_end_common.interface.interface_functions.**ChipIOBufExtractor**
Bases: object

Extract iobuf buffers from the machine, and separates lines based on their prefix

class `spinn_front_end_common.interface.interface_functions.`**ChipProvenanceUpdater**
Bases: `object`

Forces all cores to generate provenance data, and then exit

class `spinn_front_end_common.interface.interface_functions.`**ChipRuntimeUpdater**
Bases: `object`

Updates the runtime of an application running on a spinnaker machine

class `spinn_front_end_common.interface.interface_functions.`**DatabaseInterface**
Bases: `object`

Writes a database of the graph(s) and other information

database_file_path

needs_database

class `spinn_front_end_common.interface.interface_functions.`**DSGRegionReloader**
Bases: `object`

Regenerates Data Specifications

class `spinn_front_end_common.interface.interface_functions.`**EdgeToNKeysMapper**
Bases: `object`

Works out the number of keys needed for each edge

class `spinn_front_end_common.interface.interface_functions.`**GraphBinaryGatherer**
Bases: `object`

Extracts binaries to be executed

class `spinn_front_end_common.interface.interface_functions.`**GraphDataSpecificationWriter**
Bases: `object`

Executes data specification generation

class `spinn_front_end_common.interface.interface_functions.`**GraphMeasurer**
Bases: `object`

Works out how many chips a machine graph needs

class `spinn_front_end_common.interface.interface_functions.`**GraphProvenanceGatherer**
Bases: `object`

class `spinn_front_end_common.interface.interface_functions.`**HBPAllocator**
Bases: `object`

Request a machine from the HBP remote access server that will fit a number of chips

class `spinn_front_end_common.interface.interface_functions.`**HBPMaxMachineGenerator**
Bases: `object`

Generates the width and height of the maximum machine a given HBP server can generate

class `spinn_front_end_common.interface.interface_functions.`**HostExecuteDataSpecification**
Bases: `object`

Executes the host based data specification

class `spinn_front_end_common.interface.interface_functions.`**InsertEdgesToLivePacketGatherers**
Bases: `object`

Add edges from the recorded vertices to the local Live PacketGatherers

class spinn_front_end_common.interface.interface_functions.**InsertLivePacketGatherersToGrapl**
Bases: object

function to add LPG's as required into a given graph

class spinn_front_end_common.interface.interface_functions.**LoadExecutableImages**
Bases: object

class spinn_front_end_common.interface.interface_functions.**LocateExecutableStartType**
Bases: object

class spinn_front_end_common.interface.interface_functions.**MachineExecuteDataSpecification**
Bases: object

Executes the machine based data specification

spinnaker_based_data_specification_execution(*write_memory_map_report*, *dsg_targets*, *transceiver*, *app_id*)

Parameters

- **write_memory_map_report** –
- **dsg_targets** –
- **transceiver** –
- **app_id** –

Returns True

Return type bool

class spinn_front_end_common.interface.interface_functions.**MachineGenerator**
Bases: object

Interface to make a transceiver and a spinn_machine object

class spinn_front_end_common.interface.interface_functions.**NotificationProtocol**
Bases: object

The notification protocol for external device interaction

send_read_notification(*database_directory*)
Send the read notifications via the notification protocol

Parameters **database_directory** – the path to the database

Return type None:

send_start_resume_notification()
Send the start notifications via the notification protocol

Return type None:

send_stop_pause_notification()
Send the stop or pause notifications via the notification protocol

Return type None:

stop()
Ends the notification protocol

Return type None:

wait_for_confirmation()
Waits for devices to confirm they have read the database via the notification protocol

Return type None:

class spinn_front_end_common.interface.interface_functions.**PlacementsProvenanceGatherer**
Bases: object

class spinn_front_end_common.interface.interface_functions.**PreAllocateResourcesForLivePacke**
Bases: object

Adds Live Packet Gatherer resources as required for a machine

class spinn_front_end_common.interface.interface_functions.**ProvenanceJSONWriter**
Bases: object

Write provenance data into JSON

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '

class spinn_front_end_common.interface.interface_functions.**ProvenanceXMLWriter**
Bases: object

Write provenance data into XML

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '

class spinn_front_end_common.interface.interface_functions.**RouterProvenanceGatherer**
Bases: object

RouterProvenanceGatherer: gathers diagnostics from the routers.

class spinn_front_end_common.interface.interface_functions.**RoutingSetup**
Bases: object

class spinn_front_end_common.interface.interface_functions.**RoutingTableLoader**
Bases: object

class spinn_front_end_common.interface.interface_functions.**SpallocAllocator**
Bases: object

Request a machine from a SPALLOC server that will fit the given number of chips

class spinn_front_end_common.interface.interface_functions.**SpallocMaxMachineGenerator**
Bases: object

Generates the width and height of the maximum machine a given allocation server can generate

class spinn_front_end_common.interface.interface_functions.**TagsLoader**
Bases: object

Loads tags onto the machine

static load_iptags(*iptags*, *transceiver*, *progress_bar*)
Loads all the iptags individually.

Parameters

- **iptags** – the iptags to be loaded.
- **transceiver** – the transceiver object

Return type None

static load_reverse_iptags(*reverse_ip_tags*, *transceiver*, *progress_bar*)
Loads all the reverse iptags individually.

Parameters

- **reverse_ip_tags** – the reverse iptags to be loaded

- **transceiver** – the transceiver object

Return type None

class spinn_front_end_common.interface.interface_functions.**TDMAAgendaBuilder**

Bases: object

algorithm that builds an agenda for transmissions. It uses a TDMA (Time division multiple access) system and graph colouring to deduce the agenda set up. Ensures parallel transmissions so that the destination should never be overloaded

class spinn_front_end_common.interface.interface_functions.**VirtualMachineGenerator**

Bases: object

spinn_front_end_common.interface.profiling package

Submodules

spinn_front_end_common.interface.profiling.abstract_has_profile_data module

class spinn_front_end_common.interface.profiling.abstract_has_profile_data.**AbstractHasProf**

Bases: object

Indicates an object that can record a profile

get_profile_data (*transceiver*, *placement*)

Get the profile data recorded during simulation

Return type *spinn_front_end_common.interface.profiling.profile_data.ProfileData*

spinn_front_end_common.interface.profiling.profile_data module

class spinn_front_end_common.interface.profiling.profile_data.**ProfileData** (*tag_labels*)

Bases: object

A container for profile data

Parameters **tag_labels** (*list (str)*) – A list of labels indexed by tag id

DURATION = 1

START_TIME = 0

add_data (*data*)

Add profiling data read from the profile section

Parameters **data** (*bytearray*) – Data read from the profile section on the machine

get_mean_ms (*tag*)

Get the mean time in milliseconds spent on operations with the given tag

Parameters **tag** (*str*) – The tag to get the mean time for

Return type float

get_mean_ms_per_ts (*tag*, *run_time_ms*, *machine_time_step_ms*)

Get the mean time in milliseconds spent on operations with the given tag per timestep

Parameters

- **tag** (*str*) – The tag to get the data for
- **machine_time_step_ms** (*int*) – The time step of the simulation in microseconds
- **run_time_ms** (*float*) – The run time of the simulation in milliseconds

Return type float

get_mean_n_calls_per_ts (*tag*, *run_time_ms*, *machine_time_step_ms*)

Get the mean number of times the given tag was recorded per timestep

Parameters

- **tag** (*str*) – The tag to get the data for
- **machine_time_step_ms** (*int*) – The time step of the simulation in microseconds
- **run_time_ms** (*float*) – The run time of the simulation in milliseconds

Return type float

get_n_calls (*tag*)

Get the number of times the given tag was recorded

Parameters **tag** (*str*) – The tag to get the number of calls of

Return type int

tags

The tags recorded as labels

Return type list of str

spinn_front_end_common.interface.profiling.profile_utils module

spinn_front_end_common.interface.profiling.profile_utils.**get_profile_region_size** (*n_samples*)

Get the size of the region of the profile data

Parameters **n_samples** – number of different samples to record

:return the size in bytes used by the profile region

spinn_front_end_common.interface.profiling.profile_utils.**get_profiling_data** (*profile_region*, *tag_labels*, *txrx*, *placement*)

Utility function to get profile data from a profile region

Parameters

- **profile_region** – dsg region to get profiling data out of sdram
- **tag_labels** – labels for the profiling data
- **txrx** – transceiver code
- **placement** – placement

Returns ProfileData

`spinn_front_end_common.interface.profiling.profile_utils.`**`reserve_profile_region`**(*spec*, *region*, *n_samples*)

Reserves the profile region for recording the profile data

Parameters

- **`spec`** – the dsg specification writer
- **`region`** – region id for the profile data
- **`n_samples`** – n elements being sampled

Return type None

`spinn_front_end_common.interface.profiling.profile_utils.`**`write_profile_region_data`**(*spec*, *region*, *n_samples*)

Writes the profile region data

Parameters

- **`spec`** – the dsg specification writer
- **`region`** – region id for the profile data
- **`n_samples`** – n elements being sampled

Return type None

Module contents

spinn_front_end_common.interface.provenance package

Submodules

spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data module

class `spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data.`

Bases: `object`

Indicates an object that provides locally obtained provenance data

`get_local_provenance_data`()

Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_machine module

class `spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_ma`

Bases: `object`

Indicates that an object provides provenance data retrieved from the machine

get_provenance_data_from_machine (*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

spinn_front_end_common.interface.provenance.pacman_provenance_extractor module

class spinn_front_end_common.interface.provenance.pacman_provenance_extractor.**PacmanProvena**
Bases: object

Extracts Provenance data from a PACMANAlgorithmExecutor

clear ()
clears the provenance data store

Return type None

data_items
returns the provenance data items

Returns list of provenance data items.

Return type iterable of ProvenanceDataItem

extract_provenance (*executor*)
acquires the timings from pacman algorithms (provenance data)

Parameters **executor** – the pacman workflow executor

Return type None

spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl module

class spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_imp
Bases: *spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_fron AbstractProvidesProvenanceDataFromMachine*

An implementation that gets provenance data from a region of ints on the machine

class PROVENANCE_DATA_ENTRIES
Bases: enum.Enum

CALLBACK_QUEUE_OVERLOADED = 1

DMA_QUEUE_OVERLOADED = 2

MAX_NUMBER_OF_TIMER_TIC_OVERRUN = 4

TIMER_TIC_HAS_OVERRUN = 3

TRANSMISSION_EVENT_OVERFLOW = 0

get_provenance_data_from_machine (*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

static get_provenance_data_size(*n_additional_data_items*)

reserve_provenance_data_region(*spec*)

Module contents

class spinn_front_end_common.interface.provenance.**AbstractProvidesLocalProvenanceData**

Bases: object

Indicates an object that provides locally obtained provenance data

get_local_provenance_data()

Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

class spinn_front_end_common.interface.provenance.**AbstractProvidesProvenanceDataFromMachine**

Bases: object

Indicates that an object provides provenance data retrieved from the machine

get_provenance_data_from_machine(*transceiver*, *placement*)

Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

class spinn_front_end_common.interface.provenance.**PacmanProvenanceExtractor**

Bases: object

Extracts Provenance data from a PACMANAlgorithmExecutor

clear()

clears the provenance data store

Return type None

data_items

returns the provenance data items

Returns list of provenance data items.

Return type iterable of ProvenanceDataItem

extract_provenance(*executor*)

acquires the timings from pacman algorithms (provenance data)

Parameters **executor** – the pacman workflow executor

Return type None

class spinn_front_end_common.interface.provenance.**ProvidesProvenanceDataFromMachineImpl**

Bases: *spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_fro*
AbstractProvidesProvenanceDataFromMachine

An implementation that gets provenance data from a region of ints on the machine

class PROVENANCE_DATA_ENTRIES
Bases: `enum.Enum`

CALLBACK_QUEUE_OVERLOADED = 1

DMA_QUEUE_OVERLOADED = 2

MAX_NUMBER_OF_TIMER_TIC_OVERRUN = 4

TIMER_TIC_HAS_OVERRUN = 3

TRANSMISSION_EVENT_OVERFLOW = 0

get_provenance_data_from_machine(*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

static get_provenance_data_size(*n_additional_data_items*)

reserve_provenance_data_region(*spec*)

spinn_front_end_common.interface.simulation package

Submodules

spinn_front_end_common.interface.simulation.simulation_utilities module

`spinn_front_end_common.interface.simulation.simulation_utilities.`**get_simulation_header_arra**

Get data to be written to the simulation header

Parameters

- **binary_file_name** – The name of the binary of the application
- **machine_time_step** – The time step of the simulation
- **time_scale_factor** – The time scaling of the simulation

Returns An array of values to be written as the simulation header

Module contents

1.1.3.2 Submodules

1.1.3.3 spinn_front_end_common.interface.abstract_spinnaker_base module

main interface for the spinnaker tools

class spinn_front_end_common.interface.abstract_spinnaker_base.**AbstractSpinnakerBase**(*configfil ex- e- cutable_ graph_l databas ex- tra_algo n_chips_ de- fault_co val- i- da- tion_cfg front_en*

Bases: `spinn_front_end_common.utilities.simulator_interface.SimulatorInterface`

Main interface into the tools logic flow

add_application_edge(*edge_to_add*, *partition_identifier*)

Parameters

- **edge_to_add** –
- **partition_identifier** – the partition identifier for the outgoing edge partition

Return type None

add_application_vertex(*vertex_to_add*)

Parameters **vertex_to_add** – the vertex to add to the graph

Return type None

Raises ConfigurationException when both graphs contain vertices

add_extraction_timing(*timing*)

add_live_packet_gatherer_parameters(*live_packet_gatherer_params*, *vertex_to_record_from*)

adds params for a new LPG if needed, or adds to the tracker for same params.

Parameters

- **live_packet_gatherer_params** – params to look for a LPG
- **vertex_to_record_from** – the vertex that needs to send to a given LPG

Return type None

add_machine_edge(*edge*, *partition_id*)

Parameters

- **edge** – the edge to add to the graph
- **partition_id** – the partition identifier for the outgoing edge partition

Return type None

add_machine_vertex(*vertex*)

Parameters **vertex** – the vertex to add to the graph

Return type None

Raises ConfigurationException when both graphs contain vertices

add_socket_address(*socket_address*)

Parameters **socket_address** –

Return type None

application_graph

buffer_manager
The buffer manager being used for loading/extracting buffers

config
helper method for the front end implementations until we remove config

dsg_algorithm
The dsg algorithm used by the tools

exception_handler(*exctype*, *value*, *traceback_obj*)
handler of exceptions

Parameters

- **exctype** – the type of execution received
- **value** – the value of the exception
- **traceback_obj** – the trace back stuff

extend_extra_load_algorithms(*extra_load_algorithms*)

extend_extra_mapping_algorithms(*extra_mapping_algorithms*)

extend_extra_post_run_algorithms(*extra_post_run_algorithms*)

generate_file_machine()

get_current_time()

get_number_of_available_cores_on_machine
returns the number of available cores on the machine after taking into account pre allocated resources

Returns number of available cores

Return type int

graph_mapper

has_ran

has_reset_last

increment_none_labelled_edge_count()
Increment the number of new edges which have not been labelled.

increment_none_labelled_vertex_count()
Increment the number of new vertices which have not been labelled.

machine
The python machine object

Return type spinn_machine.Machine

machine_graph

machine_time_step

no_machine_time_steps

none_labelled_edge_count
The number of times edges have not been labelled.

none_labelled_vertex_count
The number of times vertices have not been labelled.

placements

prepend_extra_pre_run_algorithms(*extra_pre_run_algorithms*)

reset()
Code that puts the simulation back at time zero

routing_infos

run(*run_time*)
Run a simulation for a fixed amount of time

Parameters **run_time** – the run duration in milliseconds.

run_until_complete()
Run a simulation until it completes

set_n_chips_required(*n_chips_required*)

set_up_machine_specifics(*hostname*)
Adds machine specifics for the different modes of execution

Parameters **hostname** – machine name

Return type None

set_up_timings(*machine_time_step=None*, *time_scale_factor=None*)
Set up timings of the machine

Parameters

- **machine_time_step** – An explicitly specified time step for the machine. If None, the value is read from the config
- **time_scale_factor** – An explicitly specified time scale factor for the simulation. If None, the value is read from the config

signal_handler(*signal*, *frame*)
handles closing down of script via keyboard interrupt

Parameters

- **signal** – the signal received
- **frame** – frame executed in

Returns None

stop(*turn_off_machine=None*, *clear_routing_tables=None*, *clear_tags=None*)

Parameters

- **turn_off_machine** (*bool*) – decides if the machine should be powered down after running the execution. Note that this powers down all boards connected to the BMP connections given to the transceiver
- **clear_routing_tables** (*bool*) – informs the tool chain if it should turn off the clearing of the routing tables

- **clear_tags** (*boolean*) – informs the tool chain if it should clear the tags off the machine at stop

Return type None

timescale_factor

transceiver

update_extra_mapping_inputs(*extra_mapping_inputs*)

use_virtual_board
True if this run is using a virtual machine

verify_not_running()

1.1.3.4 spinn_front_end_common.interface.simulator_state module

class spinn_front_end_common.interface.simulator_state.**Simulator_State**(*value*, *doc=''*)

Bases: enum.Enum

Different States the SimulATOR COULD BE IN

FINISHED = 3

INIT = 0

IN_RUN = 1

RUN_FOREVER = 2

SHUTDOWN = 4

1.1.3.5 Module contents

1.1.4 spinn_front_end_common.mapping_algorithms package

1.1.4.1 Subpackages

spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression package

Submodules

spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_chip_router_comp module

class spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_c
Bases: object

Compressor that uses a on chip router compressor

OVER_RUN_THRESHOLD_BEFORE_ERROR = 1000

SIZE_OF_A_SDRAM_ENTRY = 16

SURPLUS_DATA_ENTRIES = 12

TIME_EXPECTED_TO_RUN = 1000

Module contents

1.1.4.2 Module contents

1.1.5 spinn_front_end_common.utilities package

1.1.5.1 Subpackages

spinn_front_end_common.utilities.connections package

Submodules

spinn_front_end_common.utilities.connections.live_event_connection module

class spinn_front_end_common.utilities.connections.live_event_connection.**LiveEventConnectio**

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection for receiving and sending live events from and to SpiNNaker

Parameters

- **live_packet_gather_label** – The label of the LivePacketGather vertex to which received events are being sent
- **receive_labels** (*iterable of str*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable of str*) – Labels of vertices to which live events will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_init_callback (*label*, *init_callback*)

Add a callback to be called to initialise a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **init_callback** (*function(str, int, float, float) -> None*) – A function to be called to initialise the vertex. This should take as parameters the label of the vertex, the number of neurons in the population, the run time of the simulation in milliseconds, and the simulation timestep in milliseconds

add_pause_stop_callback (*label*, *pause_stop_callback*)
Add a callback for the pause and stop state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **pause_stop_callback** (function(str, `SpynnakerLiveEventConnection`) -> None) – A function to be called when the pause or stop message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events

Return type None

add_receive_callback (*label*, *live_event_callback*)
Add a callback for the reception of live events from a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **live_event_callback** (*function(str, int, [int]) -> None*) – A function to be called when events are received. This should take as parameters the label of the vertex, the simulation timestep when the event occurred, and an array-like of atom ids.

add_start_callback (*label*, *start_callback*)
Add a callback for the start of the simulation

Parameters

- **start_callback** (function(str, `SpynnakerLiveEventConnection`) -> None) – A function to be called when the start message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events
- **label** (*str*) – the label of the function to be sent

add_start_resume_callback (*label*, *start_resume_callback*)

close ()
See `spinnman.connections.abstract_classes.connection.Connection.close()`

send_event (*label*, *atom_id*, *send_full_keys=False*)
Send an event from a single atom

Parameters

- **label** (*str*) – The label of the vertex from which the event will originate
- **atom_id** (*int*) – The id of the atom sending the event
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

send_events (*label*, *atom_ids*, *send_full_keys=False*)
Send a number of events

Parameters

- **label** (*str*) – The label of the vertex from which the events will originate
- **atom_ids** (*[int]*) – array-like of atom ids sending events
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

Module contents

class spinn_front_end_common.utilities.connections.**LiveEventConnection**(*live_packet_gather_label*, *receive_labels=None*, *send_labels=None*, *local_host=None*, *local_port=19999*, *machine_vertices=False*)

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection for receiving and sending live events from and to SpiNNaker

Parameters

- **live_packet_gather_label** – The label of the LivePacketGather vertex to which received events are being sent
- **receive_labels** (*iterable of str*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable of str*) – Labels of vertices to which live events will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_init_callback(*label*, *init_callback*)

Add a callback to be called to initialise a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **init_callback** (*function(str, int, float, float) -> None*) – A function to be called to initialise the vertex. This should take as parameters the label of the vertex, the number of neurons in the population, the run time of the simulation in milliseconds, and the simulation timestep in milliseconds

add_pause_stop_callback(*label*, *pause_stop_callback*)

Add a callback for the pause and stop state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **pause_stop_callback** (function(str, SpynnakerLiveEventConnection) -> None) – A function to be called when the pause or stop message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events

Return type None

add_receive_callback (*label*, *live_event_callback*)
Add a callback for the reception of live events from a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **live_event_callback** (*function(str, int, [int]) -> None*) – A function to be called when events are received. This should take as parameters the label of the vertex, the simulation timestep when the event occurred, and an array-like of atom ids.

add_start_callback (*label*, *start_callback*)
Add a callback for the start of the simulation

Parameters

- **start_callback** (function(str, `SpynnakerLiveEventConnection`) -> None) – A function to be called when the start message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events
- **label** (*str*) – the label of the function to be sent

add_start_resume_callback (*label*, *start_resume_callback*)

close ()
See `spinnman.connections.abstract_classes.connection.Connection.close()`

send_event (*label*, *atom_id*, *send_full_keys=False*)
Send an event from a single atom

Parameters

- **label** (*str*) – The label of the vertex from which the event will originate
- **atom_id** (*int*) – The id of the atom sending the event
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

send_events (*label*, *atom_ids*, *send_full_keys=False*)
Send a number of events

Parameters

- **label** (*str*) – The label of the vertex from which the events will originate
- **atom_ids** (*[int]*) – array-like of atom ids sending events
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

spinn_front_end_common.utilities.database package

Submodules

spinn_front_end_common.utilities.database.database_connection module

class spinn_front_end_common.utilities.database.database_connection.**DatabaseConnection**(*start_ stop_ lo- cal_h lo- cal_p*

Bases: spinnman.connections.udp_packet_connections.udp_connection.UDPConnection, threading.Thread

A connection from the toolchain which will be notified when the database has been written, and can then respond when the database has been read, and further wait for notification that the simulation has started.

Parameters

- **start_resume_callback_function** (*function() -> None*) – A function to be called when the start message has been received. This function should not take any parameters or return anything.
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_database_callback(*database_callback_function*)

Add a database callback to be called when the database is ready

Parameters **database_callback_function** (function(spynnaker_external_devices.pyNN.connections.database_reader.DatabaseReader) -> None) – A function to be called when the database message has been received. This function should take a single parameter, which will be a DatabaseReader object. Once the function returns, it will be assumed that the database has been read, and the return response will be sent.

close()

See spinnman.connections.abstract_classes.connection.Connection.close()

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

spinn_front_end_common.utilities.database.database_reader module

class spinn_front_end_common.utilities.database.database_reader.**DatabaseReader**(*database_path*)

Bases: object

A reader for the database

Parameters **database_path** (*str*) – The path to the database

close()

cursor

The database cursor. Allows custom SQL queries to be performed.

Return type `sqlite3.Cursor`

get_atom_id_to_key_mapping(*label*)
Get a mapping of atom id to event key for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of event keys indexed by atom id

get_configuration_parameter_value(*parameter_name*)
Get the value of a configuration parameter

Parameters **parameter_name** (*str*) – The name of the parameter

Returns The value of the parameter

Return type float

get_key_to_atom_id_mapping(*label*)
Get a mapping of event key to atom id for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of atom ids indexed by event key

Return type dict

get_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_machine_live_input_key(*label*)

get_machine_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_output_key(*label*, *receiver_label*)

get_n_atoms(*label*)
Get the number of atoms in a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utilities.database.database_writer module

class spinn_front_end_common.utilities.database.database_writer.**DatabaseWriter**(*database_directo*
Bases: object

The interface for the database system for main front ends. Any special tables needed from a front end should be done by sub classes of this interface.

add_application_vertices(*application_graph*)

Parameters **application_graph** –

Return type None

add_machine_objects(*machine*)
Store the machine object into the database

Parameters **machine** – the machine object.

Return type None

add_placements(*placements*)
Adds the placements objects into the database

Parameters

- **placements** – the placements object
- **machine_graph** – the machine graph object

Return type None

add_routing_infos(*routing_infos*, *machine_graph*)
Adds the routing information (key masks etc) into the database

Parameters

- **routing_infos** – the routing information object
- **machine_graph** – the machine graph object

Return type None:

add_routing_tables(*routing_tables*)
Adds the routing tables into the database

Parameters **routing_tables** – the routing tables object

Return type None

add_system_params(*time_scale_factor*, *machine_time_step*, *runtime*)
Write system params into the database

Parameters

- **time_scale_factor** – the time scale factor used in timing

- **machine_time_step** – the machine time step used in timing
- **runtime** – the amount of time the application is to run for

add_tags (*machine_graph*, *tags*)
Adds the tags into the database

Parameters

- **machine_graph** – the machine graph object
- **tags** – the tags object

Return type None

add_vertices (*machine_graph*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **graph_mapper** – the graph mapper object
- **application_graph** – the application graph object

Return type None

static auto_detect_database (*machine_graph*)
Auto detects if there is a need to activate the database system

Parameters **machine_graph** – the machine graph of the application problem space.

Returns a bool which represents if the database is needed

create_atom_to_event_id_mapping (*application_graph*, *machine_graph*, *routing_infos*, *graph_mapper*)

Parameters

- **application_graph** –
- **machine_graph** –
- **routing_infos** –
- **graph_mapper** –

Return type None

create_schema ()

database_path

Module contents

class spinn_front_end_common.utilities.database.**DatabaseConnection** (*start_resume_callback_function=N stop_pause_callback_function=No local_host=None, local_port=19999*)

Bases: spinnman.connections.udp_packet_connections.udp_connection.UDPConnection, threading.Thread

A connection from the toolchain which will be notified when the database has been written, and can then respond when the database has been read, and further wait for notification that the simulation has started.

Parameters

- **start_resume_callback_function** (*function() -> None*) – A function to be called when the start message has been received. This function should not take any parameters or return anything.
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_database_callback(*database_callback_function*)

Add a database callback to be called when the database is ready

Parameters **database_callback_function** (function(`spynnaker_external_devices.pyNN.connections.database_reader.DatabaseReader`) -> None) – A function to be called when the database message has been received. This function should take a single parameter, which will be a DatabaseReader object. Once the function returns, it will be assumed that the database has been read, and the return response will be sent.

close()

See `spinnman.connections.abstract_classes.connection.Connection.close()`

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class spinn_front_end_common.utilities.database.**DatabaseReader**(*database_path*)

Bases: `object`

A reader for the database

Parameters **database_path** (*str*) – The path to the database

close()

cursor

The database cursor. Allows custom SQL queries to be performed.

Return type `sqlite3.Cursor`

get_atom_id_to_key_mapping(*label*)

Get a mapping of atom id to event key for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of event keys indexed by atom id

get_configuration_parameter_value(*parameter_name*)

Get the value of a configuration parameter

Parameters **parameter_name** (*str*) – The name of the parameter

Returns The value of the parameter

Return type float

get_key_to_atom_id_mapping(*label*)
Get a mapping of event key to atom id for a given vertex

Parameters **label** (`str`) – The label of the vertex

Returns dictionary of atom ids indexed by event key

Return type dict

get_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given vertex

Parameters **label** (`str`) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a vertex

Parameters **label** (`str`) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given machine vertex

Parameters **label** (`str`) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_machine_live_input_key(*label*)

get_machine_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a machine vertex

Parameters **label** (`str`) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_output_key(*label*, *receiver_label*)

get_n_atoms(*label*)
Get the number of atoms in a given vertex

Parameters **label** (`str`) – The label of the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utilities.database.**DatabaseWriter**(*database_directory*)
Bases: `object`

The interface for the database system for main front ends. Any special tables needed from a front end should be done by sub classes of this interface.

add_application_vertices(*application_graph*)

Parameters **application_graph** –

Return type None

add_machine_objects (*machine*)
Store the machine object into the database

Parameters **machine** – the machine object.

Return type None

add_placements (*placements*)
Adds the placements objects into the database

Parameters

- **placements** – the placements object
- **machine_graph** – the machine graph object

Return type None

add_routing_infos (*routing_infos*, *machine_graph*)
Adds the routing information (key masks etc) into the database

Parameters

- **routing_infos** – the routing information object
- **machine_graph** – the machine graph object

Return type None:

add_routing_tables (*routing_tables*)
Adds the routing tables into the database

Parameters **routing_tables** – the routing tables object

Return type None

add_system_params (*time_scale_factor*, *machine_time_step*, *runtime*)
Write system params into the database

Parameters

- **time_scale_factor** – the time scale factor used in timing
- **machine_time_step** – the machine time step used in timing
- **runtime** – the amount of time the application is to run for

add_tags (*machine_graph*, *tags*)
Adds the tags into the database

Parameters

- **machine_graph** – the machine graph object
- **tags** – the tags object

Return type None

add_vertices (*machine_graph*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **graph_mapper** – the graph mapper object

- **application_graph** – the application graph object

Return type None

static auto_detect_database(*machine_graph*)
Auto detects if there is a need to activate the database system

Parameters **machine_graph** – the machine graph of the application problem space.

Returns a bool which represents if the database is needed

create_atom_to_event_id_mapping(*application_graph*, *machine_graph*, *routing_infos*, *graph_mapper*)

Parameters

- **application_graph** –
- **machine_graph** –
- **routing_infos** –
- **graph_mapper** –

Return type None

create_schema()

database_path

spinn_front_end_common.utilities.notification_protocol package

Submodules

spinn_front_end_common.utilities.notification_protocol.notification_protocol module

class spinn_front_end_common.utilities.notification_protocol.notification_protocol.**Notifica**

Bases: object

The protocol which hand shakes with external devices about the database and starting execution

close()
Closes the thread pool

send_read_notification(*database_path*)
sends notifications to all devices which have expressed an interest in when the database has been written

Parameters **database_path** – the path to the database file

Return type None

send_start_resume_notification()
either waits till all sources have confirmed read the database and are configured, and/or just sends the start notification (when the system is executing)

Return type None

send_stop_pause_notification()
sends the pause / stop notifications when the script has either finished or paused

Return type None

wait_for_confirmation()
if asked to wait for confirmation, waits for all external systems to confirm that they are configured and have read the database

Return type None

spinn_front_end_common.utilities.notification_protocol.socket_address module

class spinn_front_end_common.utilities.notification_protocol.socket_address.**SocketAddress**(*n*
n
t
l
t

Bases: object

Data holder for a socket interface for notification protocol.

listen_port
The port to listen to for responses

notify_host_name
The notify host name

notify_port_no
The notify port no

Module contents

class spinn_front_end_common.utilities.notification_protocol.**NotificationProtocol**(*socket_addre*
wait_for_rea

Bases: object

The protocol which hand shakes with external devices about the database and starting execution

close()
Closes the thread pool

send_read_notification(*database_path*)
sends notifications to all devices which have expressed an interest in when the database has been written

Parameters **database_path** – the path to the database file

Return type None

send_start_resume_notification()
either waits till all sources have confirmed read the database and are configured, and/or just sends the start notification (when the system is executing)

Return type None

send_stop_pause_notification()
sends the pause / stop notifications when the script has either finished or paused

Return type None

wait_for_confirmation()
if asked to wait for confirmation, waits for all external systems to confirm that they are configured and have read the database

Return type None

class spinn_front_end_common.utilities.notification_protocol.**SocketAddress**(*notify_host_name*, *notify_port_no*, *listen_port*)

Bases: object

Data holder for a socket interface for notification protocol.

listen_port
The port to listen to for responses

notify_host_name
The notify host name

notify_port_no
The notify port no

spinn_front_end_common.utilities.report_functions package

Submodules

spinn_front_end_common.utilities.report_functions.energy_report module

class spinn_front_end_common.utilities.report_functions.energy_report.**EnergyReport**
Bases: object

ENERGY_DETAILED_FILENAME = 'Detailed_energy_report.rpt'

ENERGY_SUMMARY_FILENAME = 'energy_summary_report.rpt'

JULES_PER_MILLISECOND_FOR_BOXED_48_CHIP_FRAME_IDLE_COST = 0.0045833333

JULES_PER_MILLISECOND_FOR_FRAME_IDLE_COST = 0.117

JULES_PER_MILLISECOND_PER_CHIP_ACTIVE_OVERHEAD = 0.0006399999999999999

JULES_PER_MILLISECOND_PER_FPGA = 0.000584635

JULES_PER_MILLISECOND_PER_FRAME_ACTIVE_COST = 0.154163558

JULES_PER_MILLISECOND_PER_IDLE_CHIP = 0.00036

JULES_PER_MILLISECOND_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST = 0.01666667

JULES_PER_SPIKE = 8e-10

JULES_TO_KILLIWATT_HOURS = 3600000

N_MONITORS_ACTIVE_DURING_COMMS = 2

spinn_front_end_common.utilities.report_functions.memory_map_on_chip_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_chip_report.**MemoryMap**
Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.memory_map_on_host_chip_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_host_chip_report.**Memo**

Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.memory_map_on_host_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_host_report.**MemoryMap**

Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.routing_table_from_machine_report module

class spinn_front_end_common.utilities.report_functions.routing_table_from_machine_report.**I**

Bases: object

Module contents

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnChipReport**

Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnHostChipReport**

Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnHostReport**

Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**RoutingTableFromMachineReport**

Bases: object

spinn_front_end_common.utilities.scp package

Submodules

spinn_front_end_common.utilities.scp.clear_iobuf_process module

class spinn_front_end_common.utilities.scp.clear_iobuf_process.**ClearIOBUFProcess**(*connection_se*

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

clear_iobuf(*core_subsets*, *n_cores*)

receive_response(*response*)

spinn_front_end_common.utilities.scp.scp_clear_iobuf_request module

class spinn_front_end_common.utilities.scp.scp_clear_iobuf_request.**SCPClearIOBUFRequest**(*x*, *y*, *p*, *destination*, *expec*

Bases: spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

spinn_front_end_common.utilities.scp.scp_update_runtime_request module

class spinn_front_end_common.utilities.scp.scp_update_runtime_request.**SCPUpdateRuntimeReque**

Bases: spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

spinn_front_end_common.utilities.scp.update_runtime_process module

class spinn_front_end_common.utilities.scp.update_runtime_process.**UpdateRuntimeProcess**(*conn*

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

receive_response(*response*)

update_runtime (*run_time*, *infinite_run*, *core_subsets*, *n_cores*)

Module contents

class spinn_front_end_common.utilities.scp.**ClearIOBUFProcess** (*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

clear_iobuf (*core_subsets*, *n_cores*)

receive_response (*response*)

class spinn_front_end_common.utilities.scp.**SCPClearIOBUFRequest** (*x*, *y*, *p*, *destination_port*, *expect_response=True*)

Bases: spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest

get_scp_response ()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

class spinn_front_end_common.utilities.scp.**SCPUpdateRuntimeRequest** (*x*, *y*, *p*, *run_time*, *infinite_run*, *destination_port*, *expect_response=True*)

Bases: spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest

get_scp_response ()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

class spinn_front_end_common.utilities.scp.**UpdateRuntimeProcess** (*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

receive_response (*response*)

update_runtime (*run_time*, *infinite_run*, *core_subsets*, *n_cores*)

spinn_front_end_common.utilities.utility_objs package

Submodules

spinn_front_end_common.utilities.utility_objs.executable_finder module

class spinn_front_end_common.utilities.utility_objs.executable_finder.**ExecutableFinder**(*binar in- clude*

Bases: spinn_utilities.executable_finder.ExecutableFinder

Manages a set of folders in which to search for binaries, and allows for binaries to be discovered within this path. This adds a default location to look to the base class.

Parameters

- **binary_search_paths** (*iterable of str*) – The initial set of folders to search for binaries.
- **include_common_binaries_folder** (*bool*) – If True (i.e. the default), the spinn_front_end_common.common_model_binaries folder is searched for binaries. If you are not using the common models, or the model binary names conflict with your own, this parameter can be used to avoid this issue. Note that the folder will be appended to the value of binary_search_paths, so the common binary search path will be looked in last.

spinn_front_end_common.utilities.utility_objs.executable_start_type module

class spinn_front_end_common.utilities.utility_objs.executable_start_type.**ExecutableStartTy**

Bases: enum.Enum

supports starting of different types of executables

NO_APPLICATION = 3

RUNNING = 0

SYNC = 1

USES_SIMULATION_INTERFACE = (2, 'Calls simulation_run() and simulation_exit() / simula

spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters module

class spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.**LivePacke**

Bases: object

parameter holder for LPG's so that they can be instantiated at a later date.

board_address

hostname

key_prefix

message_type

number_of_packets_sent_per_time_step

partition_id

payload_as_time_stamps

payload_prefix

payload_right_shift

port

prefix_type

right_shift

strip_sdp

tag

use_payload_prefix

use_prefix

spinn_front_end_common.utilities.utility_objs.provenance_data_item module

class spinn_front_end_common.utilities.utility_objs.provenance_data_item.**ProvenanceDataIten**

Bases: object

Container for provenance data

Parameters

- **names** – A list of strings representing the hierarchy of naming of this item
- **value** – The value of the item
- **report** – True if the item should be reported to the user
- **message** – The message to send to the end user if report is True

message
The message to report to the end user, or None if no message

names
The hierarchy of names of this bit of provenance data

report
True if this provenance data entry needs reporting to the end user

value
The value of the item

Module contents

class spinn_front_end_common.utilities.utility_objs.**ExecutableFinder**(*binary_search_paths=None, include_common_binaries_folder*

Bases: spinn_utilities.executable_finder.ExecutableFinder

Manages a set of folders in which to search for binaries, and allows for binaries to be discovered within this path. This adds a default location to look to the base class.

Parameters

- **binary_search_paths** (*iterable of str*) – The initial set of folders to search for binaries.
- **include_common_binaries_folder** (*bool*) – If True (i.e. the default), the spinn_front_end_common.common_model_binaries folder is searched for binaries. If you are not using the common models, or the model binary names conflict with your own, this parameter can be used to avoid this issue. Note that the folder will be appended to the value of binary_search_paths, so the common binary search path will be looked in last.

class spinn_front_end_common.utilities.utility_objs.**ExecutableStartType**(*value, doc=''*)

Bases: enum.Enum

supports starting of different types of executables

NO_APPLICATION = 3

RUNNING = 0

SYNC = 1

USES_SIMULATION_INTERFACE = (2, 'Calls simulation_run() and simulation_exit() / simula

class spinn_front_end_common.utilities.utility_objs.**LivePacketGatherParameters**(*port, hostname, tag, board_address, strip_sdp, use_prefix, key_prefix, prefix_type, message_type, right_shift, payload_as_time_sta use_payload_pre payload_prefix, payload_right_shift, number_of_packets_s partition_id*)

Bases: object

parameter holder for LPG's so that they can be instantiated at a later date.

board_address

hostname

key_prefix

message_type

number_of_packets_sent_per_time_step

partition_id

payload_as_time_stamps

payload_prefix

payload_right_shift

port

prefix_type

right_shift

strip_sdp

tag

use_payload_prefix

use_prefix

class spinn_front_end_common.utilities.utility_objs.**ProvenanceDataItem**(*names*, *value*, *report=False*, *message=None*)

Bases: object

Container for provenance data

Parameters

- **names** – A list of strings representing the hierarchy of naming of this item
- **value** – The value of the item
- **report** – True if the item should be reported to the user
- **message** – The message to send to the end user if report is True

message
The message to report to the end user, or None if no message

names
The hierarchy of names of this bit of provenance data

report
True if this provenance data entry needs reporting to the end user

value
The value of the item

1.1.5.2 Submodules

1.1.5.3 spinn_front_end_common.utilities.constants module

class spinn_front_end_common.utilities.constants.**BUFFERING_OPERATIONS**
Bases: enum.Enum

BUFFER_READ = 0

BUFFER_WRITE = 1

class spinn_front_end_common.utilities.constants.**SDP_PORTS**
Bases: enum.Enum

INPUT_BUFFERING_SDP_PORT = 1

OUTPUT_BUFFERING_SDP_PORT = 2

RUNNING_COMMAND_SDP_PORT = 3

spinn_front_end_common.utilities.constants.**SDP_RUNNING_MESSAGE_CODES**
alias of spinn_front_end_common.utilities.constants.SDP_RUNNING_MESSAGE_ID_CODES

1.1.5.4 spinn_front_end_common.utilities.exceptions module

exception spinn_front_end_common.utilities.exceptions.**BufferableRegionTooSmall**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when the SDRAM space of the region for buffered packets is too small to contain any packet at all

exception spinn_front_end_common.utilities.exceptions.**BufferedRegionNotPresent**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when trying to issue buffered packets for a region not managed

exception spinn_front_end_common.utilities.exceptions.**ConfigurationException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when the front end determines a input param is invalid

exception spinn_front_end_common.utilities.exceptions.**ExecutableFailedToStartException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when an executable has not entered the expected state during start up

exception spinn_front_end_common.utilities.exceptions.**ExecutableFailedToStopException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when an executable has not entered the expected state during execution

exception spinn_front_end_common.utilities.exceptions.**ExecutableNotFoundException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when a specified executable could not be found

exception spinn_front_end_common.utilities.exceptions.**RallocException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when there are not enough routing table entries

exception spinn_front_end_common.utilities.exceptions.**SpinnFrontEndException**
Bases: exceptions.Exception

Raised when the front end detects an error

1.1.5.5 spinn_front_end_common.utilities.failed_state module

class spinn_front_end_common.utilities.failed_state.**FailedState**
Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

static add_socket_address (*self*, *x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run (*run_time*)

stop()

transceiver

use_virtual_board

verify_not_running()

1.1.5.6 spinn_front_end_common.utilities.function_list module

spinn_front_end_common.utilities.function_list.**get_front_end_common_pacman_xml_paths**()
Get the XML path for the front end common interface functions

1.1.5.7 spinn_front_end_common.utilities.globals_variables module

spinn_front_end_common.utilities.globals_variables.**get_not_running_simulator**()

spinn_front_end_common.utilities.globals_variables.**get_simulator**()

spinn_front_end_common.utilities.globals_variables.**has_simulator**()

spinn_front_end_common.utilities.globals_variables.**set_failed_state**(*new_failed_state*)

spinn_front_end_common.utilities.globals_variables.**set_simulator**(*new_simulator*)

spinn_front_end_common.utilities.globals_variables.**unset_simulator**()

1.1.5.8 spinn_front_end_common.utilities.helpful_functions module

spinn_front_end_common.utilities.helpful_functions.**convert_string_into_chip_and_core_subset**
Translate a string list of cores into a core subset

Parameters **cores** (*str or None*) – string representing down cores formatted as x,y,p[:x,y,p]*

spinn_front_end_common.utilities.helpful_functions.**convert_time_diff_to_total_milliseconds**
converts between a time diff and total milliseconds

Returns total milliseconds

spinn_front_end_common.utilities.helpful_functions.**generate_unique_folder_name**(*folder*, *filename*, *extension*)

Generate a unique file name with a given extension in a given folder

Parameters

- **folder** – where to put this unique file
- **filename** – the name of the first part of the file without extension
- **extension** – extension of the file

Returns file path with a unique addition

spinn_front_end_common.utilities.helpful_functions.**get_ethernet_chip**(*machine*, *board_address*)

locate the chip with the given board IP address

Parameters

- **machine** – the spinnaker machine
- **board_address** – the board address to locate the chip of.

Returns The chip that supports that board address

Raises *ConfigurationException* – when that board address has no chip associated with it

spinn_front_end_common.utilities.helpful_functions.**locate_memory_region_for_placement**(*placement, region, transceiver*)

Get the address of a region for a placement

Parameters

- **region** (*int*) – the region to locate the base address of
- **placement** (*pacman.model.placements.Placement*) – the placement object to get the region address of
- **transceiver** (*spiNNMan.transciever.Transciever*) – the python interface to the spinnaker machine

spinn_front_end_common.utilities.helpful_functions.**read_config**(*config, section, item*)

Get the string value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_config_boolean**(*config, section, item*)

Get the boolean value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_config_int**(*config, section, item*)

Get the integer value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_data**(*x, y, address, length, data_format, transceiver*)

Reads and converts a single data item from memory

Parameters

- **x** – chip x
- **y** – chip y
- **address** – base address of the SDRAM chip to read
- **length** – length to read
- **data_format** – the format to read memory
- **transceiver** – the SpinnMan interface

spinn_front_end_common.utilities.helpful_functions.**set_up_output_application_data_specifics**

Parameters

- **where_to_write_application_data_files** – the location where all app data is by default written to
- **max_application_binaries_kept** – The max number of report folders to keep active at any one time
- **n_calls_to_run** – the counter of how many times run has been called.
- **this_run_time_string** – the time stamp string for this run

Returns the run folder for this simulation to hold app data

spinn_front_end_common.utilities.helpful_functions.**set_up_report_specifics**(*default_report_file_pat max_reports_kept, n_calls_to_run, this_run_time_string=*

Parameters

- **default_report_file_path** – The location where all reports reside
- **max_reports_kept** – The max number of report folders to keep active at any one time
- **n_calls_to_run** – the counter of how many times run has been called.
- **this_run_time_string** – holder for the timestamp for future runs

Returns The folder for this run, the time_stamp

spinn_front_end_common.utilities.helpful_functions.**sort_out_downed_chips_cores_links**(*downed downed downed*

Translate the down cores and down chips string into a form that spinnman can understand

Parameters

- **downed_cores** (*str or None*) – string representing down cores formatted as x,y,p[:x,y,p]*
- **downed_chips** (*str or None*) – string representing down chips formatted as x,y[:x,y]*
- **downed_links** – string representing down links formatted as x,y,link[:x,y,link]*

Returns a tuple of (set of (x, y) of down chips, set of (x, y, p) of down cores, set of ((x, y), link id) of down links)

Return type ({(int, int,), }, {(int, int, int), }, {((int, int), int), })

spinn_front_end_common.utilities.helpful_functions.**translate_iobuf_extraction_elements**(*hard hard executab executab*

Parameters

- **hard_coded_cores** – list of cores to read iobuf from
- **hard_coded_model_binary** – list of binary names to read iobuf from
- **executable_targets** – the targets of cores and executable binaries

- **executable_finder** – where to find binaries paths from binary names

Returns core subsets for the cores to read iobuf from

spinn_front_end_common.utilities.helpful_functions.**write_finished_file**(*app_data_runtime_folder*, *report_default_directory*)

1.1.5.9 spinn_front_end_common.utilities.math_constants module

random math constants

1.1.5.10 spinn_front_end_common.utilities.simulator_interface module

class spinn_front_end_common.utilities.simulator_interface.**SimulatorInterface**

Bases: object

add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run

stop()

transceiver

use_virtual_board

verify_not_running()

1.1.5.11 Module contents

class spinn_front_end_common.utilities.**FailedState**

Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

static add_socket_address(*self*, *x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run(*run_time*)

stop()

transceiver

use_virtual_board

verify_not_running()

class spinn_front_end_common.utilities.**SimulatorInterface**
Bases: object

add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run

stop()

transceiver

use_virtual_board

verify_not_running()

1.1.6 spinn_front_end_common.utility_models package

1.1.6.1 Submodules

1.1.6.2 spinn_front_end_common.utility_models.chip_power_monitor_application_vertex module

class spinn_front_end_common.utility_models.chip_power_monitor_application_vertex.**ChipPowe**

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

class for representing idle time recording code in a application graph.

chip power monitor application vertex constructor

Parameters

- **label** – vertex label
- **constraints** – constraints for the vertex
- **n_samples_per_recording** – how many samples to take before / recording to sdram the total
- **sampling_frequency** – how many microseconds between sampling

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_resources_used_by_atoms (*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

1.1.6.3 spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex module

class spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.**ChipPowerMon:**

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*

machine vertex for c code representing functionality to record / idle times in a machine graph

class CHIP_POWER_MONITOR_REGIONS

Bases: enum.Enum

CONFIG = 1

RECORDING = 2

SYSTEM = 0

CONFIG_SIZE_IN_BYTES = 8

DEFAULT_MALLOCS_USED = 3

MAX_BUFFER_SIZE = 1048576

MAX_CORES_PER_CHIP = 18

RECORDING_SIZE_PER_ENTRY = 72

SAMPLE_RECORDING_REGION = 0

static binary_file_name ()

returns the string binary file name

Returns basestring

static binary_start_type ()

static method to allow app verts to use this

Returns starttype enum

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_minimum_buffer_sdram_usage(**args*, ***kwargs*)
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(**args*, ***kwargs*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_data(*placement*, *buffer_manager*)
get data from sdram given placement and buffer manager

Parameters

- **placement** – the location on machien to get data from
- **buffer_manager** – the buffer manager that might have data

Returns results

Return type numpy array with 1 dimension

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance

- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static get_resources (*n_machine_time_steps*, *time_step*, *time_scale_factor*, *n_samples_per_recording*, *sampling_frequency*)
get resources used by this vertex

:return:Resource container

n_samples_per_recording

resources_required
The resources required by the vertex

Return type `pacman.model.resources.ResourceContainer`

sampling_frequency

static sdram_calculation ()
calculates the sdram requirements of the vertex

Returns int

1.1.6.4 spinn_front_end_common.utility_models.command_sender module

class spinn_front_end_common.utility_models.command_sender.**CommandSender** (*label*, *constraints*)
Bases: `pacman.model.graphs.application.application_vertex.ApplicationVertex`, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*

A utility for sending commands to a vertex (possibly an external device) at fixed times in the simulation

add_commands (*start_resume_commands*, *pause_stop_commands*, *timed_commands*, *vertex_to_send_to*)
Add commands to be sent down a given edge

Parameters

- **start_resume_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send when the simulation starts or resumes from pause
- **pause_stop_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – the commands to send when the simulation stops or pauses after running
- **timed_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send at specific times
- **vertex_to_send_to** – The vertex these commands are to be sent to

create_machine_vertex (*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

dependent_vertices()
Return the vertices which this vertex depends upon

edges_and_partitions()

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str Return a string representation of the models binary

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters partition – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters vertex_slice (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises None – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

1.1.6.5 spinn_front_end_common.utility_models.command_sender_machine_vertex module

class spinn_front_end_common.utility_models.command_sender_machine_vertex.**CommandSenderMacl**

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS

Bases: enum.Enum

COMMANDS_AT_START_RESUME = 3

COMMANDS_AT_STOP_PAUSE = 4

COMMANDS_WITH_ARBITRARY_TIMES = 2

PROVENANCE_REGION = 5

SETUP = 1

SYSTEM_REGION = 0

TOTAL_REQUIRED_MALLOCS = 5

generate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *n_machine_time_steps*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_n_command_bytes(*commands*)

static get_number_of_mallocs_used_by_dsg()

static get_timed_commands_bytes(*timed_commands*)

resources_required

The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

1.1.6.6 spinn_front_end_common.utility_models.live_packet_gather module

class spinn_front_end_common.utility_models.live_packet_gather.**LivePacketGather**(*hostname, port, board_address=*
tag=None, strip_sdp=True
use_prefix=Fals
key_prefix=Non
prefix_type=None, message_type=<EI
2>, right_shift=0, payload_as_time_s
use_payload_pr
payload_prefix=No
payload_right_shif
*number_of_packets_*
constraints=None, label=None)

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, pacman.model.graphs.application.application_vertex.ApplicationVertex

A model which stores all the events it receives during a timer tick and then compresses them into Ethernet packets and sends them out of a spinnaker machine.

create_machine_vertex(*vertex_slice, resources_required, label=None, constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*spec, placement*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

1.1.6.7 spinn_front_end_common.utility_models.live_packet_gather_machine_vertex module

class spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.**LivePacketGat**

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*

N_ADDITIONAL_PROVENANCE_ITEMS = 2

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_cpu_usage()
Get the CPU used by this vertex

Returns 0

Return type int

static get_dtcm_usage()
Get the DTCM used by this vertex

get_provenance_data_from_machine(*transceiver*, *placement*)
Get provenance from the machine

Parameters

- **transceiver** – spinnman interface to the machine
- **placement** – the location of this vertex on the machine

static get_sdram_usage()
Get the SDRAM used by this vertex

is_in_injection_mode
Whether this vertex is actually in injection mode.

resources_required
The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

1.1.6.8 spinn_front_end_common.utility_models.multi_cast_command module

class spinn_front_end_common.utility_models.multi_cast_command.**MultiCastCommand**(*key*, *payload=None*, *time=None*, *repeat=0*, *delay_between_re*

Bases: object

A command to be sent to a vertex

Parameters

- **key** (*int*) – The key of the command
- **payload** (*int*) – The payload of the command
- **time** (*int*) – The time within the simulation at which to send the command, or None if this is not a timed command
- **repeat** (*int*) – The number of times that the command should be repeated after sending it once. This could be used to ensure that the command is sent despite lost packets. Must be between 0 and 65535
- **delay_between_repeats** (*int*) – The amount of time in micro seconds to wait between sending repeats of the same command. Must be between 0 and 65535, and must be 0 if repeat is 0

Raises `SpynnakerException` – If the repeat or delay are out of range

delay_between_repeats

is_payload

Determine if this command has a payload. By default, this returns True if the payload passed in to the constructor is not None, but this can be overridden to indicate that a payload will be generated, despite None being passed to the constructor

Returns True if there is a payload, False otherwise

Return type bool

is_timed

key

payload

Get the payload of the command.

Returns The payload of the command, or None if there is no payload

Return type int

repeat

time

1.1.6.9 spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source module

class spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.**ReverseIpTagMu**

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons* *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A model which will allow events to be injected into a spinnaker machine and converted into multicast packets.

Parameters

- **n_keys** – The number of keys to be sent via this multicast source
- **label** – The label of this vertex
- **constraints** – Any initial constraints to this vertex
- **board_address** – The IP address of the board on which to place this vertex if receiving data, either buffered or live (by default, any board is chosen)
- **receive_port** – The port on the board that will listen for incoming event packets (default is to disable this feature; set a value to enable it)

- **receive_sdp_port** – The SDP port to listen on for incoming event packets (defaults to 1)
- **receive_tag** – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** – The estimated rate of packets that will be sent by this source
- **virtual_key** – The base multicast key to send received events with (assigned automatically by default)
- **prefix** – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** – An array of arrays of times at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** – The id of the partition containing the edges down which the events are to be sent
- **send_buffer_max_space** – The maximum amount of space to use of the SDRAM on the machine (default is 1MB)
- **send_buffer_space_before_notify** – The amount of space free in the sending buffer before the machine will ask the host for more data (default setting is optimised for most cases)
- **buffer_notification_ip_address** – The IP address of the host that will send new buffers (must be specified if a send buffer is specified or if recording will be used)
- **buffer_notification_port** – The port that the host that will send new buffers is listening on (must be specified if a send buffer is specified, or if recording will be used)
- **buffer_notification_tag** – The IP tag to use to notify the host about space in the buffer (default is to use any tag)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_requests=0*)

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (`data_specification.DataSpecificationGenerator`) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

send_buffer_times

1.1.6.10 spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex module

class spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.*

AbstractReceiveBuffersToHost, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

A model which allows events to be injected into spinnaker and converted in to multicast packets

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_triggers=0*)
Enable recording of the keys sent

Parameters

- **record_buffer_size** (*int*) – The size of the recording buffer in bytes. Note that when using automatic pause and resume, this will be used as the minimum size of the buffer
- **buffer_size_before_receive** (*int*) – The size that the buffer can grow to before a read request is issued to the host (in bytes)
- **time_between_triggers** (*int*) – The minimum time between the sending of read requests

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_cpu_usage()

static get_dtcm_usage()

get_minimum_buffer_sdram_usage()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

get_provenance_data_from_machine(*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static get_sdram_usage(*send_buffer_times*, *send_buffer_max_space*, *recording_enabled*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static n_regions_to_allocate(*send_buffering*, *recording*)
Get the number of regions that will be allocated

resources_required
The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

send_buffer_times

send_buffers

update_buffer(*arg*)

virtual_key

1.1.6.11 Module contents

class spinn_front_end_common.utility_models.**CommandSender**(*label*, *constraints*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*

A utility for sending commands to a vertex (possibly an external device) at fixed times in the simulation

add_commands(*start_resume_commands*, *pause_stop_commands*, *timed_commands*, *vertex_to_send_to*)

Add commands to be sent down a given edge

Parameters

- **start_resume_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send when the simulation starts or resumes from pause
- **pause_stop_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – the commands to send when the simulation stops or pauses after running
- **timed_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send at specific times
- **vertex_to_send_to** – The vertex these commands are to be sent to

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

dependent_vertices()

Return the vertices which this vertex depends upon

edges_and_partitions()

generate_data_specification(**args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str Return a string representation of the models binary

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utility_models.**CommandSenderMachineVertex**(*constraints*, *resources_required*, *label*, *commands_at_start_resume*, *commands_at_pause_stop*, *timed_commands*)
Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS
Bases: enum.Enum

COMMANDS_AT_START_RESUME = 3

COMMANDS_AT_STOP_PAUSE = 4

COMMANDS_WITH_ARBITRARY_TIMES = 2

PROVENANCE_REGION = 5

SETUP = 1

SYSTEM_REGION = 0

TOTAL_REQUIRED_MALLOCS = 5

generate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *n_machine_time_steps*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_n_command_bytes(*commands*)

static get_number_of_mallocs_used_by_dsg()

static get_timed_commands_bytes(*timed_commands*)

resources_required

The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

class spinn_front_end_common.utility_models.**LivePacketGather**(*hostname*, *port*, *board_address=None*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*, *constraints=None*, *label=None*)

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, pacman.model.graphs.application.application_vertex.ApplicationVertex

A model which stores all the events it receives during a timer tick and then compresses them into Ethernet packets and sends them out of a spinnaker machine.

create_machine_vertex (*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification (*spec*, *placement*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name ()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type ()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_resources_used_by_atoms (*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters vertex_slice (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises None – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utility_models.**LivePacketGatherMachineVertex**(*label*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType 2>*, *right_shift=0*, *payload_as_time_stamps=T*, *use_payload_prefix=Tru*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_pe*, *hostname=None*, *port=None*, *strip_sdp=None*, *board_address=None*, *tag=None*, *constraints=None*)

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*

N_ADDITIONAL_PROVENANCE_ITEMS = 2

generate_data_specification(**args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_cpu_usage()
Get the CPU used by this vertex

Returns 0

Return type int

static get_dtcm_usage()
Get the DTCM used by this vertex

get_provenance_data_from_machine(*transceiver*, *placement*)
Get provenance from the machine

Parameters

- **transceiver** – spinnman interface to the machine
- **placement** – the location of this vertex on the machine

static get_sdram_usage()
Get the SDRAM used by this vertex

is_in_injection_mode
Whether this vertex is actually in injection mode.

resources_required
The resources required by the vertex

Return type `pacman.model.resources.ResourceContainer`

class spinn_front_end_common.utility_models.**MultiCastCommand**(*key*, *payload=None*, *time=None*, *repeat=0*, *delay_between_repeats=0*)

Bases: `object`

A command to be sent to a vertex

Parameters

- **key** (*int*) – The key of the command
- **payload** (*int*) – The payload of the command
- **time** (*int*) – The time within the simulation at which to send the command, or None if this is not a timed command
- **repeat** (*int*) – The number of times that the command should be repeated after sending it once. This could be used to ensure that the command is sent despite lost packets. Must be between 0 and 65535
- **delay_between_repeats** (*int*) – The amount of time in micro seconds to wait between sending repeats of the same command. Must be between 0 and 65535, and must be 0 if repeat is 0

Raises **SpynnakerException** – If the repeat or delay are out of range

delay_between_repeats

is_payload
Determine if this command has a payload. By default, this returns True if the payload passed in to the constructor is not None, but this can be overridden to indicate that a payload will be generated, despite None being passed to the constructor

Returns True if there is a payload, False otherwise

Return type bool

is_timed

key

payload
Get the payload of the command.

Returns The payload of the command, or None if there is no payload

Return type int

repeat

time

class spinn_front_end_common.utility_models.**ReverseIpTagMultiCastSource**(*n_keys*, *label=None*, *constraints=None*, *max_atoms_per_core=9223*, *board_address=None*, *receive_port=None*, *receive_sdp_port=1*, *receive_tag=None*, *receive_rate=10*, *virtual_key=None*, *prefix=None*, *prefix_type=None*, *check_keys=False*, *send_buffer_times=None*, *send_buffer_partition_id=N*, *send_buffer_max_space=1*, *send_buffer_space_before_*, *buffer_notification_ip_addr*, *buffer_notification_port=N*, *buffer_notification_tag=No*, *reserve_reverse_ip_tag=Fals*

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A model which will allow events to be injected into a spinnaker machine and converted into multicast packets.

Parameters

- **n_keys** – The number of keys to be sent via this multicast source
- **label** – The label of this vertex
- **constraints** – Any initial constraints to this vertex
- **board_address** – The IP address of the board on which to place this vertex if receiving data, either buffered or live (by default, any board is chosen)
- **receive_port** – The port on the board that will listen for incoming event packets (default is to disable this feature; set a value to enable it)
- **receive_sdp_port** – The SDP port to listen on for incoming event packets (defaults to 1)
- **receive_tag** – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** – The estimated rate of packets that will be sent by this source
- **virtual_key** – The base multicast key to send received events with (assigned automatically by default)
- **prefix** – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** – An array of arrays of times at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** – The id of the partition containing the edges down which the events are to be sent
- **send_buffer_max_space** – The maximum amount of space to use of the SDRAM on the machine (default is 1MB)
- **send_buffer_space_before_notify** – The amount of space free in the sending buffer before the machine will ask the host for more data (default setting is optimised for most cases)
- **buffer_notification_ip_address** – The IP address of the host that will send new buffers (must be specified if a send buffer is specified or if recording will be used)
- **buffer_notification_port** – The port that the host that will send new buffers is listening on (must be specified if a send buffer is specified, or if recording will be used)
- **buffer_notification_tag** – The IP tag to use to notify the host about space in the buffer (default is to use any tag)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_requests=0*)

generate_data_specification(*spec*, *placement*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters partition – An edge that comes out of this vertex

Returns A list of constraints

Return type list of pacman.model.constraints.abstract_constraint.AbstractConstraint

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters vertex_slice (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type pacman.model.resources.ResourceContainer

Raises None – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

send_buffer_times

class spinn_front_end_common.utility_models.**ReverseIPTagMulticastSourceMachineVertex**(**args*, ***kwargs*)

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.*

abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints, *spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

A model which allows events to be injected into spinnaker and converted in to multicast packets

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_triggers=0*)
Enable recording of the keys sent

Parameters

- **record_buffer_size** (*int*) – The size of the recording buffer in bytes. Note that when using automatic pause and resume, this will be used as the minimum size of the buffer
- **buffer_size_before_receive** (*int*) – The size that the buffer can grow to before a read request is issued to the host (in bytes)
- **time_between_triggers** (*int*) – The minimum time between the sending of read requests

generate_data_specification(**args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type *spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableStartType*

static get_cpu_usage()

static get_dtcm_usage()

get_minimum_buffer_sdram_usage()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of `pacman.model.constraints.abstract_constraint.AbstractConstraint`

get_provenance_data_from_machine(*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static get_sdram_usage(*send_buffer_times*, *send_buffer_max_space*, *recording_enabled*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static n_regions_to_allocate(*send_buffering*, *recording*)
Get the number of regions that will be allocated

resources_required
The resources required by the vertex

Return type `pacman.model.resources.ResourceContainer`

send_buffer_times

send_buffers

update_buffer(*arg*)

virtual_key

1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

A

B

C

F

G

H

I

M

N

O

P

R

S

T

U