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_allocation_controller module

class spinn_front_end_common.abstract_models.impl.machine_allocation_controller.**MachineAllo**

Bases: *spinn_front_end_common.abstract_models.abstract_machine_allocation_controller.AbstractMachineAllocationController*

close()

Indicate that the use of the machine is complete

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** (*Placement*) – 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*)

Parameters

- **spec** (DataSpecificationGenerator) – The data specification to write into.
- **placement** – Where this node is on the SpiNNaker machine.
- **machine_graph** – The graph containing this node.
- **routing_info** –
- **iptags** –
- **reverse_iptags** –
- **machine_time_step** –
- **time_step_factor** –

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.**MachineAllocationController**(*thread_name*)

Bases: *spinn_front_end_common.abstract_models.abstract_machine_allocation_controller.AbstractMachineAllocationController*

close()

Indicate that the use of the machine is complete

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** (*Placement*) – 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*)

Parameters

- **spec** (DataSpecificationGenerator) – The data specification to write into.
- **placement** – Where this node is on the SpiNNaker machine.
- **machine_graph** – The graph containing this node.
- **routing_info** –
- **iptags** –
- **reverse_iptags** –
- **machine_time_step** –
- **time_step_factor** –

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_can_reset module

class spinn_front_end_common.abstract_models.abstract_can_reset.**AbstractCanReset**

Bases: object

Indicates an object that can be reset to time 0

reset_to_first_timestep()

Reset the object to first time step

1.1.1.4 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 or data generation.

mark_no_changes()
Marks the point after which changes are reported, so that new changes can be detected before the next check.

requires_data_generation
True if changes that have been made require that data generation be performed. By default this returns False but can be overridden to indicate changes that require data regeneration.

Return type bool

requires_mapping
True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

1.1.1.5 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** (*Placement*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

1.1.1.6 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

Marks a machine graph vertex that can be launched on a SpiNNaker core.

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 *ExecutableType*

1.1.1.7 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

1.1.1.8 spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constrain
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** (OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list(AbstractConstraint)

1.1.1.9 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.

1.1.1.10 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** (OutgoingPartition) – An partition that comes out of this vertex
- **graph_mapper** (GraphMapper) – A mapper between the graphs

Returns A list of constraints

Return type list(AbstractConstraint)

1.1.1.11 spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constrain
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(AbstractConstraint)

1.1.1.12 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

1.1.1.13 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

1.1.1.14 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

1.1.1.15 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.

1.1.1.16 spinn_front_end_common.abstract_models.abstract_uses_memory_io module

class spinn_front_end_common.abstract_models.abstract_uses_memory_io.**AbstractUsesMemoryIO**
Bases: object

Indicates that the class will write data using the MemoryIO interface

get_memory_io_data_size()
Get the size of the data area to allocate to this vertex

Returns The size of the data area in bytes

Return type int

write_data_to_memory_io(*memory*, *tag*)
Write the data to the given memory object

Parameters

- **memory** (MemoryIO) – The memory to write to
- **tag** (*int*) – The tag given to the allocated memory

1.1.1.17 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

1.1.1.18 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 or data generation.

mark_no_changes()
Marks the point after which changes are reported, so that new changes can be detected before the next check.

requires_data_generation
True if changes that have been made require that data generation be performed. By default this returns False but can be overridden to indicate changes that require data regeneration.

Return type bool

requires_mapping
True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

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

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

Parameters

- **spec** (*Placement*) – 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

Marks a machine graph vertex that can be launched on a SpiNNaker core.

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 *ExecutableType*

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** (OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list(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** (OutgoingPartition) – An partition that comes out of this vertex
- **graph_mapper** (GraphMapper) – A mapper between the graphs

Returns A list of constraints

Return type list(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(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

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

Indicates that the class will write data using the MemoryIO interface

get_memory_io_data_size()
Get the size of the data area to allocate to this vertex

Returns The size of the data area in bytes

Return type int

write_data_to_memory_io(*memory*, *tag*)
Write the data to the given memory object

Parameters

- **memory** (MemoryIO) – The memory to write to
- **tag** (*int*) – The tag given to the allocated memory

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

Indicates an object that can be reset to time 0

reset_to_first_timestep()
Reset the object to first time step

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_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(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_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(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 Whether there are no keys to send for the region

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 Whether there are more keys to send for the parameters

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 Whether there is more data

Return type bool

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_i 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 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_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_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_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(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_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(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 Whether there are no keys to send for the region

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 Whether there are more keys to send for the parameters

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 Whether there is more data

Return type bool

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 *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_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_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.abstract_database module

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

This API separates the required database calls from the implementation.

Methods here are designed for the convenience of the caller not the database.

There should only ever be a single Database Object in use at any time. In the case of application_graph_changed the first should closed and a new one created.

Do not assume that just because 2 database objects where opened with the same parameters (for example sqllite file) that they hold the same data. In fact the second init is allowed to delete any previous data.

While not recommended implementation objects are allowed to hold data in memory, with the exception of data required by the java which must be in the database once commit is called.

clear()
Clears the data for all regions.

Warning: This method will be removed when the database moves to keeping data after reset.

Return type None

close()
Signals that the database can be closed and will not be reused.

Once this is called any other method in this API is allowed to raise any kind of exception.

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)

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

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 **report_folder** (*str*) – The directory to write the database used to store some of the data.

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** (*int*) – placement x coordinate
- **y** (*int*) – placement y coordinate
- **p** (*int*) – placement p coordinate
- **region_id** (*int*) – 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*)
It is no longer possible to get access to the data pointer.

Use get_region_data to get the data and missing flag directly.

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.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.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

reset()

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.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.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_r
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(int)*) – The keys to send

clear()
Clears the buffer

current_timestamp
The current timestamp in the iterator

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

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(int)

spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region
Get the number of bytes used by a given number of keys

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

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(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*)

spinn_front_end_common.interface.buffer_management.storage_objects.sqllite_database module

class spinn_front_end_common.interface.buffer_management.storage_objects.sqllite_database.S

Bases: *spinn_front_end_common.interface.buffer_management.storage_objects.abstract_database.AbstractDatabase*

Specific implementation of the Database for SqlLite

Parameters **database_file** (*str*) – The name of a file that contains (or will contain) an SQLite database holding the data.

clear()

Clears the data for all regions.

Warning: This method will be removed when the database moves to keeping data after reset.

Return type None

close()

Signals that the database can be closed and will not be reused.

Once this is called any other method in this API is allowed to raise any kind of exception.

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
- **x** – x coordinate of the chip

- **y** – y coordinate of the chip
- **p** – Core within the specified chip
- **region** – 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 the data stored for a given region of a given core

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

Return type (bytearray, bool)

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

Module contents

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

This API separates the required database calls from the implementation.

Methods here are designed for the convenience of the caller not the database.

There should only ever be a single Database Object in use at any time. In the case of application_graph_changed the first should closed and a new one created.

Do not assume that just because 2 database objects where opened with the same parameters (for example sqllite file) that they hold the same data. In fact the second init is allowed to delete any previous data.

While not recommended implementation objects are allowed to hold data in memory, with the exception of data required by the java which must be in the database once commit is called.

clear()
Clears the data for all regions.

Warning: This method will be removed when the database moves to keeping data after reset.

Return type None

close()
Signals that the database can be closed and will not be reused.

Once this is called any other method in this API is allowed to raise any kind of exception.

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)

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

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedReceivingD**
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 **report_folder** (*str*) – The directory to write the database used to store some of the data.

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** (*int*) – placement x coordinate
- **y** (*int*) – placement y coordinate
- **p** (*int*) – placement p coordinate
- **region_id** (*int*) – 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*)
It is no longer possible to get access to the data pointer.

Use get_region_data to get the data and missing flag directly.

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.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.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

reset()

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.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.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(int)*) – The keys to send

clear()
Clears the buffer

current_timestamp
The current timestamp in the iterator

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

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(int)

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(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*)

class spinn_front_end_common.interface.buffer_management.storage_objects.**SqlLiteDatabase**(*da*

Bases: *spinn_front_end_common.interface.buffer_management.storage_objects.abstract_database.AbstractDatabase*

Specific implementation of the Database for SqlLite

Parameters **database_file** (*str*) – The name of a file that contains (or will contain) an SQLite database holding the data.

clear()

Clears the data for all regions.

Warning: This method will be removed when the database moves to keeping data after reset.

Return type None

close()

Signals that the database can be closed and will not be reused.

Once this is called any other method in this API is allowed to raise any kind of exception.

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
- **x** – x coordinate of the chip
- **y** – y coordinate of the chip
- **p** – Core within the specified chip
- **region** – 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 the data stored for a given region of a given core

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

Return type (bytearray, bool)

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

Submodules

spinn_front_end_common.interface.buffer_management.buffer_manager module

class spinn_front_end_common.interface.buffer_management.buffer_manager.**BufferManager**(*placem*
tags,
transc
ex-
tra_m
packet
ex-
tra_m
ma-
chine,
*fixed_*
uses_a
re-
port_f
java_c

Bases: object

Manager of send buffers.

Parameters

- **placements** (Placements) – The placements of the vertices
- **tags** (Tags) – The tags assigned to the vertices
- **transceiver** (Transceiver) – The transceiver to use for sending and receiving information
- **packet_gather_cores_to_ethernet_connection_map** – mapping of cores to
- **report_folder** (*str*) – The directory for reports which includes the file to use as an SQL database.
- **java_caller** (*JavaCaller*) – Support class to call Java or None to use python

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

Parameters **vertex** – the vertex to be managed

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** (*AbstractSendsBuffersFromHost*) – the vertex to be managed

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

Parameters

- **x** (*int*) – placement x coordinate
- **y** (*int*) – placement y coordinate
- **p** (*int*) – placement p coordinate
- **recording_region_id** (*int*) – the recording region ID

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

Parameters

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

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_data_for_placements(*placements*, *progress=None*)

get_data_for_vertex(*placement*, *recording_region_id*)
It is no longer possible to get access to the data pointer.

Use get_data_by_vertex instead which returns the data that pointer.read_all() used to return the missing flag as before

load_initial_buffers()
Load the initial buffers for the senders using memory writes.

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_recording_data_**
Get the size of the recorded data to be reserved that doesn't

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

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(int)

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

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_region_pointer**(

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*, *extra_monitor_cores*, *packet_gather_cores_to_ethe*
extra_monitor_to_chip_mappi
machine, *fixed_routes*, *uses_advanced_monitors*, *report_folder*, *java_caller=None*)

Bases: `object`

Manager of send buffers.

Parameters

- **placements** (`Placements`) – The placements of the vertices
- **tags** (`Tags`) – The tags assigned to the vertices
- **transceiver** (`Transceiver`) – The transceiver to use for sending and receiving information
- **packet_gather_cores_to_ethernet_connection_map** – mapping of cores to
- **report_folder** (*str*) – The directory for reports which includes the file to use as an SQL database.
- **java_caller** (*JavaCaller*) – Support class to call Java or None to use python

add_receiving_vertex(*vertex*)

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

Parameters **vertex** – the vertex to be managed

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** (*AbstractSendsBuffersFromHost*) – the vertex to be managed

clear_recorded_data(*x*, *y*, *p*, *recording_region_id*)

Removes the recorded data stored in memory.

Parameters

- **x** (*int*) – placement x coordinate
- **y** (*int*) – placement y coordinate
- **p** (*int*) – placement p coordinate
- **recording_region_id** (*int*) – the recording region ID

get_data_by_placement(*placement*, *recording_region_id*)

Get the data container for all the data retrieved during the simulation from a specific region area of a core.

Parameters

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

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_data_for_placements(*placements*, *progress=None*)

get_data_for_vertex(*placement*, *recording_region_id*)
It is no longer possible to get access to the data pointer.

Use get_data_by_vertex instead which returns the data that pointer.read_all() used to return the missing flag as before

load_initial_buffers()
Load the initial buffers for the senders using memory writes.

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.ds package

Submodules

spinn_front_end_common.interface.ds.data_row_reader module

spinn_front_end_common.interface.ds.data_row_writer module

spinn_front_end_common.interface.ds.data_specification_targets module

spinn_front_end_common.interface.ds.ds_abstact_database module

spinn_front_end_common.interface.ds.ds_pretend_database module

spinn_front_end_common.interface.ds.ds_sqllite_database module

spinn_front_end_common.interface.ds.ds_write_info module

Module contents

spinn_front_end_common.interface.interface_functions package

Submodules

spinn_front_end_common.interface.interface_functions.application_finisher module

spinn_front_end_common.interface.interface_functions.application_runner module

spinn_front_end_common.interface.interface_functions.buffer_extractor module

spinn_front_end_common.interface.interface_functions.buffer_manager_creator module

spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer module

spinn_front_end_common.interface.interface_functions.chip_iobuf_extractor module

spinn_front_end_common.interface.interface_functions.chip_provenance_updater module

spinn_front_end_common.interface.interface_functions.chip_runtime_updater module

spinn_front_end_common.interface.interface_functions.data_in_multicast_routing_generator module

spinn_front_end_common.interface.interface_functions.database_interface module

spinn_front_end_common.interface.interface_functions.dsg_region_reloader module

spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper module

spinn_front_end_common.interface.interface_functions.graph_binary_gatherer module

spinn_front_end_common.interface.interface_functions.graph_data_specification_writer module

spinn_front_end_common.interface.interface_functions.graph_measurer module

spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer module

spinn_front_end_common.interface.interface_functions.hbp_allocator module

spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator module

spinn_front_end_common.interface.interface_functions.host_execute_data_specification module

spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_graphs module

spinn_front_end_common.interface.interface_functions.spalloc_allocator module

spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator module

spinn_front_end_common.interface.interface_functions.tags_loader module

spinn_front_end_common.interface.interface_functions.tdma_agenda_builder module

spinn_front_end_common.interface.interface_functions.virtual_machine_generator module

spinn_front_end_common.interface.interface_functions.write_memory_io_data module

Module contents

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(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

Returns 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** – SpiNNMan transceiver
- **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** – number of 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** – number of elements being sampled

Return type None

Module contents

class spinn_front_end_common.interface.profiling.**AbstractHasProfileData**

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*

class spinn_front_end_common.interface.profiling.**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(str)

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.A
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(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_fro*
AbstractProvidesProvenanceDataFromMachine

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

NUM_PROVENANCE_DATA_ENTRIES = 5

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(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.

NUM_PROVENANCE_DATA_ENTRIES = 5

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

spinn_front_end_common.interface.simulation.**get_simulation_header_array**(*binary_file_name*, *machine_time_step*, *time_scale_factor*)

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

1.1.3.2 Submodules

1.1.3.3 spinn_front_end_common.interface.abstract_spinnaker_base module

1.1.3.4 spinn_front_end_common.interface.config_handler module

class spinn_front_end_common.interface.config_handler.**ConfigHandler**(*configfile*, *default_config_paths*, *validation_cfg*)

Bases: object

Subclass of AbstractSpinnaker base that handles function only dependent of the config and the order its methods are called

child_folder(*parent*, *child_name*)

set_up_output_application_data_specifics(*n_calls_to_run*)

Parameters **n_calls_to_run** –

the counter of how many times run has been called. :type n_calls_to_run: int :return: the run folder for this simulation to hold app data

write_finished_file()

1.1.3.5 spinn_front_end_common.interface.java_caller module

class spinn_front_end_common.interface.java_caller.**JavaCaller**(*json_folder*, *java_call*, *java_spinnaker_path=None*, *java_properties=None*)

Bases: object

Support class that holds all the stuff for running stuff in Java.

This includes the work of preparing data for transmitting to Java and back.

This separates the choices of how to call the Java batch vs streaming, jar locations, parameters, etc from the rest of the python code.

Creates a java caller and checks the user/config parameters.

Parameters

- **json_folder** (*str*) – The location where the machine JSON is written.
- **java_call** (*str*) – Call to start java. Including the path if required.
- **java_spinnaker_path** – the path where the java code can be found. This must point to a local copy of https://github.com/SpiNNakerManchester/JavaSpiNNaker. It must also have been built! If None the assumption is that it is the same parent directory as https://github.com/SpiNNakerManchester/SpiNNFrontEndCommon.
- **java_properties** (*str*) – Optional properties that will be passed to Java. Must start with -D For example -Dlogging.level=DEBUG

:raise ConfigurationException if simple parameter checking fails.

execute_app_data_specification(*use_monitors*)

execute_data_specification()
Writes all the data specs, uploading the result to the machine.

execute_system_data_specification()
Writes all the data specs, uploading the result to the machine.

get_all_data()
Gets all the data from the previously set placements and put these in the previously set database.

set_advanced_monitors(*placements*, *tags*, *monitor_cores*, *packet_gathers*)

Parameters

- **placements** (pacman.model.placements.Placements) – The placements of the vertices
- **tags** (pacman.model.tags.Tags) – The tags assigned to the vertices
- **monitor_cores** –
- **packet_gathers** –

Return type None

set_machine(*machine*)
Passes the machine in leaving this class to decide pass it to Java.

Parameters **machine** (spinn_machine.machine.Machine) – A machine Object

set_placements(*placements*, *transceiver*)
Passes in the placements leaving this class to decide pass it to Java.

This method may obtain extra information about he placements which is why it also needs the transceiver.

Currently the extra information extracted is recording region base address but this could change if recording region saved in the database.

Currently this method uses JSON but that may well change to using the database.

Parameters

- **placements** – The Placements Object
- **transceiver** – The Transceiver

set_report_folder(*report_folder*)
Passes the database file in.

Parameters **report_folder** (*str*) – Path to directory with SQLite databases and into which java will write

1.1.3.6 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

STOP_REQUESTED = 5

1.1.3.7 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

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(str)*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable(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, list(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_receive_label(*label*)

add_send_label(*label*)

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()
Closes the connection

Returns Nothing is returned

Return type None

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

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** (*list(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(str)*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable(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, list(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_receive_label(*label*)

add_send_label(*label*)

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()
Closes the connection

Returns Nothing is returned

Return type None

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

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** (*list(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

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(spinn_front_end_common.utilities.database.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.

Raises **SpinnmanIOException** – If anything goes wrong

close()

Closes the connection

Returns Nothing is returned

Return type None

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

run()

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

Return type dict(int, int)

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_ip_address(*x*, *y*)
Get an IP address to contact a chip

Parameters

- **x** – The x-coordinate of the chip
- **y** – The y-coordinate of the chip

Returns The IP address of the Ethernet to use to contact the chip

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(int, int)

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 tuple(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 tuple(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 tuple(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 tuple(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

get_placement(*label*)
Get the placement of a machine vertex with a given label

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

Returns The x, y, p coordinates of the vertex

Return type tuple(int, int, int)

get_placements(*label*)
Get the placements of an application vertex with a given label

Parameters **label** – The label of the vertex

:type label:str :return: A list of x, y, p coordinates of the vertices :rtype: list(tuple(int, int, 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*, *data_n_timesteps*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **data_n_timesteps** – The number of timesteps for which data space will been reserved
- **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

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(*spinn_front_end_common.utilities.database.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.

Raises **SpinnmanIOException** – If anything goes wrong

close()

Closes the connection

Returns Nothing is returned

Return type None

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

run()

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

Return type dict(int, int)

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_ip_address(*x*, *y*)
Get an IP address to contact a chip

Parameters

- **x** – The x-coordinate of the chip
- **y** – The y-coordinate of the chip

Returns The IP address of the Ethernet to use to contact the chip

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(int, int)

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 tuple(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 tuple(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 tuple(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 tuple(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

get_placement(*label*)
Get the placement of a machine vertex with a given label

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

Returns The x, y, p coordinates of the vertex

Return type tuple(int, int, int)

get_placements(*label*)
Get the placements of an application vertex with a given label

Parameters **label** – The label of the vertex

:type label:str :return: A list of x, y, p coordinates of the vertices :rtype: list(tuple(int, int, 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*, *data_n_timesteps*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **data_n_timesteps** – The number of timesteps for which data space will been reserved
- **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*
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.board_chip_report module

class spinn_front_end_common.utilities.report_functions.board_chip_report.**BoardChipReport**
Bases: object

Report on memory usage

AREA_CODE_REPORT_NAME = 'board_chip_report.txt'

spinn_front_end_common.utilities.report_functions.energy_report module

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

Creates a report about the approximate total energy consumed by a SpiNNaker job execution.

ENERGY_DETAILED_FILENAME = 'Detailed_energy_report.rpt'

ENERGY_SUMMARY_FILENAME = 'energy_summary_report.rpt'

JOULES_PER_SPIKE = 8e-10

JOULES_TO_KILOWATT_HOURS = 3600000

MILLIWATTS_FOR_BOXED_48_CHIP_FRAME_IDLE_COST = 0.0045833333

MILLIWATTS_FOR_FRAME_IDLE_COST = 0.117

MILLIWATTS_PER_CHIP_ACTIVE_OVERHEAD = 0.0006399999999999999

MILLIWATTS_PER_FPGA = 0.000584635

MILLIWATTS_PER_FRAME_ACTIVE_COST = 0.154163558

MILLIWATTS_PER_IDLE_CHIP = 0.00036

MILLIWATTS_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST = 0.01666667

N_MONITORS_ACTIVE_DURING_COMMS = 2

spinn_front_end_common.utilities.report_functions.fixed_route_from_machine_report module

class spinn_front_end_common.utilities.report_functions.fixed_route_from_machine_report.**Fix**
Bases: object

Generate a report of the fixed routes from the machine.

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.**R**
Bases: object

Module contents

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

Creates a report about the approximate total energy consumed by a SpiNNaker job execution.

ENERGY_DETAILED_FILENAME = 'Detailed_energy_report.rpt'

ENERGY_SUMMARY_FILENAME = 'energy_summary_report.rpt'

JOULES_PER_SPIKE = 8e-10

JOULES_TO_KILOWATT_HOURS = 3600000

MILLIWATTS_FOR_BOXED_48_CHIP_FRAME_IDLE_COST = 0.0045833333

MILLIWATTS_FOR_FRAME_IDLE_COST = 0.117

MILLIWATTS_PER_CHIP_ACTIVE_OVERHEAD = 0.0006399999999999999

MILLIWATTS_PER_FPGA = 0.000584635

MILLIWATTS_PER_FRAME_ACTIVE_COST = 0.154163558

MILLIWATTS_PER_IDLE_CHIP = 0.00036

MILLIWATTS_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST = 0.01666667

N_MONITORS_ACTIVE_DURING_COMMS = 2

class spinn_front_end_common.utilities.report_functions.**FixedRouteFromMachineReport**
Bases: object

Generate a report of the fixed routes from the machine.

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*, *des*
ti-
na-
tion
ex-
pec
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(*current_time*, *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*, *current_time*, *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(*current_time*, *run_time*, *infinite_run*, *core_subsets*, *n_cores*)

spinn_front_end_common.utilities.utility_objs package

Subpackages

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages package

Submodules

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.clear_reinjection_queue_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.clear_reinje

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

An SCP Request to set the dropped packet reinjected packet types

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255

- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.get_reinjection_status_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinject

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

An SCP Request to get the status of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.get_reinject
Bases: spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse

An SCP response to a request for the dropped packet reinjection status

read_data_bytestring(*data*, *offset*)
See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

reinjection_functionality_status

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.load_application_mc_routes_messag module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.load_applica

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

An SCP Request to write the application multicast routes into the router.

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.load_system_mc_routes_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.load_system_

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

An SCP Request to write the system multicast routes into the router

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **command_code** – the command code used by the extra monitor vertex for setting system multicast routes.

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.utility_objs.extra_monitor_scp_messages.reinjector_scp_commands module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.reinjector_s

Bases: enum.Enum

SCP Command codes for reinjection

CLEAR = 6

EXIT = 5

GET_STATUS = 3

RESET_COUNTERS = 4

SET_PACKET_TYPES = 2

SET_ROUTER_EMERGENCY_TIMEOUT = 1

SET_ROUTER_TIMEOUT = 0

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.reset_counters_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.reset_counte

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

An SCP Request to reset the statistics counters of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.set_reinjection_packet_types_messa module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_reinject

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

An SCP Request to set the dropped packet reinjected packet types

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255

- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **point_to_point** (*bool*) – If point to point should be set
- **multicast** (*bool*) – If multicast should be set
- **nearest_neighbour** (*bool*) – If nearest neighbour should be set
- **fixed_route** (*bool*) – If fixed route should be set

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.utility_objs.extra_monitor_scp_messages.set_router_emergency_timeout_mes module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_e

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

An SCP Request to set the router emergency timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** (*int*) – The exponent of the timeout value, between 0 and 15

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.utility_objs.extra_monitor_scp_messages.set_router_timeout_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_t

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

An SCP Request to the extra monitor core to set the router timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** (*int*) – The exponent of the timeout value, between 0 and 15

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.utility_objs.extra_monitor_scp_messages.speedup_in_scp_commands module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.speedup_in_s

Bases: enum.Enum

SCP Command codes for data speed up in

LOAD_APPLICATION_MC_ROUTES = 7

LOAD_SYSTEM_MC_ROUTES = 8

SAVE_APPLICATION_MC_ROUTES = 6

Module contents

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.**GetReinject**

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

An SCP Request to get the status of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.**GetReinject**

Bases: spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse

An SCP response to a request for the dropped packet reinjection status

read_data_bytestring(*data*, *offset*)

See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

reinjection_functionality_status

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.**ResetCounte**

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

An SCP Request to reset the statistics counters of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.**SetReinject**

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

An SCP Request to set the dropped packet reinjected packet types

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **point_to_point** (*bool*) – If point to point should be set
- **multicast** (*bool*) – If multicast should be set
- **nearest_neighbour** (*bool*) – If nearest neighbour should be set
- **fixed_route** (*bool*) – If fixed route should be set

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.utility_objs.extra_monitor_scp_messages.**SetRouterEme**

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

An SCP Request to set the router emergency timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** (*int*) – The exponent of the timeout value, between 0 and 15

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.utility_objs.extra_monitor_scp_messages.**SetRouterTi**

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

An SCP Request to the extra monitor core to set the router timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** (*int*) – The exponent of the timeout value, between 0 and 15

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.utility_objs.extra_monitor_scp_messages.**ClearReinje**

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

An SCP Request to set the dropped packet reinjected packet types

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.**LoadApplicat**

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

An SCP Request to write the application multicast routes into the router.

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17

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.utility_objs.extra_monitor_scp_messages.**LoadSystemM**

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

An SCP Request to write the system multicast routes into the router

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the extra monitor vertex, between 0 and 17
- **command_code** – the command code used by the extra monitor vertex for setting system multicast routes.

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.utility_objs.extra_monitor_scp_processes package

Submodules

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.clear_queue_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.clear_queue
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

reset_counters(*core_subsets*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.load_application_mc_routes_proces module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.load_applic

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

load_application_mc_routes(*core_subsets*)

Parameters **core_subsets** – sets of cores to send command to

Return type None

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.load_system_mc_routes_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.load_syste

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

load_system_mc_routes(*core_subsets*)

Parameters **core_subsets** – sets of cores to send command to

Return type None

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

get_reinjection_status(*x*, *y*, *p*)

get_reinjection_status_for_core_subsets(*core_subsets*)

handle_reinjection_status_response(*response*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.reset_counters_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.reset_count
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

reset_counters(*core_subsets*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_packet_types_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_packet_
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_packet_types(*core_subsets*, *point_to_point*, *multicast*, *nearest_neighbour*, *fixed_route*)
Set what types of packets should be reinjected.

Parameters

- **core_subsets** – sets of cores to send command to
- **point_to_point** (*bool*) – If point-to-point should be set
- **multicast** (*bool*) – If multicast should be set
- **nearest_neighbour** (*bool*) – If nearest neighbour should be set
- **fixed_route** (*bool*) – If fixed route should be set
- **command_code** – The SCP command code

Return type None

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_emergency_timeout_proc module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_timeout(*mantissa*, *exponent*, *core_subsets*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_timeout_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_timeout(*mantissa*, *exponent*, *core_subsets*)

Module contents

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ReadStatusR**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

get_reinjection_status(*x*, *y*, *p*)

get_reinjection_status_for_core_subsets(*core_subsets*)

handle_reinjection_status_response(*response*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ResetCounte**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

reset_counters(*core_subsets*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**SetPacketTy**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_packet_types(*core_subsets*, *point_to_point*, *multicast*, *nearest_neighbour*, *fixed_route*)
Set what types of packets should be reinjected.

Parameters

- **core_subsets** – sets of cores to send command to
- **point_to_point** (*bool*) – If point-to-point should be set
- **multicast** (*bool*) – If multicast should be set
- **nearest_neighbour** (*bool*) – If nearest neighbour should be set
- **fixed_route** (*bool*) – If fixed route should be set
- **command_code** – The SCP command code

Return type None

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**SetRouterEm**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_timeout(*mantissa*, *exponent*, *core_subsets*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**SetRouterTi**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

set_timeout(*mantissa*, *exponent*, *core_subsets*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ClearQueueI**
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

reset_counters(*core_subsets*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**LoadApplica**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

load_application_mc_routes(*core_subsets*)

Parameters **core_subsets** – sets of cores to send command to

Return type None

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**LoadSystemM**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

load_system_mc_routes(*core_subsets*)

Parameters **core_subsets** – sets of cores to send command to

Return type None

Submodules

spinn_front_end_common.utilities.utility_objs.data_written module

class spinn_front_end_common.utilities.utility_objs.data_written.**DataWritten**(*start_address*, *memory_used*, *memory_written*)

Bases: object

Describes data written to SpiNNaker.

memory_used

memory_written

start_address

spinn_front_end_common.utilities.utility_objs.dpri_flags module

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

Bases: enum.Enum

SCP Dropped Packet Reinjection (DPRI) packet type flags

FIXED_ROUTE = 8

MULTICAST = 1

NEAREST_NEIGHBOUR = 4

POINT_TO_POINT = 2

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_targets module

class spinn_front_end_common.utilities.utility_objs.executable_targets.**ExecutableTargets**

Bases: spinnman.model.executable_targets.ExecutableTargets

add_processor(*binary*, *chip_x*, *chip_y*, *chip_p*, *executable_type=None*)

Add a processor to the executable targets

Parameters

- **binary** – the binary path for executable
- **chip_x** – the coordinate on the machine in terms of x for the chip
- **chip_y** – the coordinate on the machine in terms of y for the chip
- **chip_p** – the processor ID to place this executable on

Returns

add_subsets(*binary*, *subsets*, *executable_type=None*)

Add core subsets to a binary

Parameters

- **binary** – the path to the binary needed to be executed
- **subsets** – the subset of cores that the binary needs to be loaded on

Returns

executable_types_in_binary_set()
get the executable types in the set of binaries

Returns iterable of the executable types in this binary set.

get_binaries_of_executable_type(*executable_type*)
get the binaries of a given a executable type

Parameters **execute_type** – the executable type enum value

Returns iterable of binaries with that executable type

get_n_cores_for_executable_type(*executable_type*)
returns the number of cores that the executable type is using

Parameters **executable_type** – the executable type for locating n cores of

Returns the number of cores using this executable type

spinn_front_end_common.utilities.utility_objs.executable_type module

class spinn_front_end_common.utilities.utility_objs.executable_type.**ExecutableType**(*value*, *start_state*, *end_state*, *supports_auto*, *doc=''*)

Bases: enum.Enum

The different types of executable from the perspective of how they are started and controlled.

NO_APPLICATION = 3

RUNNING = 0

SYNC = 1

SYSTEM = 4

USES_SIMULATION_INTERFACE = 2

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 LPGs 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.**ProvenanceDataIter**

Bases: object

Container for provenance data

Parameters

- **names** – A list of strings representing the naming hierarchy 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

spinn_front_end_common.utilities.utility_objs.reinjection_status module

class spinn_front_end_common.utilities.utility_objs.reinjection_status.**ReInjectionStatus**(*da*
of
se

Bases: object

Represents a status information from dropped packet reinjection

Parameters

- **data** (*str*) – The data containing the information
- **offset** (*int*) – The offset in the data where the information starts

is_reinjecting_fixed_route
True if re-injection of fixed-route packets is enabled

is_reinjecting_multicast
True if re-injection of multicast packets is enabled

is_reinjecting_nearest_neighbour
True if re-injection of nearest neighbour packets is enabled

is_reinjecting_point_to_point
True if re-injection of point-to-point packets is enabled

n_dropped_packet_overflows
Of the n_dropped_packets received, how many were lost due to not having enough space in the queue of packets to reinject

n_dropped_packets
The number of packets dropped by the router and received by the re injection functionality (may not fit in the queue though)

n_link_dumps
The number of times that when a dropped packet was caused due to a link failing to take the packet.

Returns int

n_missed_dropped_packets
The number of times that when a dropped packet was read it was found that another one or more packets had also been dropped, but had been missed

n_processor_dumps
The number of times that when a dropped packet was caused due to a processor failing to take the packet.

Returns int

n_reinjected_packets
Of the n_dropped_packets received, how many packets were successfully re injected

router_emergency_timeout
The WAIT2 timeout value of the router in cycles

router_emergency_timeout_parameters
The WAIT2 timeout value of the router as mantissa and exponent

router_timeout
The WAIT1 timeout value of the router in cycles

router_timeout_parameters
The WAIT1 timeout value of the router as mantissa and exponent

Module contents

class spinn_front_end_common.utilities.utility_objs.**DataWritten**(*start_address*, *memory_used*, *memory_written*)

Bases: object

Describes data written to SpiNNaker.

memory_used

memory_written

start_address

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

Bases: enum.Enum

SCP Dropped Packet Reinjection (DPRI) packet type flags

FIXED_ROUTE = 8

MULTICAST = 1

NEAREST_NEIGHBOUR = 4

POINT_TO_POINT = 2

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.**ExecutableType**(*value*, *start_state*, *end_state*, *supports_auto_pause_and_resume*, *doc=''*)

Bases: enum.Enum

The different types of executable from the perspective of how they are started and controlled.

NO_APPLICATION = 3

RUNNING = 0

SYNC = 1

SYSTEM = 4

USES_SIMULATION_INTERFACE = 2

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 LPGs 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 naming hierarchy 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

class spinn_front_end_common.utilities.utility_objs.**ReInjectionStatus**(*data*, *offset*)

Bases: object

Represents a status information from dropped packet reinjection

Parameters

- **data** (*str*) – The data containing the information
- **offset** (*int*) – The offset in the data where the information starts

is_reinjecting_fixed_route
True if re-injection of fixed-route packets is enabled

is_reinjecting_multicast
True if re-injection of multicast packets is enabled

is_reinjecting_nearest_neighbour
True if re-injection of nearest neighbour packets is enabled

is_reinjecting_point_to_point
True if re-injection of point-to-point packets is enabled

n_dropped_packet_overflows
Of the n_dropped_packets received, how many were lost due to not having enough space in the queue of packets to reinject

n_dropped_packets
The number of packets dropped by the router and received by the re injection functionality (may not fit in the queue though)

n_link_dumps
The number of times that when a dropped packet was caused due to a link failing to take the packet.

Returns int

n_missed_dropped_packets
The number of times that when a dropped packet was read it was found that another one or more packets had also been dropped, but had been missed

n_processor_dumps
The number of times that when a dropped packet was caused due to a processor failing to take the packet.

Returns int

n_reinjected_packets
Of the n_dropped_packets received, how many packets were successfully re injected

router_emergency_timeout
The WAIT2 timeout value of the router in cycles

router_emergency_timeout_parameters
The WAIT2 timeout value of the router as mantissa and exponent

router_timeout
The WAIT1 timeout value of the router in cycles

router_timeout_parameters
The WAIT1 timeout value of the router as mantissa and exponent

class spinn_front_end_common.utilities.utility_objs.**ExecutableTargets**
Bases: spinnman.model.executable_targets.ExecutableTargets

add_processor(*binary*, *chip_x*, *chip_y*, *chip_p*, *executable_type=None*)
Add a processor to the executable targets

Parameters

- **binary** – the binary path for executable
- **chip_x** – the coordinate on the machine in terms of x for the chip
- **chip_y** – the coordinate on the machine in terms of y for the chip
- **chip_p** – the processor ID to place this executable on

Returns

add_subsets(*binary*, *subsets*, *executable_type=None*)
Add core subsets to a binary

Parameters

- **binary** – the path to the binary needed to be executed
- **subsets** – the subset of cores that the binary needs to be loaded on

Returns

executable_types_in_binary_set()
get the executable types in the set of binaries

Returns iterable of the executable types in this binary set.

get_binaries_of_executable_type(*executable_type*)
get the binaries of a given a executable type

Parameters **execute_type** – the executable type enum value

Returns iterable of binaries with that executable type

get_n_cores_for_executable_type(*executable_type*)
returns the number of cores that the executable type is using

Parameters **executable_type** – the executable type for locating n cores of

Returns the number of cores using this executable type

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

EXTRA_MONITOR_CORE_DATA_IN_SPEED_UP = 6

EXTRA_MONITOR_CORE_DATA_SPEED_UP = 5

EXTRA_MONITOR_CORE_REINJECTION = 4

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 parameter 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*

add_socket_address(*socket_address*)

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()

tags

time_scale_factor

transceiver

use_virtual_board

verify_not_running()

1.1.5.6 spinn_front_end_common.utilities.function_list module

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**
Convert between a time diff and total milliseconds.

Returns total milliseconds

Return type float

spinn_front_end_common.utilities.helpful_functions.**convert_vertices_to_core_subset**(*vertices*, *placements*)

Converts vertices into core subsets.

Parameters

- **extra_monitor_cores_to_set** – the vertices to convert to core subsets
- **placements** – the placements object

Returns the CoreSubSets of the vertices

spinn_front_end_common.utilities.helpful_functions.**determine_flow_states**(*executable_types*, *no_sync_changes*)

Get the start and end states for these executable types.

Parameters

- **executable_types** (dict(*ExecutableType* -> any)) – the execute types to locate start and end states from
- **no_sync_changes** (*int*) – the number of times sync signals been sent

Returns dict of executable type to states.

Return type 2-tuple

spinn_front_end_common.utilities.helpful_functions.**emergency_recover_state_from_failure**(*txrx*, *app*, *vertex*, *placement*)

Used to get at least *some* information out of a core when something goes badly wrong. Not a replacement for what abstract spinnaker base does.

Parameters

- **txrx** – The transceiver.
- **app_id** – The ID of the application.

- **vertex** (*spinn_front_end_common.abstract_models.AbstractHasAssociatedBinary*) – The vertex to retrieve the IOBUF from if it is suspected as being dead
- **placement** – Where the vertex is located.

spinn_front_end_common.utilities.helpful_functions.**emergency_recover_states_from_failure**(*txrx*, *app_id*, *executable_targets*)

Used to get at least *some* information out of a core when something goes badly wrong. Not a replacement for what abstract spinnaker base does.

Parameters

- **txrx** – The transceiver.
- **app_id** – The ID of the application.
- **executable_targets** – The what/where mapping

spinn_front_end_common.utilities.helpful_functions.**flood_fill_binary_to_spinnaker**(*executable_targets*, *binary*, *txrx*, *app_id*)

flood fills a binary to spinnaker on a given app_id given the executable targets and binary.

Parameters

- **executable_targets** – the executable targets object
- **binary** – the binary to flood fill
- **txrx** (Tranceiver) – spinnman instance
- **app_id** – the app id to load it on

Returns the number of cores it was loaded onto

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_extra_monitor_mc_receiver**(*machine*, *placement_x*, *placement_y*, *packet_ga*

spinn_front_end_common.utilities.helpful_functions.**locate_memory_region_for_placement**(*placem* *region*, *transc*

Get the address of a region for a placement

Parameters

- **region** (*int*) – the region to locate the base address of
- **placement** (Placement) – the placement object to get the region address of
- **transceiver** (Transceiver) – 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.**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 tuple(set(tuple(int, int)), set(tuple(int, int, int)), set(tuple(tuple(int, int), int)))

spinn_front_end_common.utilities.helpful_functions.**write_address_to_user0**(*txrx*, *x*, *y*, *p*, *address*)

Writes the given address into the user_0 register of the given core.

Parameters

- **txrx** – The transceiver.
- **x** – Chip coordinate.
- **y** – Chip coordinate.
- **p** – Core ID on chip.
- **address** – Value to write (32-bit integer)

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(*socket_address*)

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()

tags

time_scale_factor

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*

add_socket_address(*socket_address*)

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()

tags

time_scale_factor

transceiver

use_virtual_board

verify_not_running()

class spinn_front_end_common.utilities.**SimulatorInterface**

Bases: object

add_socket_address(*socket_address*)

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()

tags

time_scale_factor

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 module

class spinn_front_end_common.utility_models.chip_power_monitor.**ChipPowerMonitor**(*label*, *constraints*, *n_samples_per_*, *sampling_frequency*)

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*

Represents idle time recording code in a application graph.

Parameters

- **label** (*str*) – vertex label
- **constraints** – constraints for the vertex
- **n_samples_per_recording** (*int*) – 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** (*Slice*) – The slice of atoms that the machine vertex will cover

- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(**args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

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

SAMPLE_RECORDING_REGION = 0

static binary_file_name()
Return the string binary file name

Returns basestring

static binary_start_type()
The type of binary that implements this vertex

Returns starttype enum

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_recorded_data(*placement*, *buffer_manager*)
Get data from SDRAM given placement and buffer manager

Parameters

- **placement** – the location on machine 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(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(*time_step*, *time_scale_factor*, *n_samples_per_recording*, *sampling_frequency*)
Get the resources used by this vertex

Returns Resource container

n_samples_per_recording

resources_required
The resources required by the vertex

Return type ResourceContainer

sampling_frequency

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*)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

edges_and_partitions()

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

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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(AbstractConstraint)

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

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.**CommandSenderMach**

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*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS
Bases: enum.Enum

COMMANDS_AT_START_RESUME = 2

COMMANDS_AT_STOP_PAUSE = 3

COMMANDS_WITH_ARBITRARY_TIMES = 1

PROVENANCE_REGION = 4

SYSTEM_REGION = 0

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(*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(*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(*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

edges_and_partitions()

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_edges_and_partitions(*pre_vertex*, *edge_type*)

static get_n_command_bytes(*commands*)

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(AbstractConstraint)

get_timed_commands_bytes()

resources_required
The resources required by the vertex

Return type ResourceContainer

1.1.6.6 spinn_front_end_common.utility_models.data_speed_up_packet_gatherer module

class spinn_front_end_common.utility_models.data_speed_up_packet_gatherer.**DataSpeedUpPacket**

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*

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

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

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises `None` – this method does not raise any known exception

machine_vertex

n_atoms
The number of atoms in the vertex

Return type int

1.1.6.7 spinn_front_end_common.utility_models.data_speed_up_packet_gatherer_machine_vertex module

class spinn_front_end_common.utility_models.data_speed_up_packet_gatherer_machine_vertex.**DA**
Bases: enum.Enum

RECEIVE_FINISHED = 2005

RECEIVE_FIRST_MISSING_SEQ = 2003

RECEIVE_MISSING_SEQ_DATA = 2004

SEND_DATA_TO_LOCATION = 200

SEND_DONE = 2002

SEND_SEQ_DATA = 2000

class spinn_front_end_common.utility_models.data_speed_up_packet_gatherer_machine_vertex.**DA**
Bases: enum.Enum

CLEAR = 2000

MISSING_SEQ = 1001

START_MISSING_SEQ = 1000

START_SENDING = 100

class spinn_front_end_common.utility_models.data_speed_up_packet_gatherer_machine_vertex.**Da**

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.interface.provenance.abstract_provides_local_provenance_data.AbstractProvidesLocalProvenanceData*

BASE_KEY = 4294967289

BASE_MASK = 4294967291

END_FLAG_KEY = 4294967286

END_FLAG_KEY_OFFSET = 3

FIRST_DATA_KEY = 4294967287

FIRST_DATA_KEY_OFFSET = 2

IN_REPORT_NAME = 'speeds_gained_in_speed_up_process.rpt'

LAST_MESSAGE_FLAG_BIT_MASK = 2147483648

LONG_TIMEOUT = (14, 14)

MISSING_SEQ_NUMS_END_FLAG = 4294967295

NEW_SEQ_KEY = 4294967288

NEW_SEQ_KEY_OFFSET = 1

OUT_REPORT_NAME = 'routers_used_in_speed_up_process.rpt'

SEQUENCE_NUMBER_MASK = 2147483647

SHORT_TIMEOUT = (1, 1)

TEMP_TIMEOUT = (15, 4)

TIMEOUT_PER_RECEIVE_IN_SECONDS = 1

TIME_OUT_FOR_SENDING_IN_SECONDS = 0.01

TRAFFIC_TYPE = 2

TRANSMISSION_THROTTLE_TIME = 1e-06

ZERO_TIMEOUT = (0, 0)

calculate_max_seq_num()
Deduce the max sequence number expected to be received

Returns int of the biggest sequence num expected

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_data(*placement*, *memory_address*, *length_in_bytes*, *fixed_routes*)
Gets data from a given core and memory address.

Parameters

- **placement** – placement object for where to get data from
- **memory_address** – the address in SDRAM to start reading from
- **length_in_bytes** – the length of data to read in bytes

- **fixed_routes** – the fixed routes, used in the report of which chips were used by the speed up process

Returns byte array of the data

get_local_provenance_data()
Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

static load_application_routing_tables(*transceiver*, *extra_monitor_cores*, *placements*)
Set all chips to have application table loaded in the router

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static load_system_routing_tables(*transceiver*, *extra_monitor_cores*, *placements*)
Set all chips to have the system table loaded in the router

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static locate_correct_write_data_function_for_chip_location(*uses_advanced_monitors*, *machine*, *x*, *y*, *transceiver*, *extra_monitor_cores_to_ethernet_connec*
supports other components figuring out which gather and function to call for writing data onto spinnaker

Parameters

- **uses_advanced_monitors** (*bool*) – Whether the system is using advanced monitors
- **machine** – the SpiNNMachine instance
- **x** – the chip x coordinate to write data to
- **y** – the chip y coordinate to write data to
- **extra_monitor_cores_to_ethernet_connection_map** – mapping between cores and connections
- **transceiver** – the SpiNNMan instance

Returns a write function of either a LPG or the spinnMan

Return type func

resources_required
The resources required by the vertex

Return type ResourceContainer

send_data_into_spinnaker(*x*, *y*, *base_address*, *data*, *n_bytes=None*, *offset=0*, *cpu=0*, *is_filename=False*)
sends a block of data into SpiNNaker to a given chip

Parameters

- **x** – chip x for data
- **y** – chip y for data
- **base_address** – the address in SDRAM to start writing memory
- **data** – the data to write
- **n_bytes** – how many bytes to read, or None if not set
- **offset** – where in the data to start from
- **is_filename** (*bool*) – whether data is actually a file.

Return type None

set_cores_for_data_streaming(*transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static static_resources_required()

static streaming(*gatherers*, *transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming via a Python context manager (i.e., using the 'with' statement).

Parameters

- **gatherers** – All the gatherers that are to be set
- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type a context manager

unset_cores_for_data_streaming(*transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

spinn_front_end_common.utility_models.data_speed_up_packet_gatherer_machine_vertex.**ceildiv**

How to divide two possibly-integer numbers and round up.

1.1.6.8 spinn_front_end_common.utility_models.extra_monitor_support module

class spinn_front_end_common.utility_models.extra_monitor_support.**ExtraMonitorSupport**(*constr*

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*

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

1.1.6.9 spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex module

class spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.**ExtraMonit**

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*

Parameters

- **constraints** – constraints on this vertex
- **reinject_multicast** – if we reinject multicast packets; defaults to value of *enable_reinjection* setting in configuration file
- **reinject_point_to_point** – if we reinject point-to-point packets
- **reinject_nearest_neighbour** – if we reinject nearest-neighbour packets
- **reinject_fixed_route** – if we reinject fixed route packets

clear_reinjection_queue(*transceiver*, *placements*, *extra_monitor_cores_to_set*)
Clears the queues for reinjection

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_reinjection_status(*placements*, *transceiver*)
Get the reinjection status from this extra monitor vertex

Parameters

- **transceiver** – the spinnMan interface
- **placements** – the placements object

Returns the reinjection status for this vertex

get_reinjection_status_for_vertices(*placements*, *extra_monitor_cores_for_data*, *transceiver*)

Get the reinjection status from a set of extra monitor cores

Parameters

- **placements** – the placements object
- **extra_monitor_cores_for_data** – the extra monitor cores to get status from
- **transceiver** – the spinnMan interface

Return type None

load_application_mc_routes(*placements*, *extra_monitor_cores_for_data*, *transceiver*)

Get the extra monitor cores to load up the application-based multicast routes (used by data in).

Parameters

- **placements** (Placements) – the placements object
- **extra_monitor_cores_for_data** (iterable(*ExtraMonitorSupportMachineVertex*)) – the extra monitor cores to get status from
- **transceiver** (Transceiver) – the spinnMan interface

Return type None

load_system_mc_routes(*placements*, *extra_monitor_cores_for_data*, *transceiver*)

Get the extra monitor cores to load up the system-based multicast routes (used by data in).

Parameters

- **placements** (Placements) – the placements object
- **extra_monitor_cores_for_data** (iterable(*ExtraMonitorSupportMachineVertex*)) – the extra monitor cores to get status from
- **transceiver** (Transceiver) – the spinnMan interface

Return type None

placement

reinject_fixed_route

reinject_multicast

reinject_nearest_neighbour

reinject_point_to_point

reset_reinjection_counters(*transceiver*, *placements*, *extra_monitor_cores_to_set*)

Resets the counters for reinjection

resources_required

The resources required by the vertex

Return type ResourceContainer

set_reinjection_packets(*placements*, *extra_monitor_cores_for_data*, *transceiver*, *point_to_point=None*, *multicast=None*, *nearest_neighbour=None*, *fixed_route=None*)

Parameters

- **placements** – placements object
- **extra_monitor_cores_for_data** – the extra monitor cores to set the packets of
- **transceiver** – spinnman instance
- **point_to_point** (*bool or None*) – If point to point should be set, or None if left as before
- **multicast** (*bool or None*) – If multicast should be set, or None if left as before
- **nearest_neighbour** (*bool or None*) – If nearest neighbour should be set, or None if left as before
- **fixed_route** (*bool or None*) – If fixed route should be set, or None if left as before.

Return type None

set_router_emergency_timeout(*timeout*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

Sets the timeout of the routers

Parameters

- **timeout** (*(int, int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (Transceiver) – the spinnMan instance
- **placements** (Placements) – the placements object
- **extra_monitor_cores_to_set** – the set of vertices to change the local chip for.

set_router_time_outs(*timeout*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

Supports setting of the router time outs for a set of chips via their extra monitor cores.

Parameters

- **timeout** (*(int, int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (Transceiver) – the spinnman interface
- **placements** (Placements) – placements object
- **extra_monitor_cores_to_set** – which vertices to use

Return type None

static static_get_binary_file_name()

static static_get_binary_start_type()

static static_resources_required()

1.1.6.10 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=<El*, *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: 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*

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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

1.1.6.11 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

TRAFFIC_IDENTIFIER = 'LPG_EVENT_STREAM'

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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 ResourceContainer

1.1.6.12 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 microseconds 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

1.1.6.13 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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

enable_recording(*new_state=True*)

generate_data_specification(*spec*, *placement*)

Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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(AbstractConstraint)

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

send_buffer_times

1.1.6.14 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(*new_state=True*)
Enable recording of the keys sent

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

static get_cpu_usage()

static get_dtcm_usage()

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(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(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

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

static get_sdram_usage(*send_buffer_times*, *recording_enabled*, *machine_time_step*, *receive_rate*, *n_keys*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static max_send_buffer_keys_per_timestep(*send_buffer_times*, *n_keys*)

static n_regions_to_allocate(*send_buffering*, *recording*)
Get the number of regions that will be allocated

static recording_sdram_per_timestep(*machine_time_step*, *is_recording*, *receive_rate*, *send_buffer_times*, *n_keys*)

resources_required
The resources required by the vertex

Return type ResourceContainer

static send_buffer_sdram_per_timestep(*send_buffer_times*, *n_keys*)

send_buffer_times

send_buffers

update_buffer(*arg*)

virtual_key

1.1.6.15 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*)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

edges_and_partitions()

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

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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(AbstractConstraint)

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

class spinn_front_end_common.utility_models.**CommandSenderMachineVertex**(*label*, *constraints*)

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*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS
Bases: enum.Enum

COMMANDS_AT_START_RESUME = 2

COMMANDS_AT_STOP_PAUSE = 3

COMMANDS_WITH_ARBITRARY_TIMES = 1

PROVENANCE_REGION = 4

SYSTEM_REGION = 0

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(*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(*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(*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

edges_and_partitions()

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_edges_and_partitions(*pre_vertex*, *edge_type*)

static get_n_command_bytes(*commands*)

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(AbstractConstraint)

get_timed_commands_bytes()

resources_required
The resources required by the vertex

Return type ResourceContainer

class spinn_front_end_common.utility_models.**ChipPowerMonitor**(*label*, *constraints*, *n_samples_per_recording*, *sampling_frequency*)

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*

Represents idle time recording code in a application graph.

Parameters

- **label** (*str*) – vertex label
- **constraints** – constraints for the vertex
- **n_samples_per_recording** (*int*) – 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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(**args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

class spinn_front_end_common.utility_models.**ChipPowerMonitorMachineVertex**(**args*, ***kwargs*)

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

SAMPLE_RECORDING_REGION = 0

static binary_file_name()
Return the string binary file name

Returns basestring

static binary_start_type()
The type of binary that implements this vertex

Returns starttype enum

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_recorded_data(*placement*, *buffer_manager*)
Get data from SDRAM given placement and buffer manager

Parameters

- **placement** – the location on machine 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(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(*time_step*, *time_scale_factor*, *n_samples_per_recording*, *sampling_frequency*)
Get the resources used by this vertex

Returns Resource container

n_samples_per_recording

resources_required
The resources required by the vertex

Return type ResourceContainer

sampling_frequency

class spinn_front_end_common.utility_models.**DataSpeedUpPacketGather**(*x*, *y*, *ip_address*, *extra_monitors_by_chip*, *report_default_directory*, *write_data_speed_up_reports*, *constraints=None*)
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*

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover

- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

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

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

machine_vertex

n_atoms
The number of atoms in the vertex

Return type int

class spinn_front_end_common.utility_models.**DataSpeedUpPacketGatherMachineVertex**(*x*, *y*, *extra_monitors_*, *ip_address*, *report_default_*, *write_data_sp*, *constraints=None*)

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.interface.provenance.abstract_provides_local_provenance_data.AbstractProvidesLocalProvenanceData*

BASE_KEY = 4294967289

BASE_MASK = 4294967291

END_FLAG_KEY = 4294967286

END_FLAG_KEY_OFFSET = 3

FIRST_DATA_KEY = 4294967287

FIRST_DATA_KEY_OFFSET = 2

IN_REPORT_NAME = 'speeds_gained_in_speed_up_process.rpt'

LAST_MESSAGE_FLAG_BIT_MASK = 2147483648

LONG_TIMEOUT = (14, 14)

MISSING_SEQ_NUMS_END_FLAG = 4294967295

NEW_SEQ_KEY = 4294967288

NEW_SEQ_KEY_OFFSET = 1

OUT_REPORT_NAME = 'routers_used_in_speed_up_process.rpt'

SEQUENCE_NUMBER_MASK = 2147483647

SHORT_TIMEOUT = (1, 1)

TEMP_TIMEOUT = (15, 4)

TIMEOUT_PER_RECEIVE_IN_SECONDS = 1

TIME_OUT_FOR_SENDING_IN_SECONDS = 0.01

TRAFFIC_TYPE = 2

TRANSMISSION_THROTTLE_TIME = 1e-06

ZERO_TIMEOUT = (0, 0)

calculate_max_seq_num()
Deduce the max sequence number expected to be received

Returns int of the biggest sequence num expected

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_data(*placement*, *memory_address*, *length_in_bytes*, *fixed_routes*)
Gets data from a given core and memory address.

Parameters

- **placement** – placement object for where to get data from
- **memory_address** – the address in SDRAM to start reading from
- **length_in_bytes** – the length of data to read in bytes
- **fixed_routes** – the fixed routes, used in the report of which chips were used by the speed up process

Returns byte array of the data

get_local_provenance_data()
Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

static load_application_routing_tables(*transceiver*, *extra_monitor_cores*, *placements*)
Set all chips to have application table loaded in the router

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static load_system_routing_tables(*transceiver*, *extra_monitor_cores*, *placements*)
Set all chips to have the system table loaded in the router

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static locate_correct_write_data_function_for_chip_location(*uses_advanced_monitors*, *machine*, *x*, *y*, *transceiver*, *extra_monitor_cores_to_ethernet_connec*
supports other components figuring out which gather and function to call for writing data onto spinnaker

Parameters

- **uses_advanced_monitors** (*bool*) – Whether the system is using advanced monitors
- **machine** – the SpiNNMachine instance
- **x** – the chip x coordinate to write data to
- **y** – the chip y coordinate to write data to
- **extra_monitor_cores_to_ethernet_connection_map** – mapping between cores and connections
- **transceiver** – the SpiNNMan instance

Returns a write function of either a LPG or the spinnMan

Return type func

resources_required
The resources required by the vertex

Return type ResourceContainer

send_data_into_spinnaker(*x*, *y*, *base_address*, *data*, *n_bytes=None*, *offset=0*, *cpu=0*, *is_filename=False*)
sends a block of data into SpiNNaker to a given chip

Parameters

- **x** – chip x for data
- **y** – chip y for data
- **base_address** – the address in SDRAM to start writing memory
- **data** – the data to write
- **n_bytes** – how many bytes to read, or None if not set
- **offset** – where in the data to start from
- **is_filename** (*bool*) – whether data is actually a file.

Return type None

set_cores_for_data_streaming(*transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

static static_resources_required()

static streaming(*gatherers*, *transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming via a Python context manager (i.e., using the 'with' statement).

Parameters

- **gatherers** – All the gatherers that are to be set
- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type a context manager

unset_cores_for_data_streaming(*transceiver*, *extra_monitor_cores*, *placements*)
Helper method for setting the router timeouts to a state usable for data streaming

Parameters

- **transceiver** – the SpiNNMan instance
- **extra_monitor_cores** – the extra monitor cores to set
- **placements** – placements object

Return type None

class spinn_front_end_common.utility_models.**ExtraMonitorSupport**(*constraints*)

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*

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Return type int

class spinn_front_end_common.utility_models.**ExtraMonitorSupportMachineVertex**(*constraints*, *reinject_multicast=Non* *reinject_point_to_point=* *reinject_nearest_neighb* *reinject_fixed_route=Fa*

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*

Parameters

- **constraints** – constraints on this vertex
- **reinject_multicast** – if we reinject multicast packets; defaults to value of *enable_reinjection* setting in configuration file
- **reinject_point_to_point** – if we reinject point-to-point packets
- **reinject_nearest_neighbour** – if we reinject nearest-neighbour packets
- **reinject_fixed_route** – if we reinject fixed route packets

clear_reinjection_queue(*transceiver*, *placements*, *extra_monitor_cores_to_set*)
Clears the queues for reinjection

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_reinjection_status(*placements*, *transceiver*)
Get the reinjection status from this extra monitor vertex

Parameters

- **transceiver** – the spinnMan interface
- **placements** – the placements object

Returns the reinjection status for this vertex

get_reinjection_status_for_vertices(*placements*, *extra_monitor_cores_for_data*, *transceiver*)
Get the reinjection status from a set of extra monitor cores

Parameters

- **placements** – the placements object
- **extra_monitor_cores_for_data** – the extra monitor cores to get status from
- **transceiver** – the spinnMan interface

Return type None

load_application_mc_routes(*placements*, *extra_monitor_cores_for_data*, *transceiver*)
Get the extra monitor cores to load up the application-based multicast routes (used by data in).

Parameters

- **placements** (Placements) – the placements object
- **extra_monitor_cores_for_data** (iterable(*ExtraMonitorSupportMachineVertex*)) – the extra monitor cores to get status from
- **transceiver** (Transceiver) – the spinnMan interface

Return type None

load_system_mc_routes(*placements*, *extra_monitor_cores_for_data*, *transceiver*)
Get the extra monitor cores to load up the system-based multicast routes (used by data in).

Parameters

- **placements** (Placements) – the placements object
- **extra_monitor_cores_for_data** (iterable(*ExtraMonitorSupportMachineVertex*)) – the extra monitor cores to get status from
- **transceiver** (Transceiver) – the spinnMan interface

Return type None

placement

reinject_fixed_route

reinject_multicast

reinject_nearest_neighbour

reinject_point_to_point

reset_reinjection_counters(*transceiver*, *placements*, *extra_monitor_cores_to_set*)
Resets the counters for reinjection

resources_required
The resources required by the vertex

Return type ResourceContainer

set_reinjection_packets(*placements*, *extra_monitor_cores_for_data*, *transceiver*, *point_to_point=None*, *multicast=None*, *nearest_neighbour=None*, *fixed_route=None*)

Parameters

- **placements** – placements object

- **extra_monitor_cores_for_data** – the extra monitor cores to set the packets of
- **transceiver** – spinnman instance
- **point_to_point** (*bool or None*) – If point to point should be set, or None if left as before
- **multicast** (*bool or None*) – If multicast should be set, or None if left as before
- **nearest_neighbour** (*bool or None*) – If nearest neighbour should be set, or None if left as before
- **fixed_route** (*bool or None*) – If fixed route should be set, or None if left as before.

Return type None

set_router_emergency_timeout(*timeout*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

Sets the timeout of the routers

Parameters

- **timeout** (*(int, int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (Transceiver) – the spinnMan instance
- **placements** (Placements) – the placements object
- **extra_monitor_cores_to_set** – the set of vertices to change the local chip for.

set_router_time_outs(*timeout*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

Supports setting of the router time outs for a set of chips via their extra monitor cores.

Parameters

- **timeout** (*(int, int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (Transceiver) – the spinnman interface
- **placements** (Placements) – placements object
- **extra_monitor_cores_to_set** – which vertices to use

Return type None

static static_get_binary_file_name()

static static_get_binary_start_type()

static static_resources_required()

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: 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*

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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

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

TRAFFIC_IDENTIFIER = 'LPG_EVENT_STREAM'

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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 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 microseconds 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=922*, *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*, *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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

enable_recording(*new_state=True*)

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

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

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(AbstractConstraint)

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type ResourceContainer

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Return type int

send_buffer_times

class spinn_front_end_common.utility_models.**ReverseIPTagMulticastSourceMachineVertex**(**args*, ***kwarg*

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(*new_state=True*)
Enable recording of the keys sent

generate_data_specification(**args*, ***kwargs*)
Generate a data specification.

Parameters

- **spec** (*Placement*) – 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 *ExecutableType*

static get_cpu_usage()

static get_dtcm_usage()

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(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(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

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

static get_sdram_usage(*send_buffer_times*, *recording_enabled*, *machine_time_step*, *receive_rate*, *n_keys*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static max_send_buffer_keys_per_timestep(*send_buffer_times*, *n_keys*)

static n_regions_to_allocate(*send_buffering*, *recording*)
Get the number of regions that will be allocated

static recording_sdram_per_timestep(*machine_time_step*, *is_recording*, *receive_rate*, *send_buffer_times*, *n_keys*)

resources_required
The resources required by the vertex

Return type ResourceContainer

static send_buffer_sdram_per_timestep(*send_buffer_times*, *n_keys*)

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

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

U

V

W

Z