SpiNNFrontEndCommonEnd Documentation

Release 6.0.0

Apr 08, 2021

Contents

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

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

Contents:

CHAPTER 1

spinn_front_end_common

1.1 spinn_front_end_common package

1.1.1 Subpackages

1.1.1.1 spinn_front_end_common.abstract_models package

Subpackages

spinn_front_end_common.abstract_models.impl package

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

How to manage the allocation of a machine so that it gets cleaned up neatly when the script dies.

Parameters **thread_name** (*str*) –

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

Support for a vertex that simplifies generating a data specification.

generate_data_specification(*spec*, *placement*, *machine_graph*, *routing_info*, *tags*, *machine_time_step*, *time_scale_factor*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **machine_graph** (*MachineGraph*) – (Injected)
- **routing_info** (*RoutingInfo*) – (Injected)
- **tags** (*Tags*) – (Injected)
- **machine_time_step** (*int*) – (Injected)
- **time_scale_factor** (*int*) – (Injected)

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** (*Placement*) – Where this node is on the SpiNNaker machine.
- **machine_graph** (*MachineGraph*) – The graph containing this node.
- **routing_info** (*RoutingInfo*) – The routing info.
- **iptags** (*iterable(IPTag) or None*) – The (forward) IP tags for the vertex, if any
- **reverse_iptags** (*iterable(ReverseIPTag) or None*) – The reverse IP tags for the vertex, if any
- **machine_time_step** (*int*) – The machine time step
- **time_scale_factor** (*int*) – The time step scaling factor

Return type None

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(*routing_info*, *partition*)

Returns a list of atom to key mapping.

Parameters

- **routing_info** (*RoutingInfo*) – the routing info object to consider
- **partition** (*OutgoingEdgePartition*) – the routing partition to handle.

Returns a iterable of tuples of atom IDs to keys.

Return type iterable(tuple(int,int))

class spinn_front_end_common.abstract_models.impl.**TDMAAwareApplicationVertex**(*label*, *constraints*, *max_atoms_per_co*, *splitter=None*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex

An application vertex that contains the code for using TDMA to spread packet transmission to try to avoid overloading any SpiNNaker routers.

Parameters

- **label** (*str or None*) – The name of the vertex.
- **constraints** (*iterable(AbstractConstraint) or None*) – The initial constraints of the vertex.
- **max_atoms_per_core** (*int*) – The max number of atoms that can be placed on a core, used in partitioning.

Raises **PacmanInvalidParameterException** – If one of the constraints is not valid

find_n_phases_for(*machine_graph*, *n_keys_map*)

Compute the number of phases needed for this application vertex. This is the maximum number of packets any machine vertex created by this application vertex can send in one simulation time step.

Parameters

- **machine_graph** (*MachineGraph*) –
- **n_keys_map** (*AbstractMachinePartitionNKeysMap*) –

Return type int

generate_tdma_data_specification_data(*vertex_index*)

Generates the TDMA configuration data needed for the data spec

Parameters **vertex_index** (*int*) – the machine vertex index in the pop

Returns array of data to write.

Return type list(int)

get_n_cores()

Get the number of cores this application vertex is using in the TDMA.

Returns the number of cores to use in the TDMA

Return type int

get_tdma_provenance_item(*names*, *x*, *y*, *p*, *tdma_slots_missed*)

Get the provenance item used for the TDMA provenance

Parameters

- **names** (*list(str)*) – the names for the provenance data item
- **x** (*int*) – chip x
- **y** (*int*) – chip y
- **p** (*int*) – processor id
- **tdma_slots_missed** (*int*) – the number of TDMA slots missed

Returns the provenance data item

Return type *ProvenanceDataItem*

set_initial_offset(*new_value*)

Sets the initial offset

Parameters **new_value** (*int*) – the new initial offset, in clock ticks

set_other_timings(*clocks_between_cores*, *n_slots*, *clocks_between_spikes*, *n_phases*, *clocks_per_cycle*)
Sets the other timings needed for the TDMA.

Parameters

- **clocks_between_cores** (*int*) – clock cycles between cores
- **n_slots** (*int*) – the number of slots
- **clocks_between_spikes** (*int*) – the clock cycles to wait between spikes
- **n_phases** (*int*) – the number of phases
- **clocks_per_cycle** (*int*) – the number of clock cycles per TDMA cycle

tdma_sdram_size_in_bytes
The number of bytes needed by the TDMA data

Return type int

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** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at

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** (*float*) – The total run time that is now required starting from when the machine was first allocated

where_is_machine(*chip_x*, *chip_y*)
Locates and returns cabinet, frame, board for a given chip in a machine allocated to this job.

Parameters

- **chip_x** (*int*) – chip x location
- **chip_y** (*int*) – chip y location

Returns (cabinet, frame, board)

Return type tuple(int,int,int)

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 partition that goes into a vertex of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – 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** (*RoutingInfo*) – the routing info object to consider
- **partition** (*OutgoingEdgePartition*) – the routing partition to handle.

Returns a iterable of tuples of atom IDs to keys.

Return type iterable(tuple(int,int))

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

A vertex that can provide constraints for its outgoing edge partitions.

If a Machine_vertex is an instance the Application vertex will not be checked. However if the MachineVertex does not implement this API ProcessPartitionConstraint will then check the ApplicationVertex

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge partition that comes out of this vertex.

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

Returns A list of constraints

Return type list(AbstractConstraint)

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

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

Parameters

- **spec** (*DataSpecificationGenerator*) – Where to write the regenerated spec
- **placement** (*Placement*) – Where are we regenerating for?

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

Return type bool

set_reload_required(*new_value*)
Indicate that the regions have been reloaded

Parameters **new_value** – the new value

Return type None

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

A device that may be a virtual vertex which wants to commands to be sent to it as multicast packets at fixed points in the simulation.

Note: The device might not be a vertex at all. It could instead be instantiated entirely host side, in which case these methods will never be called.

---

pause_stop_commands
The commands needed when pausing or stopping simulation

Return type iterable(*MultiCastCommand*)

start_resume_commands
The commands needed when starting or resuming simulation

Return type iterable(*MultiCastCommand*)

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

Return type iterable(*MultiCastCommand*)

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.

Return type bool

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

Return type iterable(ApplicationVertex)

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for a particular dependent vertex.

Parameters **vertex** (*ApplicationVertex*) –

Return type iterable(str)

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

Indicates an object that can be reset to time 0.

This is used when AbstractSpinnakerBase.reset is called. All Vertices and all edges in the original graph (the one added to by the user) will be checked and reset.

reset_to_first_timestep()
Reset the object to first time step.

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

Marks a vertex that can provide information about bitfields it wants generated on-chip.

bit_field_base_address(*transceiver*, *placement*)
Returns the SDRAM address for the bit field table data.

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Returns the SDRAM address for the bitfield address

Return type int

bit_field_builder_region(*transceiver*, *placement*)
returns the SDRAM address for the bit field builder data

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Returns the SDRAM address for the bitfield builder data

Return type int

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

Marks a machine vertex that can support having the on-chip bitfield compressor running on its core.

bit_field_base_address(*transceiver*, *placement*)
Returns the SDRAM address for the bit-field table data.

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Returns the SDRAM address for the bitfield address

Return type int

key_to_atom_map_region_base_address(*transceiver*, *placement*)
Returns the SDRAM address for the region that contains key-to-atom data.

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Returns the SDRAM address for the key-to-atom data

Return type int

regeneratable_sdram_blocks_and_sizes(*transceiver*, *placement*)
Returns the SDRAM addresses and sizes for the cores' SDRAM that are available (borrowed) for generating bitfield tables.

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Returns list of tuples containing (the SDRAM address for the cores SDRAM address's for the core's SDRAM that can be used to generate bitfield tables loaded, and the size of memory chunks located there)

Return type list(tuple(int,int))

1.1.1.2 spinn_front_end_common.common_model_binaries package

Module contents

This module contains no python code

1.1.1.3 spinn_front_end_common.interface package

Subpackages

spinn_front_end_common.interface.buffer_management package

Subpackages

spinn_front_end_common.interface.buffer_management.buffer_models package

Module contents

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

Indicates that this MachineVertex 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** (*Transceiver*) – the SpiNNMan instance
- **placement** (*Placement*) – the placement object of the core to find the address of

Returns the base address of the recording region

Return type int

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.

Return type bool

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 *AbstractReceiveBuffersToHost* which uses an existing set of buffers for the details.

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

Return type bool

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_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

send_buffers

Return type dict(int, *BufferedSendingRegion*)

spinn_front_end_common.interface.buffer_management.storage_objects package

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 SQLite 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.

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

Return type None

clear_region(*x*, *y*, *p*, *region*)
Clears the data for a single region.

Note: This method *loses information!*

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 cleared

Returns True if any region was changed

Return type bool

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

a buffer containing all the data received during the simulation, and a flag indicating if any data was missing

Note: Implementations should not assume that the total buffer is necessarily shorter than 1GB.

Return type tuple(memoryview, bool)

store_data_in_region_buffer(*x*, *y*, *p*, *region*, *missing*, *data*)

Store some information in the corresponding buffer for a specific chip, core and recording 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
- **missing** (*bool*) – Whether any data is missing
- **data** (*bytearray*) – data to be stored

Note: Implementations may assume this to be shorter than 1GB

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedReceivingD**

Bases: object

Stores the information received through the buffering output from the SpiNNaker system.

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

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 a buffer containing all the data received during the simulation, and a flag indicating if any data was missing

Return type tuple(memoryview, bool)

get_region_information(*x*, *y*, *p*, *region_id*)
Get the size, address and is_missing of the region

Parameters

- **x** (*int*) – The x-coordinate of the core whose data this is
- **y** (*int*) – The y-coordinate of the core whose data this is
- **p** (*int*) – The processor id of the core whose data this is
- **region_id** (*int*) – The id of the region to get the data for

Return type tuple(int, int, bool)

has_region_information(*x*, *y*, *p*)
Determine if region information has been stored for this core

Parameters

- **x** (*int*) – The x-coordinate of the core whose data this is
- **y** (*int*) – The y-coordinate of the core whose data this is
- **p** (*int*) – The processor id of the core whose data this is

Return type bool

is_data_from_region_flushed(*x*, *y*, *p*, *region*)
Determine if data has been stored for this 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

reset()
Perform tasks to restart recording from time=0

resume()
Perform tasks that will continue running without resetting

store_data_in_region_buffer(*x*, *y*, *p*, *region*, *missing*, *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
- **missing** (*bool*) – Whether any data is missing

- **data** (*bytearray*) – data to be stored

store_region_information(*x*, *y*, *p*, *sizes_and_addresses*)
Store the sizes, addresses and is_missing data of the regions

Parameters

- **x** (*int*) – The x-coordinate of the core whose data this is
- **y** (*int*) – The y-coordinate of the core whose data this is
- **p** (*int*) – The processor id of the core whose data this is
- **sizes_and_addresses** (*list(int, int, bool)*) – The size and address of each region

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** (*bool*) – the time stamp to check if there's still keys to transmit

is_next_timestamp

Determines if the region is empty. 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**(*r*
s
n

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** (*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(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.**SqlLiteDatabase**(*da*

Bases: *spinn_front_end_common.utilities.sqlite_db.SQLiteDB*, spinn_utilities.abstract_context_manager.AbstractContextManager

Specific implementation of the Database for SQLite 3.

Note: *Not thread safe on the same database file!* Threads can access different DBs just fine.

Parameters **database_file** (*str*) – The name of a file that contains (or will contain) an SQLite database holding the data. If omitted, an unshared in-memory database will be used.

clear()

Clears the data for all regions.

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

Return type None

clear_region(*x*, *y*, *p*, *region*)

Clears the data for a single region.

Note: This method *loses information!*

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 cleared

Returns True if any region was changed

Return type bool

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

a buffer containing all the data received during the simulation, and a flag indicating if any data was missing

Note: Implementations should not assume that the total buffer is necessarily shorter than 1GB.

Return type tuple(memoryview, bool)

store_data_in_region_buffer(*x*, *y*, *p*, *region*, *missing*, *data*)

Store some information in the corresponding buffer for a specific chip, core and recording 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
- **missing** (*bool*) – Whether any data is missing
- **data** (*bytearray*) – data to be stored

Note: Implementations may assume this to be shorter than 1GB

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
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – The monitors.

- **packet_gather_cores_to_ethernet_connection_map** (*dict(tuple(int,int), DataSpeedUpPacketGatherMachineVertex)*) – mapping of cores to the gatherer vertex placed on them
- **extra_monitor_to_chip_mapping** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) –
- **machine** (*Machine*) –
- **fixed_routes** (*dict(tuple(int,int),FixedRouteEntry)*) –
- **uses_advanced_monitors** (*bool*) –
- **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** (AbstractReceiveBuffersToHost) – 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 tuple(bytearray, bool)

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

Parameters

- **placements** (*Placements*) – Where to get the data from.
- **progress** (*ProgressBar or None*) – How to measure/display the progress.

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.

Return type iterable(*AbstractSendsBuffersFromHost*)

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

spinn_front_end_common.interface.ds package

Module contents

class spinn_front_end_common.interface.ds.**DataRowWriter**(*x*, *y*, *p*, *targets*)
Bases: io.RawIOBase

close()
Closes the writer if not already closed.

fileno()
Returns underlying file descriptor if one exists.

OSError is raised if the IO object does not use a file descriptor.

readable()
Return whether object was opened for reading.

If False, read() will raise OSError.

seekable()
Return whether object supports random access.

If False, seek(), tell() and truncate() will raise OSError. This method may need to do a test seek().

truncate(*size=None*)
Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by tell(). Returns the new size.

writable()
Return whether object was opened for writing.

If False, write() will raise OSError.

write(*data*)

class spinn_front_end_common.interface.ds.**DataSpecificationTargets**(*machine*, *report_folder*, *init=None*, *clear=True*)
Bases: collections.abc.MutableMapping

Parameters

- **machine** (*Machine*) –
- **report_folder** (*str*) –
- **init** (*bool or None*) –
- **clear** (*bool*) –

create_data_spec(*x*, *y*, *p*)

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **p** (*int*) –

Return type *DataRowWriter*

get_app_id(*x*, *y*, *p*)
Gets the app_id set for this core

Parameters

- **x** (*int*) – core x
- **y** (*int*) – core y
- **p** (*int*) – core p

Return type int

get_database()
Expose the database so it can be shared

Return type DsAbstractDatabase

items()

Returns iterator over the core locations and how to read the data spec for them

Return type iterable(tuple(tuple(int,int,int),RawIOBase))

keys()
Yields the keys.

As the more typical call is iteritems this makes use of that

Return type iterable(tuple(int,int,int))

mark_system_cores(*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) –

n_targets()
TEMP implementation

Returns

set_app_id(*app_id*)
Sets the same app_id for all rows that have DS content

Parameters **app_id** (*int*) – value to set

write_data_spec(*x*, *y*, *p*, *ds*)

Parameters

- **x** (*int*) –

- **y** (*int*) –
- **p** (*int*) –
- **ds** (*bytearray*) –

class spinn_front_end_common.interface.ds.**DsWriteInfo**(*database*)

Bases: object

Parameters **database** (*DsAbstractDatabase*) – Database to map

clear_write_info()

Clears the info for all cores,

get_info(*x*, *y*, *p*)

gets the info for the core x, y, p

Parameters

- **x** (*int*) – core x
- **y** (*int*) – core y
- **p** (*int*) – core p

Returns dict with the keys start_address, memory_used and memory_written

Return type dict(str,int)

items()

iteritems()

keys()

Yields the keys.

As the more typical call is iteritems this makes use of that

Return type iterable(tuple(int,int,int))

set_info(*x*, *y*, *p*, *info*)

Sets the info for the core x, y, p

Parameters

- **x** (*int*) – core x
- **y** (*int*) – core y
- **p** (*int*) – core p
- **info** (*dict(str,int)*) – dict with the keys start_address, memory_used and memory_written

set_size_info(*x*, *y*, *p*, *memory_used*)

Sets the size info for the core x, y p.

Parameters

- **x** – core x
- **y** – core y
- **p** – core p
- **memory_used** – memory allocated

Rtype None

spinn_front_end_common.interface.interface_functions package

Module contents

The code in this module is intended primarily for being invoked via the PACMAN Executor.

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

Handles finishing the running of an application, collecting the status of the cores that the application was running on.

__call__(*app_id*, *txrx*, *executable_types*)

Parameters

- **app_id** (*int*) –
- **txrx** (*Transceiver*) –
- **executable_types** (*dict(ExecutableType, CoreSubsets)*) –

Raises ***ExecutableFailedToStopException*** –

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

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

__call__(*buffer_manager*, *notification_interface*, *executable_types*, *app_id*, *txrx*, *runtime*, *time_scale_factor*, *no_sync_changes*, *time_threshold*, *machine*, *run_until_complete=False*)

Parameters

- **buffer_manager** (BufferManager) –
- **notification_interface** (NotificationProtocol) –
- **executable_types** (*dict(ExecutableType, CoreSubsets)*) –
- **app_id** (*int*) –
- **txrx** (*Transceiver*) –
- **runtime** (*int*) –
- **time_scale_factor** (*int*) –
- **no_sync_changes** (*int*) – Number of synchronisation changes
- **time_threshold** (*int*) –
- **machine** (*Machine*) – the spinn machine instance
- **run_until_complete** (*bool*) –

Returns Number of synchronisation changes

Return type int

Raises ***ConfigurationException*** –

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

Extracts data in between runs.

__call__(*machine_graph*, *placements*, *buffer_manager*)

Parameters

- **machine_graph** (*MachineGraph*) –
- **placements** (*Placements*) –
- **buffer_manager** (BufferManager) –

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

Creates a buffer manager.

__call__(*placements*, *tags*, *txrx*, *uses_advanced_monitors*, *report_folder*, *extra_monitor_cores=None*, *extra_monitor_to_chip_mapping=None*, *packet_gather_cores_to_ethernet_connection_map=None*, *machine=None*, *fixed_routes=None*, *java_caller=None*)

Parameters

- **placements** (*Placements*) –
- **tags** (*Tags*) –
- **txrx** (*Transceiver*) –
- **uses_advanced_monitors** (*bool*) –
- **report_folder** (*str*) – The path where the SQLite database holding the data will be placed, and where any java provenance can be written.
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) –
- **extra_monitor_to_chip_mapping** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) –
- **packet_gather_cores_to_ethernet_connection_map** (*dict(tuple(int,int),DataSpeedUpPacketGatherMachineVertex)*) –
- **machine** (*Machine*) –
- **fixed_routes** (*dict(tuple(int,int),FixedRouteEntry)*) –
- **java_caller** (JavaCaller) –

Return type *BufferManager*

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

Clears the logging output buffer of an application running on a SpiNNaker machine.

__call__(*transceiver*, *executable_types*)

Parameters

- **transceiver** (*Transceiver*) –
- **executable_types** (*dict(ExecutableType,CoreSubsets)*) –

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

Extract the logging output buffers from the machine, and separates lines based on their prefix.

__call__(*transceiver*, *executable_targets*, *executable_finder*, *app_provenance_file_path=None*, *system_provenance_file_path=None*, *from_cores='ALL'*, *binary_types=None*)

Parameters

- **transceiver** (*Transceiver*) –
- **executable_targets** (*ExecutableTargets*) –
- **executable_finder** (ExecutableFinder) –
- **app_provenance_file_path** (*str or None*) –
- **system_provenance_file_path** (*str or None*) –
- **from_cores** (*str*) –
- **binary_types** (*str*) –

Returns error_entries, warn_entries

Return type tuple(list(str),list(str))

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

Forces all cores to generate provenance data, and then exit.

__call__(*txrx*, *app_id*, *all_core_subsets*)

Parameters

- **txrx** (*Transceiver*) –
- **app_id** (*int*) –
- **all_core_subsets** (*CoreSubsets*) –

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

Updates the runtime of an application running on a SpiNNaker machine.

__call__(*txrx*, *app_id*, *executable_types*, *run_until_timesteps*, *current_timesteps*, *n_sync_steps*)

Parameters

- **transceiver** (*Transceiver*) –
- **app_id** (*int*) –
- **executable_types** (*dict(ExecutableType,CoreSubsets)*) –
- **run_until_timesteps** (*int or None*) –
- **current_timesteps** (*int*) –
- **n_sync_steps** (*int or None*) –

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

Builds the notification protocol for GUI and external device interaction.

__call__(*wait_for_read_confirmation*, *socket_addresses*, *database_file_path*)

Parameters

- **wait_for_read_confirmation** (*bool*) –
- **socket_addresses** (*list(SocketAddress)*) – Where to notify.
- **database_file_path** (*str*) –

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

This algorithm does the actual work of computing energy used by a simulation (or other application) running on SpiNNaker.

JOULES_PER_SPIKE = 8e-10
stated in papers (SpiNNaker: A 1-W 18 core system-on-Chip for Massively-Parallel Neural Network Simulation)

MILLIWATTS_FOR_BOXED_48_CHIP_FRAME_IDLE_COST = 0.0045833333
measured from the real power meter and timing between the photos for a day powered off

MILLIWATTS_FOR_FRAME_IDLE_COST = 0.117
measured from the real power meter and timing between the photos for a days powered off

MILLIWATTS_PER_CHIP_ACTIVE_OVERHEAD = 0.0006399999999999999
stated in papers (SpiNNaker: A 1-W 18 core system-on-Chip for Massively-Parallel Neural Network Simulation)

MILLIWATTS_PER_FPGA = 0.000584635
given from Indar's measurements

MILLIWATTS_PER_FRAME_ACTIVE_COST = 0.154163558
measured from the loading of the column and extrapolated

MILLIWATTS_PER_IDLE_CHIP = 0.00036
stated in papers (SpiNNaker: A 1-W 18 core system-on-Chip for Massively-Parallel Neural Network Simulation)

MILLIWATTS_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST = 0.01666667

N_MONITORS_ACTIVE_DURING_COMMS = 2

__call__(*placements*, *machine*, *version*, *time_scale_factor*, *router_provenance*, *runtime*, *buffer_manager*, *mapping_time*, *load_time*, *execute_time*, *dsg_time*, *extraction_time*, *spalloc_server=None*, *remote_spinnaker_url=None*, *machine_allocation_controller=None*)

Parameters

- **placements** (*Placements*) –
- **machine** (*Machine*) –
- **version** (*int*) – The version of the SpiNNaker boards in use.
- **time_scale_factor** (*int*) –
- **router_provenance** (*list(ProvenanceDataItem)*) – Provenance information from routers.
- **runtime** (*float*) –
- **buffer_manager** (BufferManager) –
- **mapping_time** (*float*) – From simulator via *FinaliseTimingData*.
- **load_time** (*float*) – From simulator via *FinaliseTimingData*.
- **execute_time** (*float*) – From simulator via *FinaliseTimingData*.
- **dsg_time** (*float*) – From simulator via *FinaliseTimingData*.
- **extraction_time** (*float*) – From simulator via *FinaliseTimingData*.
- **spalloc_server** (*str or None*) – (optional)

- **remote_spinnaker_url** (*str or None*) – (optional)
- **machine_allocation_controller** (MachineAllocationController) – (optional)

Return type *PowerUsed*

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

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

__call__(*machine_graph*, *user_create_database*, *tags*, *runtime*, *machine*, *data_n_timesteps*, *time_scale_factor*, *machine_time_step*, *placements*, *routing_infos*, *router_tables*, *report_folder*, *create_atom_to_event_id_mapping=False*, *application_graph=None*)

Parameters

- **machine_graph** (*MachineGraph*) –
- **user_create_database** (*str*) –
- **tags** (*Tags*) –
- **runtime** (*int*) –
- **machine** (*Machine*) –
- **data_n_timesteps** (*int*) –
- **time_scale_factor** (*int*) –
- **machine_time_step** (*int*) –
- **placements** (*Placements*) –
- **routing_infos** (*RoutingInfo*) –
- **router_tables** (*MulticastRoutingTables*) –
- **report_folder** (*str*) – Where the database will be put.
- **create_atom_to_event_id_mapping** (*bool*) –
- **application_graph** (*ApplicationGraph*) –

Returns Database interface, where the database is located

Return type tuple(*DatabaseInterface*, str)

database_file_path

Return type str or None

needs_database

Return type bool

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

Generates routing table entries used by the data in processes with the extra monitor cores.

__call__(*machine*, *extra_monitor_cores*, *placements*)

Parameters

- **machine** (*Machine*) –

- **extra_monitor_cores** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) –
- **placements** (*Placements*) –

Returns routing tables, destination-to-key map, board-locn-to-timeout-key map

Return type tuple(MulticastRoutingTables, dict(tuple(int,int),int), dict(tuple(int,int),int))

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

Regenerates and reloads the data specifications.

__call__(*transceiver*, *placements*, *hostname*, *report_directory*, *write_text_specs*)

Parameters

- **transceiver** (*Transceiver*) – SpiNNMan transceiver for communication
- **placements** (*Placements*) – the list of placements of the machine graph to cores
- **hostname** (*str*) – the machine name
- **report_directory** (*str*) – the location where reports are stored
- **write_text_specs** (*bool*) – Whether the textual version of the specification is to be written

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

Works out the number of keys needed for each edge.

ERROR_MSG = 'A machine graph is required for this mapper. Please choose and try again'

PROG_BAR_NAME = 'Getting number of keys required by each edge using application graph'

__call__(*machine_graph*)

Parameters **machine_graph** (*MachineGraph*) –

Return type DictBasedMachinePartitionNKeysMap

Raises ***ConfigurationException*** – If no graph is available

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

Converts the power usage information into provenance data.

__call__(*power_used*, *placements*)

Parameters

- **power_used** (PowerUsed) – The computed basic power consumption information
- **placements** (*Placements*) – Used for describing what a core was actually doing

Return type list(*ProvenanceDataItem*)

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

Produces the timing information for the run.

__call__()

Returns mapping_time, dsg_time, load_time, execute_time, extraction_time

Return type tuple(float, float, float, float, float)

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

Builds a set of stats on how many chips were used for application cores.

__call__(*placements*)
Finds how many application chips there were and the cost on each chip

Parameters **placements** (*Placements*) – placements

Returns

a tuple with 4 elements.

1. how many chips were used
2. the max application cores on any given chip
3. the lowest number of application cores on any given chip
4. the average number of application cores on any given chip

Return type tuple(int,int,int,float)

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

Extracts binaries to be executed.

__call__(*placements*, *graph*, *executable_finder*)

Parameters

- **placements** (*Placements*) –
- **graph** (*MachineGraph*) –
- **executable_finder** (*ExecutableFinder*) –

Return type ExecutableTargets

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

Executes the data specification generation step.

__call__(*placements*, *hostname*, *report_default_directory*, *write_text_specs*, *machine*, *data_n_timesteps*, *placement_order=None*)

Parameters

- **placements** (*Placements*) – placements of machine graph to cores
- **hostname** (*str*) – SpiNNaker machine name
- **report_default_directory** (*str*) – the location where reports are stored
- **write_text_specs** (*bool*) – True if the textual version of the specification is to be written
- **machine** (*Machine*) – the python representation of the SpiNNaker machine
- **data_n_timesteps** (*int*) – The number of timesteps for which data space will been reserved
- **placement_order** (*list(Placement)*) – the optional order in which placements should be examined

Returns DSG targets (map of placement tuple and filename)

Return type tuple(*DataSpecificationTargets*, dict(tuple(int,int,int), int))

Raises ***ConfigurationException*** – If the DSG asks to use more SDRAM than is available.

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

Works out how many chips a machine graph needs.

__call__(*machine_graph*, *machine*, *plan_n_timesteps*)

Parameters

- **machine_graph** (*MachineGraph*) – The machine_graph to measure.
- **machine** (*Machine*) – The machine with respect to which to partition the application graph.
- **plan_n_timesteps** (*int*) – Number of timesteps to plan for.

Returns The size of the graph in number of chips.

Return type int

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

Gets provenance information from the graphs.

__call__(*machine_graph*, *application_graph=None*)

Parameters

- **machine_graph** (*MachineGraph*) – The machine graph to inspect
- **application_graph** (*ApplicationGraph*) – The optional application graph

Return type list(*ProvenanceDataItem*)

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

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

__call__(*hbp_server_url*, *total_run_time*, *n_chips=None*, *n_boards=None*)

Parameters

- **hbp_server_url** (*str*) – The URL of the HBP server from which to get the machine
- **total_run_time** (*int*) – The total run time to request
- **n_chips** (*int*) – The number of chips required. Only used if n_boards is None
- **n_boards** (*int*) – The number of boards required

Returns machine name, machine version, BMP details (if any), reset on startup flag, auto-detect BMP, SCAMP connection details, boot port, allocation controller

Return type tuple(str, int, object, bool, bool, object, object, *MachineAllocationController*)

Raises **PacmanConfigurationException** – If neither *n_chips* or *n_boards* provided

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

Host-based fancy router compressor using the bitfield filters of the cores. Compresses bitfields and router table entries together as much as feasible.

MERGED_SETTER = 2147483648

N_ATOMS_MASK = 1073741823

__call__(*router_tables*, *machine*, *placements*, *transceiver*, *default_report_folder*, *produce_report*, *machine_graph*, *routing_infos*, *machine_time_step*, *time_scale_factor*, *target_length=None*)
Entry point when using the PACMANAlgorithmExecutor

Parameters

- **router_tables** (*MulticastRoutingTables*) – routing tables (uncompressed and unordered)
- **machine** (*Machine*) – SpiNNMachine instance
- **placements** (*Placements*) – placements
- **transceiver** (*Transceiver*) – SpiNNMan instance
- **default_report_folder** (*str*) – report folder
- **produce_report** (*bool*) – flag for producing report
- **machine_graph** (*MachineGraph*) – the machine graph level
- **routing_infos** (*RoutingInfo*) – routing information
- **machine_time_step** (*int*) – time step
- **time_scale_factor** (*int*) – time scale factor
- **target_length** (*int or None*) – length of table entries to get to.

Returns compressed routing table entries

Return type MulticastRoutingTables

static generate_key_to_atom_map(*machine_graph*, *routing_infos*)
THIS IS NEEDED due to the link from key to edge being lost.

Parameters

- **machine_graph** (*MachineGraph*) – machine graph
- **routing_infos** (*RoutingInfo*) – routing infos

Returns key to atom map based of key to n atoms

Return type dict(int,int)

generate_report_path(*default_report_folder*)

Parameters **default_report_folder** (*str*) –

Return type str

get_bit_field_sdram_base_addresses(*chip_x*, *chip_y*, *machine*, *placements*, *transceiver*)

Parameters

- **chip_x** (*int*) –
- **chip_y** (*int*) –

- **machine** (*Machine*) –
- **placements** (*Placements*) –
- **transceiver** (*Transceiver*) –

start_compression_selection_process(*router_table*, *produce_report*, *report_folder_path*, *transceiver*, *machine_graph*, *placements*, *machine*, *target_length*, *compressed_pacman_router_tables*, *key_atom_map*)

Entrance method for doing on host compression. Can be used as a public method for other compressors.

Parameters

- **router_table** (*UnCompressedMulticastRoutingTable*) – the routing table in question to compress
- **produce_report** (*bool*) – whether the report should be generated
- **report_folder_path** (*str*) – the report folder base address
- **transceiver** (*Transceiver*) – spinnMan instance
- **machine_graph** (*MachineGraph*) – machine graph
- **placements** (*Placements*) – placements
- **machine** (*Machine*) – SpiNNMan instance
- **target_length** (*int*) – length of router compressor to get to
- **compressed_pacman_router_tables** (*MulticastRoutingTables*) – a data holder for compressed tables
- **key_atom_map** (*dict(int,int)*) – key to atoms map should be allowed to handle per time step

class spinn_front_end_common.interface.interface_functions.**HBPMaxMachineGenerator**

Bases: object

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

__call__(*hbp_server_url*, *total_run_time*, *max_machine_core_reduction=0*)

Parameters

- **hbp_server_url** (*str*) – The URL of the HBP server from which to get the machine
- **total_run_time** (*int*) – The total run time to request
- **max_machine_core_reduction** (*int*) – the number of cores less than DEFAULT_MAX_CORES_PER_CHIP that each chip should have

Return type Machine

class spinn_front_end_common.interface.interface_functions.**HostExecuteDataSpecification**

Bases: object

Executes the host based data specification.

execute_application_data_specs(*transceiver*, *machine*, *app_id*, *dsg_targets*, *uses_advanced_monitors*, *executable_targets*, *region_sizes*, *placements=None*, *extra_monitor_cores=None*, *extra_monitor_cores_to_ethernet_connection_map=None*, *report_folder=None*, *java_caller=None*, *processor_to_app_data_base_address=None*, *disable_advanced_monitor_usage=False*)

Execute the data specs for all non-system targets.

Parameters

- **machine** (*Machine*) – the python representation of the SpiNNaker machine
- **transceiver** (*Transceiver*) – the spinnman instance
- **app_id** (*int*) – the application ID of the simulation
- **region_sizes** (*dict(tuple(int,int,int),int)*) – the coord for region sizes for each core
- **dsg_targets** (DataSpecificationTargets) – map of placement to file path
- **uses_advanced_monitors** (*bool*) – whether to use fast data in protocol
- **executable_targets** (*ExecutableTargets*) – what core will running what binary
- **placements** (*Placements*) – where vertices are located
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the deployed extra monitors, if any
- **extra_monitor_cores_to_ethernet_connection_map** (*dict(tuple(int,int), DataSpeedUpPacketGatherMachineVertex)*) – how to talk to extra monitor cores
- **processor_to_app_data_base_address** (*dict(tuple(int,int,int), DsWriteInfo)*) – map of placement and DSG data
- **disable_advanced_monitor_usage** (*bool*) – whether to avoid using advanced monitors even if they're available

Returns map of placement and DSG data

Return type dict(tuple(int,int,int),*DataWritten*) or *DsWriteInfo*

execute_system_data_specs(*transceiver*, *machine*, *app_id*, *dsg_targets*, *region_sizes*, *executable_targets*, *report_folder=None*, *java_caller=None*, *processor_to_app_data_base_address=None*)

Execute the data specs for all system targets.

Parameters

- **transceiver** (*Transceiver*) – the spinnman instance
- **machine** (*Machine*) – the python representation of the spinnaker machine
- **app_id** (*int*) – the application ID of the simulation
- **dsg_targets** (*dict(tuple(int,int,int),str)*) – map of placement to file path
- **region_sizes** (*dict(tuple(int,int,int),int)*) – the coordinates for region sizes for each core

- **executable_targets** (*ExecutableTargets*) – the map between binaries and locations and executable types
- **report_folder** (*str*) –
- **java_caller** (JavaCaller) –
- **processor_to_app_data_base_address** (*dict(tuple(int,int,int),* DataWritten*)*) –

Returns map of placement and DSG data, and loaded data flag.

Return type dict(tuple(int,int,int),*DataWritten*) or *DsWriteInfo*

first = True

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

Adds chip power monitors into a given graph.

__call__(*machine*, *machine_graph*, *n_samples_per_recording*, *sampling_frequency*, *application_graph=None*)

Adds chip power monitor vertices on Ethernet connected chips as required.

Parameters

- **machine** (*Machine*) – the SpiNNaker machine as discovered
- **machine_graph** (*MachineGraph*) – the machine graph
- **n_samples_per_recording** (*int*) –
- **sampling_frequency** (*int*) –
- **application_graph** (*ApplicationGraph*) – the application graph

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

Inserts edges between vertices who use MC speed up and its local MC data gatherer.

EDGE_LABEL = 'edge between {} and {}'

__call__(*machine_graph*, *placements*, *machine*, *vertex_to_ethernet_connected_chip_mapping*, *application_graph=None*)

Parameters

- **machine_graph** (*MachineGraph*) – the machine graph instance
- **placements** (*Placements*) – the placements
- **machine** (*Machine*) – the machine object
- **vertex_to_ethernet_connected_chip_mapping** (*dict(tuple(int, int),* DataSpeedUpPacketGatherMachineVertex*)*) – mapping between ethernet connected chips and packet gatherers
- **application_graph** (*ApplicationGraph*) – the application graph

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

Add edges from the recorded vertices to the local Live PacketGatherers.

__call__(*live_packet_gatherer_parameters*, *placements*, *live_packet_gatherers_to_vertex_mapping*, *machine*, *machine_graph*, *application_graph=None*, *n_keys_map=None*)

Parameters

- **live_packet_gatherer_parameters** (*dict(LivePacketGatherParameters, list(tuple(AbstractVertex, list(str))))*) – the set of parameters
- **placements** (*Placements*) – the placements object
- **live_packet_gatherers_to_vertex_mapping** (*dict(LivePacketGatherParameters, dict(tuple(int,int), LivePacketGatherMachineVertex))*) – the mapping of LPG parameters and the machine vertices associated with it
- **machine** (*Machine*) – the SpiNNaker machine
- **machine_graph** (*MachineGraph*) – the machine graph
- **application_graph** (*ApplicationGraph*) – the application graph
- **n_keys_map** (*DictBasedMachinePartitionNKeysMap*) – key map

class spinn_front_end_common.interface.interface_functions.**InsertExtraMonitorVerticesToGrap**

Bases: object

Inserts the extra monitor vertices into the graph that correspond to the extra monitor cores required.

__call__(*machine*, *machine_graph*, *default_report_directory*, *write_data_speed_up_reports*, *application_graph=None*)

Parameters

- **machine** (*Machine*) – spinnMachine instance
- **machine_graph** (*MachineGraph*) – machine graph
- **default_report_directory** (*str*) – the directory where reports go
- **write_data_speed_up_reports** (*bool*) – determine whether to write the reports for data speed up
- **n_cores_to_allocate** (*int*) – number of cores to allocate for reception
- **application_graph** (*ApplicationGraph*) – app graph

Returns vertex to Ethernet connection map, list of extra_monitor_vertices, vertex_to_chip_map

Return type tuple(dict(tuple(int,int),*DataSpeedUpPacketGatherMachineVertex*), list(*ExtraMonitorSupportMachineVertex*), dict(tuple(int,int),*ExtraMonitorSupportMachineVertex*))

class spinn_front_end_common.interface.interface_functions.**InsertLivePacketGatherersToGrapl**

Bases: object

Adds LPGs as required into a given graph.

__call__(*live_packet_gatherer_parameters*, *machine*, *machine_graph*, *application_graph=None*)

Add LPG vertices on Ethernet connected chips as required.

Parameters

- **live_packet_gatherer_parameters** (*dict(LivePacketGatherParameters, list(tuple(AbstractVertex, list(str))))*) – the Live Packet Gatherer parameters requested by the script
- **machine** (*Machine*) – the SpiNNaker machine as discovered
- **machine_graph** (*MachineGraph*) – the machine graph
- **application_graph** (*ApplicationGraph*) – the application graph

Returns mapping between LPG parameters and LPG vertex

Return type dict(*LivePacketGatherParameters*, dict(tuple(int,int),*LivePacketGatherMachineVertex*))

spinn_front_end_common.interface.interface_functions.**interface_xml**()

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

Go through the executable targets and load each binary to everywhere and then send a start request to the cores that actually use it.

static filter_targets(*targets*, *filt*)

Parameters

- **executable_targets** (*ExecutableTargets*) –
- **filt** (*callable(ExecutableType, bool)*) –

Return type tuple(list(str), ExecutableTargets)

load_app_images(*executable_targets*, *app_id*, *transceiver*)

Parameters

- **executable_targets** (*ExecutableTargets*) –
- **app_id** (*int*) –
- **transceiver** (*Transceiver*) –

load_sys_images(*executable_targets*, *app_id*, *transceiver*)

Parameters

- **executable_targets** (*ExecutableTargets*) –
- **app_id** (*int*) –
- **transceiver** (*Transceiver*) –

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

Load a set of fixed routes onto a SpiNNaker machine.

__call__(*fixed_routes*, *transceiver*, *app_id*)

Parameters

- **fixed_routes** (*dict(tuple(int, int), FixedRouteEntry)*) –
- **transceiver** (*Transceiver*) –
- **app_id** (*int*) –

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

Builds a localised TDMA which allows a number of machine vertices of the same application vertex to fire at the same time. Ensures that other application vertices are not firing at the same time. Verifies if the total time required fits into the time scale factor and machine time step. Below are text diagrams to show how this works in principle.

Figure 1: bits needed to figure out time between spikes. Cores 0-4 have 2 atoms, core 5 has 1 atom:

```
#         0     1       2       3       4       5
# T2-[    X                     X
#     |         X                       X
#     |                 X                       X
#     [   X                       X
#         |------| T
#                X                       X
#                       X <- T3

T = time_between_cores
T2 = time_between_phases
T3 = end of TDMA (equiv of ((n_phases + 1) * T2))
cutoff = 2. n_phases = 3 max_atoms = 2
```

Constants etc just to get into head:

- clock cycles = 200 Mhz = 200 = sv->cpu_clk
- 1ms = 200000 for timer 1. = clock cycles
- 200 per microsecond
- machine time step = microseconds already.
- *__time_between_cores* = microseconds.

Figure 2: initial offset (used to try to interleave packets from other app verts into the TDMA without extending the overall time, and trying to stop multiple packets in flight at same time).

Figure 3: bits needed to figure out time between spikes. Cores 0-4 have 2 atoms, core 5 has 1 atom:

```
#         0  .5   1   .5    2   .5   3    .5   4   .5   5  .5
# T2-[    X   Y                     X     Y
#     |           X   Y                         X    Y
#     |                       X    Y                      X   Y
#     [   X    Y                        X     Y
#         |-------| T
#                 X    Y                        X    Y
#                 |----| T4
#                     T3 ->  X    Y

T4 is the spreader between populations.
X is pop0 firing,
Y is pop1 firing
```

FRACTION_OF_TIME_FOR_SPIKE_SENDING = 0.8

FRACTION_OF_TIME_STEP_BEFORE_SPIKE_SENDING = 0.1

__call__(*machine_graph*, *machine_time_step*, *time_scale_factor*, *n_keys_map*, *application_graph=None*)

main entrance

Parameters

- **machine_graph** (*MachineGraph*) – machine graph.
- **machine_time_step** (*int*) – the machine time step.
- **time_scale_factor** (*int*) – the time scale factor.
- **n_keys_map** (*AbstractMachinePartitionNKeysMap*) – the map of partitions to n keys.

- **application_graph** (*ApplicationGraph or None*) – app graph.

config_values(*clocks_per_cycle*)
Read the config for the right parameters and combinations.

Parameters **clocks_per_cycle** (*int*) – The number of clock cycles per time step

Returns (app_machine_quantity, clocks_between_cores, clocks_for_sending, clocks_waiting, initial_clocks)

Return type tuple(int, int, int, int. int)

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

Discovers where applications of particular types need to be launched.

__call__(*graph*, *placements*)

Parameters

- **graph** (*MachineGraph*) –
- **placements** (*Placements*) –

Return type dict(*ExecutableType*,CoreSubsets or None)

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

On-machine bitfield-aware routing table compression.

BIT_FIELD_ADDRESSES_SDRAM_TAG = 2
sdram tag for the addresses the router compressor expects to find the bitfield addresses for the chip.

N_REGIONS_ELEMENT = 1
how many header elements are in the region addresses (1, n addresses)

ROUTING_TABLE_SDRAM_TAG = 1
sdram tag the router compressor expects to find there routing tables in

SUCCESS = 0
the successful identifier

TIMES_CYCLED_ROUTING_TABLES = 3

__call__(*routing_tables*, *transceiver*, *machine*, *app_id*, *provenance_file_path*, *machine_graph*, *placements*, *executable_finder*, *write_compressor_iobuf*, *produce_report*, *default_report_folder*, *target_length*, *routing_infos*, *time_to_try_for_each_iteration*, *use_timer_cut_off*, *machine_time_step*, *time_scale_factor*, *threshold_percentage*, *retry_count*, *executable_targets*, *compress_as_much_as_possible=False*, *provenance_data_objects=None*)
entrance for routing table compression with bit field

Parameters

- **routing_tables** (*MulticastRoutingTables*) – routing tables
- **transceiver** (*Transceiver*) – spinnman instance
- **machine** (*Machine*) – spinnMachine instance
- **app_id** (*int*) – app id of the application
- **provenance_file_path** (*str*) – file path for prov data
- **machine_graph** (*MachineGraph*) – machine graph
- **placements** (*Placements*) – placements on machine

- **executable_finder** (ExecutableFinder) – where are binaries are located
- **write_compressor_iobuf** (*bool*) – flag saying if read IOBUF
- **produce_report** (*bool*) –
- **default_report_folder** (*str*) –
- **target_length** (*int*) –
- **routing_infos** (*RoutingInfo*) –
- **time_to_try_for_each_iteration** (*int*) –
- **use_timer_cut_off** (*bool*) –
- **machine_time_step** (*int*) –
- **time_scale_factor** (*int*) –
- **threshold_percentage** (*int*) – the percentage of bitfields to do on chip before its considered a success
- **retry_count** (*int*) – Number of times that the sorters should set of the compressions again
- **executable_targets** (*ExecutableTargets*) – the set of targets and executables
- **compress_as_much_as_possible** (*bool*) – whether to compress as much as possible
- **provenance_data_objects** (*list(ProvenanceDataItem)*) –

Returns where the compressors ran, and the provenance they generated

Return type tuple(ExecutableTargets, list(*ProvenanceDataItem*))

compressor_aplx

Returns The name of the compressor aplx file to use

compressor_type

Returns The name of the compressor (excluding bitfields) being used

class spinn_front_end_common.interface.interface_functions.**MachineGenerator**

Bases: object

Makes a transceiver and a machine object.

POWER_CYCLE_FAILURE_WARNING = 'The end user requested the power-cycling of the board. I

POWER_CYCLE_WARNING = 'When power-cycling a board, it is recommended that you wait for

__call__(*hostname*, *bmp_details*, *downed_chips*, *downed_cores*, *downed_links*, *board_version*, *auto_detect_bmp*, *scamp_connection_data*, *boot_port_num*, *reset_machine_on_start_up*, *report_waiting_logs*, *max_sdram_size=None*, *repair_machine=False*, *ignore_bad_ethernets=True*, *default_report_directory=None*)

Parameters

- **hostname** (*str*) – the hostname or IP address of the SpiNNaker machine
- **bmp_details** (*str*) – the details of the BMP connections
- **downed_chips** (*set(tuple(int,int))*) – the chips that are down which SARK thinks are alive

- **downed_cores** (*set(tuple(int,int,int))*) – the cores that are down which SARK thinks are alive
- **downed_links** (*set(tuple(int,int,int))*) – the links that are down which SARK thinks are alive
- **board_version** (*int*) – the version of the boards being used within the machine (1, 2, 3, 4 or 5)
- **auto_detect_bmp** (*bool*) – Whether the BMP should be automatically determined
- **scamp_connection_data** (*list(SocketAddressWithChip)*) – the list of SC&MP connection data or None
- **boot_port_num** (*int*) – the port number used for the boot connection
- **reset_machine_on_start_up** (*bool*) –
- **max_sdram_size** (*int or None*) – the maximum SDRAM each chip can say it has (mainly used in debugging purposes)
- **repair_machine** (*bool*) – Flag to set the behaviour if a repairable error is found on the machine. If *True* will create a machine without the problematic bits. (See machine_factory.machine_repair) If *False*, get machine will raise an Exception if a problematic machine is discovered.
- **ignore_bad_ethernets** (*bool*) – Flag to say that ip_address information on non-ethernet chips should be ignored. Non-ethernet chips are defined here as ones that do not report themselves their nearest ethernet. The bad IP address is always logged. If True, the IP address is ignored. If False, the chip with the bad IP address is removed.
- **default_report_directory** (*str*) – Directory to write any reports too. If None the current directory will be used.
- **report_waiting_logs** (*bool*) – flag for the txrx to report when waiting on busy cores.

Returns Transceiver, and description of machine it is connected to

Return type tuple(Transceiver, Machine)

class spinn_front_end_common.interface.interface_functions.**PlacementsProvenanceGatherer**

Bases: object

Gets provenance information from vertices on the machine.

__call__(*transceiver*, *placements*)

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan interface object
- **placements** (*Placements*) – The placements of the vertices

Return type list(*ProvenanceDataItem*)

class spinn_front_end_common.interface.interface_functions.**PreAllocateForBitFieldRouterComp**

Bases: object

Preallocates resources required for bitfield router compression.

__call__(*machine*, *sdram_to_pre_alloc_for_bit_fields*, *pre_allocated_resources*)

Parameters

- **machine** (*Machine*) – the SpiNNaker machine as discovered

- **sdram_to_pre_alloc_for_bit_fields** (*int*) – SDRAM end user managed to help with bitfield compressions. Basically ensuring some SDRAM is available in the case where there is no SDRAM available to steal/use.
- **pre_allocated_resources** (*PreAllocatedResourceContainer*) – other preallocated resources

Returns preallocated resources

Return type PreAllocatedResourceContainer

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

Adds chip power monitor resources as required for a machine.

__call__(*machine*, *n_samples_per_recording*, *sampling_frequency*, *time_scale_factor*, *machine_time_step*, *pre_allocated_resources*)

Parameters

- **machine** (*Machine*) – the SpiNNaker machine as discovered
- **n_samples_per_recording** (*int*) – how many samples between record entries
- **sampling_frequency** (*int*) – the frequency of sampling
- **time_scale_factor** (*int*) – the time scale factor
- **machine_time_step** (*int*) – the machine time step
- **pre_allocated_resources** (*PreAllocatedResourceContainer*) – other preallocated resources

Returns preallocated resources

Return type PreAllocatedResourceContainer

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

Allocates resources for the extra monitors.

__call__(*machine*, *pre_allocated_resources*, *n_cores_to_allocate=1*)

Parameters

- **machine** (*Machine*) – SpiNNaker machine object
- **pre_allocated_resources** (*PreAllocatedResourceContainer*) – resources already preallocated
- **n_cores_to_allocate** (*int*) – how many gatherers to use per chip

Return type PreAllocatedResourceContainer

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

Adds Live Packet Gatherer resources as required for a machine.

__call__(*live_packet_gatherer_parameters*, *machine*, *pre_allocated_resources*)

Parameters

- **live_packet_gatherer_parameters** (*dict(LivePacketGatherParameters, list(tuple(AbstractVertex, list(str))))*) – the LPG parameters requested by the script

- **machine** (*Machine*) – the SpiNNaker machine as discovered
- **pre_allocated_resources** (*PreAllocatedResourceContainer*) – other preallocated resources

Returns preallocated resources

Return type PreAllocatedResourceContainer

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

Adds constraints to partitions if the vertices at either end of the partition request it.

__call__(*machine_graph*)

Parameters **machine_graph** (*MachineGraph*) –

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

Gets all the profiling data recorded by vertices and writes it to files.

__call__(*transceiver*, *placements*, *provenance_file_path*, *machine_time_step*)

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan interface object
- **placements** (*Placements*) – The placements of the vertices
- **provenance_file_path** (*str*) – The location to store the profile data
- **machine_time_step** (*int*) – machine time step in ms

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

Write provenance data into JSON

__call__(*provenance_data_items*, *provenance_data_path*)

Parameters

- **provenance_data_items** (*list(ProvenanceDataItem)*) –
- **provenance_data_path** (*str*) –

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

Writes provenance in SQL format.

__call__(*provenance_data_items*, *provenance_data_path*)
Writes provenance in SQL format

Parameters

- **provenance_data_items** (*list(ProvenanceDataItem)*) –
- **provenance_data_path** (*str*) –

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

Writes provenance in XML format.

__call__(*provenance_data_items*, *provenance_data_path*)

Parameters

- **provenance_data_items** (*list(ProvenanceDataItem)*) – data items for provenance
- **provenance_data_path** (*str*) – the file path to store provenance in

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

Reads compressed routing tables from a SpiNNaker machine.

__call__(*transceiver*, *routing_tables*, *app_id*)

Parameters

- **transceiver** (*Transceiver*) –
- **routing_tables** (*MulticastRoutingTables*) – uncompressed routing tables
- **app_id** (*int*) –

Return type MulticastRoutingTables

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

Gathers diagnostics from the routers.

__call__(*transceiver*, *machine*, *router_tables*, *using_reinjection*, *provenance_data_objects=None*, *extra_monitor_vertices=None*, *placements=None*)

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan interface object
- **machine** (*Machine*) – the SpiNNaker machine
- **router_tables** (*MulticastRoutingTables*) – the router tables that have been generated
- **using_reinjection** (*bool*) – whether we are reinjecting packets
- **provenance_data_objects** (*list(ProvenanceDataItem)*) – any existing provenance information to add to
- **extra_monitor_vertices** (*list(ExtraMonitorSupportMachineVertex)*) – vertices which represent the extra monitor code
- **placements** (*Placements*) – the placements object

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

Initialises the routers. Note that this does not load any routes into them.

__call__(*router_tables*, *app_id*, *transceiver*, *machine*)

Parameters

- **router_tables** (*MulticastRoutingTables*) –
- **app_id** (*int*) –
- **transceiver** (*Transceiver*) –
- **machine** (*Machine*) –

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

Loads routes into initialised routers.

__call__(*router_tables*, *app_id*, *transceiver*, *machine*)

Parameters

- **router_tables** (*MulticastRoutingTables*) –
- **app_id** (*int*) –
- **transceiver** (*Transceiver*) –
- **machine** (*Machine*) –

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

__call__(*machine_graph*, *transceiver*, *placements*, *app_id*)
Call self as a function.

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

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

__call__(*spalloc_server*, *spalloc_user*, *n_chips=None*, *n_boards=None*, *spalloc_port=None*, *spalloc_machine=None*)

Parameters

- **spalloc_server** (*str*) – The server from which the machine should be requested
- **spalloc_user** (*str*) – The user to allocate the machine to
- **n_chips** (*int or None*) – The number of chips required. IGNORED if n_boards is not None
- **n_boards** (*int or None*) – The number of boards required
- **spalloc_port** (*int*) – The optional port number to speak to spalloc
- **spalloc_machine** (*str*) – The optional spalloc machine to use

Return type tuple(str, int, None, bool, bool, None, None, *MachineAllocationController*)

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

Generates a maximum virtual machine a given allocation server can generate.

__call__(*spalloc_server*, *spalloc_port=22244*, *spalloc_machine=None*, *max_sdram_size=None*, *max_machine_core_reduction=0*)

Parameters

- **spalloc_server** (*str*) –
- **spalloc_port** (*int*) –
- **spalloc_machine** (*str*) –
- **max_sdram_size** (*int*) –
- **max_machine_core_reduction** (*int*) – the number of cores less than DEFAULT_MAX_CORES_PER_CHIP that each chip should have

Returns A virtual machine

Return type Machine

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

Loads tags onto the machine.

__call__(*transceiver*, *tags=None*, *iptags=None*, *reverse_iptags=None*)

Parameters

- **transceiver** (*Transceiver*) – the transceiver object
- **tags** (*Tags*) – the tags object which contains IP and reverse IP tags; could be None if these are being given in separate lists
- **iptags** (*list(IPTag)*) – a list of IP tags, given when tags is None
- **reverse_iptags** (*list(ReverseIPTag)*) – a list of reverse IP tags when tags is None

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

Parameters

- **iptags** (*list(IPTag)*) – the IP tags to be loaded.
- **transceiver** (*Transceiver*) – the transceiver object
- **progress_bar** (*ProgressBar*) –

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

Parameters

- **reverse_ip_tags** (*list(ReverseIPTag)*) – the reverse IP tags to be loaded
- **transceiver** (*Transceiver*) – the transceiver object
- **progress_bar** (*ProgressBar*) –

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

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

__call__(*machine_graph*, *number_of_cpu_cycles_per_receive*, *other_cpu_demands_in_cpu_cycles*, *n_packets_per_time_window*, *machine_time_step*, *time_scale_factor*, *safety_factor=1*)

Parameters

- **machine_graph** (*MachineGraph*) – the graph for which we are planning an agenda
- **number_of_cpu_cycles_per_receive** (*int*) – how long the packet reception callback takes in CPU cycles
- **other_cpu_demands_in_cpu_cycles** (*int*) – extra costs (e.g. timer tick callback etc.)
- **n_packets_per_time_window** (*int*) – how many packets are to be sent per time window

- **machine_time_step** (*int*) – the timer tick in microseconds
- **time_scale_factor** (*int*) – the multiplicative factor on the machine time step.
- **safety_factor** (*float*) – the end user safety factor

Returns the agenda for each vertex on its time window and its offset between spike transmissions

Return type dict(MachineVertex,dict(str,int))

Raises ***ConfigurationException*** – If colouring fails

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

Bases: object

Generates a virtual machine with given dimensions and configuration.

__call__(*width=None*, *height=None*, *version=None*, *down_chips=None*, *down_cores=None*, *down_links=None*, *max_sdram_size=None*, *router_entries_per_chip=1024*, *json_path=None*)

Parameters

- **width** (*int*) – The width of the machine in chips
- **height** (*int*) – The height of the machine in chips
- **version** (*int*) – The version of board to create
- **down_chips** (*list(tuple(int,int))*) – The set of chips that should be considered broken
- **down_cores** (*list(tuple(int,int,int))*) – The set of cores that should be considered broken
- **down_links** (*list(tuple(int,int,int))*) – The set of links that should be considered broken
- **max_sdram_size** (*int*) – The SDRAM that should be given to each chip
- **router_entries_per_chip** (*int*) – The number of router entries to allocate.
- **json_path** (*str*) – Where to load a JSON description of the machine from, if anywhere.

Returns The virtual machine.

Return type Machine

Raises **Exception** – If given bad arguments

spinn_front_end_common.interface.profiling package

Module contents

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

Bases: object

Indicates a MachineVertex that can record a profile.

get_profile_data(*transceiver*, *placement*)

Get the profile data recorded during simulation

Parameters

- **transceiver** (*Transceiver*) –
- **placement** (*Placement*) –

Return type *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*, *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** (*float*) – The time step of the simulation in microseconds

Return type float

get_mean_n_calls_per_ts(*tag*, *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** (*float*) – The time step of the simulation in microseconds

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

Module contents

class spinn_front_end_common.interface.provenance.**AbstractProvidesLocalProvenanceData**
Bases: object

Indicates an object that provides locally obtained provenance data.

GraphProvenanceGatherer will check all Vertices and all Edges in both the MachineGraph and if it exists the ApplicationGraph

get_local_provenance_data()
Get an iterable of provenance data items.

Returns the provenance items

Return type iterable(*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** (*Transceiver*) – the SpinnMan interface object
- **placement** (*Placement*) – the placement of the object

Returns the provenance items

Return type iterable(*ProvenanceDataItem*)

class spinn_front_end_common.interface.provenance.**PacmanProvenanceExtractor**
Bases: object

Extracts Provenance data from a PACMANAlgorithmExecutor

TOP_NAME = 'pacman'

clear()
Clears the provenance data store

Return type None

data_items
Returns the provenance data items

Returns the provenance items

Return type iterable(*ProvenanceDataItem*)

extract_provenance(*executor*)
Acquires the timings from PACMAN algorithms (provenance data)

Parameters **executor** (*PACMANAlgorithmExecutor*) – the PACMAN workflow executor

Return type None

class spinn_front_end_common.interface.provenance.**ProvenanceReader**(*provenance_data_path=None*)
Bases: object

Provides a connection to a database containing provenance for the current run and some convenience methods for extracting provenance data from it.

By default this will wrap around the database used to store the provenance of the last run. The path is not updated so this reader is not effected by a reset or an end.

The assumption is that the database is in the current provenance format. This includes both that DDL statements used to create the database but also the underlying database structure (currently sqlite3)

Warning: This class is only a wrapper around the database file so if the file is deleted the class will no longer work.

Create a wrapper around the database.

The suggested way to call this is *without* the `provenance_data_path` parameter, allowing *`get_last_run_database_path()`* to find the correct path.

Parameters **provenance_data_path** (*`None or str`*) – Path to the provenance database to wrap

cores_with_late_spikes()
Gets the x, y, p and count of the cores where late spikes arrived.

Cores that received spikes but where none were late are *not* included.

Returns A list hopefully empty of tuples (x, y, p, count) of cores where their where late arriving spikes.

Return type list(tuple(int, int, int, int))

get_database_handle(*read_only=True*, *use_sqlite_rows=False*)
Gets a handle to the open database.

You *should* use this as a Python context handler. A typical usage pattern is this:

```
with reader.get_database_handler() as db:
    with db.transaction() as cursor:
        for row in cursor.execute(...):
            # process row
```

Note: This method is mainly provided as a support method for the later methods that return specific data. For new IntergationTests please add a specific method rather than call this directly.

Warning: It is the callers responsibility to close the database. The recommended usage is therefore a `with` statement

Parameters

- **read_only** (*`bool`*) – If true will return a readonly database
- **use_sqlite_rows** (*`bool`*) – If `True` the results of *`run_query()`* will be `Row`s. If `False` the results of *`run_query()`* will be `tuple`s.

Returns an open sqlite3 connection

Return type *SQLiteDB*

static get_last_run_database_path()
Get the path of the current provenance database of the last run

> **Warning:**
>
> Calling this method between start/reset and run may result in a path to a database not yet created.
>
> **raises ValueError** if the system is in a state where path can't be retrieved, for example before run is called

get_provenance(*description_name*)
Gets the provenance item(s) from the last run

Parameters **description_name** (*str*) – The value to LIKE search for in the description_name column. Can be the full name, or have % and _ wildcards.

Returns A possibly multiline string with for each row which matches the like a line `description_name: value`

Return type str

get_provenance_for_chip(*x*, *y*)
Gets the provenance item(s) from the last run relating to a chip

Parameters

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

Returns A possibly multiline string with for each row which matches the like a line `description_name: value`

Return type str

get_run_time_of_BufferExtractor()
Gets the BufferExtractor provenance item(s) from the last run

Returns A possibly multiline string with for each row which matches the like %BufferExtractor description_name: value

Return type str

get_run_times()
Gets the algorithm running times from the last run. If an algorithm is invoked multiple times in the run, its times are summed.

Returns A possibly multiline string with for each row which matches the like a line `description_name: time`. The times are in seconds.

Return type str

run_query(*query*, *params=()*, *read_only=True*, *use_sqlite_rows=False*)
Opens a connection to the database, runs a query, extracts the results and closes the connection

The return type depends on the use_sqlite_rows param. By default this method returns tuples (lookup by index) but the advanced tuple type can be used instead, which supports lookup by name used in the query (use `AS name` in the query to set).

This method will not allow queries that change the database unless the read_only flag is set to False.

Note: This method is mainly provided as a support method for the later methods that return specific data. For new IntergationTests please add a specific method rather than call this directly.

Parameters

- **query** (*str*) – The SQL query to be run. May include ? wildcards
- **or int) params** (*Iterable(str*) – The values to replace the ? wildcards with. The number and types must match what the query expects
- **read_only** (*bool*) – see *get_database_handle()*
- **use_sqlite_rows** (*bool*) – see *get_database_handle()*

Returns A list possibly empty of tuples/rows (one for each row in the database) where the number and type of the values corresponds to the where statement

Return type list(tuple or Row)

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

Entries for the provenance data generated by models using provides provenance vertex.

CALLBACK_QUEUE_OVERLOADED = 1
The counter of the number of times the callback queue was overloaded

DMA_QUEUE_OVERLOADED = 2
The counter of the number of times the DMA queue was overloaded

MAX_NUMBER_OF_TIMER_TIC_OVERRUN = 4
The counter of the number of times the timer tick overran

TIMER_TIC_HAS_OVERRUN = 3
Whether the timer tick has overrun at all at any point

TRANSMISSION_EVENT_OVERFLOW = 0
The counter of transmission overflows

get_provenance_data_from_machine(*transceiver*, *placement*)
Retrieve the provenance data.

Parameters

- **transceiver** (*Transceiver*) – How to talk to the machine
- **placement** (*Placement*) – Which vertex are we retrieving from, and where was it

Return type list(*ProvenanceDataItem*)

classmethod get_provenance_data_size(*n_additional_data_items*)

Parameters **n_additional_data_items** (*int*) –

Return type int

reserve_provenance_data_region(*spec*)

Parameters **spec** (*DataSpecificationGenerator*) – The data specification being written.

class spinn_front_end_common.interface.provenance.**SqlLiteDatabase**(*database_file=None*)
Bases: *spinn_front_end_common.utilities.sqlite_db.SQLiteDB*

Specific implementation of the Database for SQLite 3.

Note: *Not thread safe on the same database file.* Threads can access different DBs just fine.

Note: This totally relies on the way SQLite's type affinities function. You can't port to a different database engine without a lot of work.

Parameters **database_file** (*str*) – The name of a file that contains (or will contain) an SQLite database holding the data. If omitted, an unshared in-memory database will be used (suitable only for testing).

insert_items(*items*)
Does a bulk insert of the items into the database.

Parameters **items** (*list(ProvenanceDataItem)*) – The items to insert

spinn_front_end_common.interface.simulation package

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** (*str*) – The name of the binary of the application
- **machine_time_step** (*int*) – The time step of the simulation
- **time_scale_factor** (*int*) – The time scaling of the simulation

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

Return type list(int)

spinn_front_end_common.interface.splitter_selectors package

Module contents

class spinn_front_end_common.interface.splitter_selectors.**SplitterSelector**
Bases: object

Splitter object selector that allocates nothing but legacy splitter objects where required

NOT_KNOWN_APP_VERTEX_ERROR_MESSAGE = 'The SplitterSelector has not seen the {} vertex

__call__(*app_graph*)
basic selector which puts the legacy splitter object on everything without a splitter object

Parameters **app_graph** (*ApplicationGraph*) – app graph

Return type None

vertex_selector(*app_vertex*)
main point for selecting a splitter object for a given app vertex.

Will assume the SplitterSliceLegacy if no heuristic is known for the app vertex.

Parameters **app_vertex** (*ApplicationVertex*) – app vertex to give a splitter object to

Return type None

Submodules

spinn_front_end_common.interface.abstract_spinnaker_base module

main interface for the SpiNNaker tools

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

Bases: *spinn_front_end_common.interface.config_handler.ConfigHandler*, spinn_front_end_common.utilities.simulator_interface.SimulatorInterface

Main interface into the tools logic flow.

Parameters

- **configfile** (*str*) – What the configuration file is called
- **executable_finder** (*ExecutableFinder*) – How to find APLX files to deploy to SpiNNaker
- **graph_label** (*str*) – A label for the overall application graph
- **database_socket_addresses** (*iterable(SocketAddress) or None*) – How to talk to notification databases

- **extra_algorithm_xml_paths** (*iterable(str)*) – Where to load definitions of extra algorithms from
- **n_chips_required** (*int*) – Overrides the number of chips to allocate from spalloc
- **n_boards_required** (*int*) – Overrides the number of boards to allocate from spalloc
- **default_config_paths** (*list(str)*) – Directories to load configurations from
- **validation_cfg** (*str*) – How to validate configuration files
- **front_end_versions** (*list(tuple(str,str))*) – Information about what software is in use

add_application_edge(*edge_to_add*, *partition_identifier*)

Parameters

- **edge_to_add** (*ApplicationEdge*) – the edge to add to the graph
- **partition_identifier** (*str*) – the partition identifier for the outgoing edge partition

add_application_vertex(*vertex*)

Parameters **vertex** (*ApplicationVertex*) – the vertex to add to the graph

Raises

- ***ConfigurationException*** – when both graphs contain vertices
- **PacmanConfigurationException** – If there is an attempt to add the same vertex more than once

add_extraction_timing(*timing*)

Record the time taken for doing data extraction.

Parameters **timing** (*timedelta*) –

add_live_packet_gatherer_parameters(*live_packet_gatherer_params*, *vertex_to_record_from*, *partition_ids*)

Adds parameters for a new LPG if needed, or adds to the tracker for parameters. Note that LPGs can be inserted to track behaviour either at the application graph level or at the machine graph level, but not both at the same time.

Parameters

- **live_packet_gatherer_params** (LivePacketGatherParameters) – params to look for a LPG
- **vertex_to_record_from** (*AbstractVertex*) – the vertex that needs to send to a given LPG
- **partition_ids** (*list(str)*) – the IDs of the partitions to connect from the vertex

add_machine_edge(*edge*, *partition_id*)

Parameters

- **edge** (*MachineEdge*) – the edge to add to the graph
- **partition_id** (*str*) – the partition identifier for the outgoing edge partition

add_machine_vertex(*vertex*)

Parameters **vertex** (*MachineVertex*) – the vertex to add to the graph

Raises

- ***ConfigurationException*** – when both graphs contain vertices
- **PacmanConfigurationException** – If there is an attempt to add the same vertex more than once

add_socket_address(*socket_address*)
Add the address of a socket used in the run notification protocol.

Parameters **socket_address** (*SocketAddress*) – The address of the database socket

application_graph

The frozen clone of the application graph used to derive the runtime machine configuration.

Return type ApplicationGraph

buffer_manager
The buffer manager being used for loading/extracting buffers

Return type *BufferManager*

continue_simulation()
Continue a simulation that has been started in stepped mode

dsg_algorithm
The DSG algorithm used by the tools

Return type str

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

Parameters

- **exctype** (*type*) – the type of exception received
- **value** (*Exception*) – the value of the exception
- **traceback_obj** (*traceback*) – the trace back stuff

extend_extra_load_algorithms(*extra_load_algorithms*)
Add custom data-loading algorithms to the sequence of such algorithms to be run.

Parameters **extra_load_algorithms** (*list(str)*) – Algorithms to add

extend_extra_mapping_algorithms(*extra_mapping_algorithms*)
Add custom mapping algorithms to the end of the sequence of mapping algorithms to be run.

Parameters **extra_mapping_algorithms** (*list(str)*) – Algorithms to add

extend_extra_post_run_algorithms(*extra_post_run_algorithms*)
Add custom post-execution algorithms to the sequence of such algorithms to be run.

Parameters **extra_post_run_algorithms** (*list(str)*) – Algorithms to add

fixed_routes

Return type dict(tuple(int,int),FixedRouteEntry)

get_current_time()
Get the current simulation time.

Return type float

get_generated_output(*name_of_variable*)
Get the value of an inter-algorithm variable.

Parameters **name_of_variable** (*str*) – The variable to retrieve

Returns The value (of arbitrary type), or *None* if the variable is not found.

Raises ***ConfigurationException*** – If the simulation hasn't yet run

get_number_of_available_cores_on_machine
The number of available cores on the machine after taking into account preallocated resources.

Returns number of available cores

Return type int

has_ran
Whether the simulation has executed anything at all.

Return type bool

has_reset_last

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

machine
The python machine description object.

Return type Machine

machine_graph
Returns a frozen clone of the machine_graph :rtype: ~pacman.model.graphs.machine.MachineGraph

no_machine_time_steps
The number of machine time steps.

Return type int

none_labelled_edge_count
The number of times edges have not been labelled.

Return type int

original_application_graph

Return type ApplicationGraph

original_machine_graph

Return type MachineGraph

placements
Where machine vertices are placed on the machine.

Return type Placements

prepend_extra_pre_run_algorithms(*extra_pre_run_algorithms*)
Add custom pre-execution algorithms to the front of the sequence of algorithms to be run.

Parameters **extra_pre_run_algorithms** (*list(str)*) – Algorithms to add

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

routing_infos

Return type RoutingInfo

run(*run_time*, *sync_time=0*)
Run a simulation for a fixed amount of time

Parameters

- **run_time** (*int*) – the run duration in milliseconds.
- **sync_time** (*float*) – If not 0, this specifies that the simulation should pause after this duration. The continue_simulation() method must then be called for the simulation to continue.

run_until_complete(*n_steps=None*)
Run a simulation until it completes

Parameters **n_steps** (*int*) – If not None, this specifies that the simulation should be requested to run for the given number of steps. The host will still wait until the simulation itself says it has completed

set_n_boards_required(*n_boards_required*)
Sets the machine requirements.

> **Warning:** This method should not be called after the machine requirements have be computed based on the graph.

Parameters **n_boards_required** (*int*) – The number of boards required

Raises ConfigurationException If any machine requirements have already been set

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

Parameters **hostname** (*str*) – machine name

stop(*turn_off_machine=None*, *clear_routing_tables=None*, *clear_tags=None*)
End running of the simulation.

Parameters

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

stop_run()
Request that the current infinite run stop.

> **Note:** This will need to be called from another thread as the infinite run call is blocking.

tags

Return type Tags

transceiver
How to talk to the machine.

Return type Transceiver

update_extra_inputs(*extra_inputs*)
Supply extra inputs to the runtime algorithms. Mappings are from known names (the logical type names) to the values to bind to them.

Parameters **extra_inputs** (*dict(str,any)*) – The additional inputs to provide

update_extra_mapping_inputs(*extra_mapping_inputs*)
Supply extra inputs to the mapping algorithms. Mappings are from known names (the logical type names) to the values to bind to them.

Parameters **extra_inputs** (*dict(str,any)*) – The additional inputs to provide

use_virtual_board

Return type bool True if this run is using a virtual machine

Return type bool

verify_not_running()
Verify that the simulator is in a state where it can start running.

spinn_front_end_common.interface.abstract_spinnaker_base.**DEFAULT_N_VIRTUAL_CORES = 16**
Number of cores to be used when using a Virtual Machine and not specified

spinn_front_end_common.interface.abstract_spinnaker_base.**MINIMUM_OFF_STATE_TIME = 20**
The minimum time a board is kept in the off state, in seconds

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

Superclass of AbstractSpinnakerBase that handles function only dependent of the config and the order its methods are called.

Parameters

- **configfile** (*str*) – The base name of the configuration file(s). Should not include any path components.
- **default_config_paths** (*list(str) or None*) – The list of files to get default configurations from.
- **validation_cfg** (*list(str) or None*) – The list of files to read a validation configuration from. If None, no such validation is performed.

child_folder(*parent*, *child_name*, *must_create=False*)

Parameters

- **parent** (*str*) –
- **child_name** (*str*) –
- **must_create** (*bool*) – If *True*, the directory named by *child_name* (but not necessarily its parents) must be created by this call, and an exception will be thrown if this fails.

Returns The fully qualified name of the child folder.

Return type str

Raises `OSError` – if the directory existed ahead of time and creation was required by the user

config

machine_time_step
The machine timestep, in microseconds

Return type int

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

Parameters

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

time_scale_factor
The time scaling factor.

Return type int

write_finished_file()
Write a finished file that allows file removal to only remove folders that are finished.

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** (*str*) – 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`

Raises *ConfigurationException* – if simple parameter checking fails.

execute_app_data_specification(*use_monitors*)

Writes all the data specs for application cores, uploading the result to the machine.

Parameters **use_monitors** (*bool*) –

Raises **PacmanExternalAlgorithmFailedToCompleteException** – On failure of the Java code.

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

Raises **PacmanExternalAlgorithmFailedToCompleteException** – On failure of the Java code.

execute_system_data_specification()

Writes all the data specs for system cores, uploading the result to the machine.

Raises **PacmanExternalAlgorithmFailedToCompleteException** – On failure of the Java code.

get_all_data()

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

Raises **PacmanExternalAlgorithmFailedToCompleteException** – On failure of the Java code.

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

Parameters

- **placements** (*Placements*) – The placements of the vertices
- **tags** (*Tags*) – The tags assigned to the vertices
- **monitor_cores** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) – Where the advanced monitor for each core is
- **packet_gathers** (*dict(tuple(int,int), DataSpeedUpPacketGatherMachineVertex)*) – Where the packet gatherers are

Return type None

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

Parameters **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** (*Placements*) – The Placements Object
- **transceiver** (*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.

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
run ended shutdown not called

INIT = 0
init called

IN_RUN = 1
inside run method

RUN_FOREVER = 2
finished run method, but running forever

SHUTDOWN = 4
shutdown called

STOP_REQUESTED = 5
stop requested in middle of run forever

Module contents

1.1.1.4 spinn_front_end_common.utilities package

Subpackages

spinn_front_end_common.utilities.connections package

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** (*str*) – 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** (*callable(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** (*callable(str, LiveEventConnection) -> 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*, *translate_key=True*)
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** (*callable(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.
- **translate_key** (*bool*) – True if the key is to be converted to an atom ID, False if the key should stay a key

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** (*callable(str, LiveEventConnection) -> 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*)
Add a callback for the start and resume state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **start_resume_callback** (*callable(str, LiveEventConnection) -> None*) – A function to be called when the start or resume 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

close()
Closes the connection

send_eieio_message(*message*, *label*)
Send an EIEIO message (using one-way the live input) to the vertex with the given label.

Parameters

- **message** (*AbstractEIEIOMessage*) – The EIEIO message to send
- **label** (*str*) – The label of the receiver machine vertex

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_event_with_payload(*label*, *atom_id*, *payload*)
Send an event with a payload 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
- **payload** (*int*) – The payload to send

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

send_events_with_payloads(*label*, *atom_ids_and_payloads*)
Send a number of events with payloads

Parameters

- **label** (*str*) – The label of the vertex from which the events will originate
- **atom_ids_and_payloads** (*list(tuple(int,int))*) – array-like of tuples of atom IDs sending events with their payloads

spinn_front_end_common.utilities.database package

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.

Note: The machine description database reader can only be used while the registered database callbacks are running.

Parameters

- **start_resume_callback_function** (*callable*) – 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** (*callable(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 will not be needed further, and the return response will be sent.

close()
Closes the connection

class spinn_front_end_common.utilities.database.**DatabaseReader**(*database_path*)
Bases: *spinn_front_end_common.utilities.sqlite_db.SQLiteDB*

A reader for the database.

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

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 or None

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

Parameters

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

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

Return type str or None

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, board address, tag)

Return type tuple(str, int, bool, str, int)

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*)

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

Return type tuple(int,int)

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
- **receiver_label** (*str*) –

Returns tuple of (IP address, port, strip SDP, board address, tag)

Return type tuple(str, int, bool, str, int)

get_machine_live_output_key(*label*, *receiver_label*)

Parameters

- **label** (*str*) – The label of the vertex
- **receiver_label** (*str*) –

Return type tuple(int,int)

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** (*str*) – The label of the vertex

Returns A list of x, y, p coordinates of the vertices

Return type list(tuple(int, int, int))

class spinn_front_end_common.utilities.database.**DatabaseWriter**(*database_directory*)
Bases: *spinn_front_end_common.utilities.sqlite_db.SQLiteDB*

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.

Parameters **database_directory** (*str*) – Where the database will be written

add_application_vertices(*application_graph*)
Stores the main application graph description (vertices, edges).

Parameters **application_graph** (*ApplicationGraph*) – The graph to add from

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

Parameters **machine** (*Machine*) – the machine object.

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

Parameters **placements** (*Placements*) – the placements object

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

Parameters

- **routing_infos** (*RoutingInfo*) – the routing information object
- **machine_graph** (*MachineGraph*) – the machine graph object

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

Parameters **routing_tables** (*MulticastRoutingTables*) – the routing tables object

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

Parameters

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

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

Parameters

- **machine_graph** (*MachineGraph*) – the machine graph object
- **tags** (*Tags*) – the tags object

add_vertices(*machine_graph*, *data_n_timesteps*, *application_graph*)
Add the machine graph into the database.

Parameters

- **machine_graph** (*MachineGraph*) – The machine graph object
- **data_n_timesteps** (*int*) – The number of timesteps for which data space will been reserved
- **application_graph** (*ApplicationGraph*) – The application graph object

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

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

Returns whether the database is needed for the application

Return type bool

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

Parameters

- **application_graph** (*ApplicationGraph*) –
- **machine_graph** (*MachineGraph*) –
- **routing_infos** (*RoutingInfo*) –

database_path

Return type str

spinn_front_end_common.utilities.notification_protocol package

Module contents

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

Bases: spinn_utilities.abstract_context_manager.AbstractContextManager

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

Parameters

- **socket_addresses** (*list(SocketAddress)*) – Where to notify.
- **wait_for_read_confirmation** (*bool*) – Whether to wait for the other side to acknowledge

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** (*str*) – the path to the database file

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

sent_visualisation_confirmation
Whether the external application has actually been notified yet.

Return type bool

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.report_functions package

Submodules

spinn_front_end_common.utilities.report_functions.energy_report module

class spinn_front_end_common.utilities.report_functions.energy_report.**EnergyReport**(*report_default_directory*, *version*, *spalloc_server*, *remote_spinnaker_url*, *time_scale_factor*)

Bases: object

This class creates a report about the approximate total energy consumed by a SpiNNaker job execution.

Parameters

- **report_default_directory** (*str*) – location for reports
- **version** (*int*) – version of machine
- **spalloc_server** (*str*) – spalloc server IP
- **remote_spinnaker_url** (*str*) – remote SpiNNaker URL
- **time_scale_factor** (*int*) – the time scale factor

JOULES_TO_KILOWATT_HOURS = 3600000
converter between joules to kilowatt hours

write_energy_report(*placements*, *machine*, *runtime*, *buffer_manager*, *power_used*)
Writes the report.

Parameters

- **placements** (*Placements*) – the placements
- **machine** (*Machine*) – the machine
- **runtime** (*int*) –
- **buffer_manager** (BufferManager) –
- **power_used** (PowerUsed) –

Return type None

Module contents

Much of the code in this module is intended primarily for being invoked via the PACMAN Executor.

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

Generates a report that shows the impact of the compression of bitfields into the routing table.

__call__(*report_default_directory*, *machine_graph*, *placements*, *provenance_items=None*)

Parameters

- **report_default_directory** (*str*) – report folder
- **machine_graph** (*MachineGraph*) – the machine graph
- **placements** (*Placements*) – the placements
- **provenance_items** (*list(ProvenanceDataItem) or None*) – prov items

Returns a summary, or *None* if the report file can't be written

Return type *BitFieldSummary*

class spinn_front_end_common.utilities.report_functions.**BitFieldSummary**(*total_merged*, *max_per_chip*, *lowest_per_chip*, *total_to_merge*, *max_to_merge_per_chip*, *low_to_merge_per_chip*, *average_per_chip_merged*, *average_per_chip_to_merge*)

Bases: object

Summary description of generated bitfields.

Parameters

- **total_merged** (*int*) –
- **max_per_chip** (*int*) –
- **lowest_per_chip** (*int*) –
- **total_to_merge** (*int*) –
- **max_to_merge_per_chip** (*int*) –
- **low_to_merge_per_chip** (*int*) –
- **average_per_chip_merged** (*float*) –
- **average_per_chip_to_merge** (*float*) –

average_per_chip_merged

Return type float

average_per_chip_to_merge

Return type float

low_to_merge_per_chip

Return type int

lowest_per_chip

Return type int

max_per_chip

Return type int

max_to_merge_per_chip

Return type int

total_merged

Return type int

total_to_merge

Return type int

class spinn_front_end_common.utilities.report_functions.**BoardChipReport**

Bases: object

Report on memory usage

AREA_CODE_REPORT_NAME = 'board_chip_report.txt'

__call__(*report_default_directory*, *machine*)

Creates a report that states where in SDRAM each region is.

Parameters

- **report_default_directory** (*str*) – the folder where reports are written
- **machine** (*Machine*) – python representation of the machine

Return type None

class spinn_front_end_common.utilities.report_functions.**EnergyReport**(*report_default_directory*, *version*, *spalloc_server*, *remote_spinnaker_url*, *time_scale_factor*)

Bases: object

This class creates a report about the approximate total energy consumed by a SpiNNaker job execution.

Parameters

- **report_default_directory** (*str*) – location for reports
- **version** (*int*) – version of machine
- **spalloc_server** (*str*) – spalloc server IP
- **remote_spinnaker_url** (*str*) – remote SpiNNaker URL
- **time_scale_factor** (*int*) – the time scale factor

JOULES_TO_KILOWATT_HOURS = 3600000

converter between joules to kilowatt hours

write_energy_report(*placements*, *machine*, *runtime*, *buffer_manager*, *power_used*)

Writes the report.

Parameters

- **placements** (*Placements*) – the placements
- **machine** (*Machine*) – the machine
- **runtime** (*int*) –

- **buffer_manager** (BufferManager) –
- **power_used** (PowerUsed) –

Return type None

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

Generate a report of the fixed routes from the machine.

__call__(*transceiver*, *machine*, *report_default_directory*, *app_id*)
Writes the fixed routes from the machine

Parameters

- **transceiver** (*Transceiver*) – spinnMan instance
- **machine** (*Machine*) – SpiNNMachine instance
- **report_default_directory** (*str*) – folder where reports reside
- **app_id** (*int*) – the application ID the fixed routes were loaded with

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

Report on memory usage. Creates a report that states where in SDRAM each region is (read from machine)

__call__(*report_default_directory*, *dsg_targets*, *transceiver*)

Parameters

- **report_default_directory** (*str*) – the folder where reports are written
- **dsg_targets** (*dict(tuple(int,int,int),..)*) – the map between placement and file writer
- **transceiver** (*Transceiver*) – the spinnMan instance

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

Report on memory usage.

__call__(*report_default_directory*, *processor_to_app_data_base_address*)

Parameters

- **report_default_directory** (*str*) –
- **processor_to_app_data_base_address** (*dict(tuple(int,int,int), DataWritten)*) –

spinn_front_end_common.utilities.report_functions.**report_xml**()

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

Report the routing table that was actually on the machine.

__call__(*report_default_directory*, *routing_tables*)

Parameters

- **report_default_directory** (*str*) –
- **routing_tables** (*MulticastRoutingTables*) –

- **transceiver** (*Transceiver*) –
- **app_id** (*int*) –

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

Describes what the tags actually present on the machine are.

__call__(*report_default_directory*, *transceiver*)

Parameters

- **report_default_directory** (*str*) –
- **transceiver** (*Transceiver*) –

spinn_front_end_common.utilities.scp package

Module contents

class spinn_front_end_common.utilities.scp.**ClearIOBUFProcess**(*connection_selector*)
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to clear the IOBUF buffers of a set of cores.

Note: The cores must be using the simulation interface.

Parameters **connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

clear_iobuf(*core_subsets*, *n_cores=None*)

Parameters

- **core_subsets** (*CoreSubsets*) –
- **n_cores** (*int*) – Defaults to the number of cores in *core_subsets*.

class spinn_front_end_common.utilities.scp.**UpdateRuntimeProcess**(*connection_selector*)
Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to update the target running time of a set of cores.

Note: The cores must be using the simulation interface.

Parameters **connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

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

Parameters

- **current_time** (*int*) –
- **run_time** (*int*) –

- **infinite_run** (*bool*) –
- **core_subsets** (*CoreSubsets*) –
- **n_cores** (*int*) – Number of cores being updated

spinn_front_end_common.utilities.utility_objs package

Subpackages

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages package

Module contents

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.**ClearReinjec**

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 AbstractSCPResponse

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 AbstractSCPResponse

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*)

Reads the remainder of the data following the header

Parameters

- **data** (*bytes*) – The bytestring to read from
- **offset** (*int*) – The offset into the data after the headers

reinjection_functionality_status

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 AbstractSCPResponse

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.**LoadSystemMC**

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

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 AbstractSCPResponse

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.**ResetCounter**

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 AbstractSCPResponse

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 AbstractSCPResponse

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
- **wait** (*int*) – Which wait to set. Should be 1 or 2.

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 AbstractSCPResponse

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes package

Module contents

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ClearQueue**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to clear the reinjection queue.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

reset_counters(*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) –

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

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to load the saved application multicast routing tables.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

load_application_mc_routes(*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) – sets of cores to send command to

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

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to load the configured system multicast routing tables (and save the application routing tables).

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

load_system_mc_routes(*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) – sets of cores to send command to

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ReadStatusM**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to read the status of extra monitors.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

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

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **p** (*int*) –

Return type *ReInjectionStatus*

get_reinjection_status_for_core_subsets (*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) –

Return type dict(tuple(int,int), *ReInjectionStatus*)

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**ResetCounte**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to clear the reinjection state counters.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

reset_counters (*core_subsets*)

Parameters **core_subsets** (*CoreSubsets*) –

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**SetPacketTy**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to control what messages are reinjected.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

set_packet_types (*core_subsets*, *point_to_point*, *multicast*, *nearest_neighbour*, *fixed_route*)

Set what types of packets should be reinjected.

Parameters

- **core_subsets** (*CoreSubsets*) – 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

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.**SetRouterT**

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

How to send messages to set router timeouts. These messages need to be sent to cores running the extra monitor binary.

Note: SCAMP sets wait2 to zero by default!

Note: Timeouts are specified in a weird floating point format. See the SpiNNaker datasheet for details.

Parameters **next_connection_selector** (*AbstractMultiConnectionProcessConnectionSelector*) –

set_wait1_timeout(*mantissa*, *exponent*, *core_subsets*)

The wait1 timeout is the time from when a packet is received to when emergency routing becomes enabled.

Parameters

- **mantissa** (*int*) – Timeout mantissa (0 to 15)
- **exponent** (*int*) – Timeout exponent (0 to 15)
- **core_subsets** (*CoreSubsets*) – Where the extra monitors that manage the routers are located.

set_wait2_timeout(*mantissa*, *exponent*, *core_subsets*)

The wait2 timeout is the time from when a packet has emergency routing enabled for it to when it is dropped.

Parameters

- **mantissa** (*int*) – Timeout mantissa (0 to 15)
- **exponent** (*int*) – Timeout exponent (0 to 15)
- **core_subsets** (*CoreSubsets*) – Where the extra monitors that manage the routers are located.

Module contents

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

Bases: object

Describes data written to SpiNNaker.

Parameters

- **start_address** (*int*) –
- **memory_used** (*int*) –
- **memory_written** (*int*) –

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(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
Situation where there user has supplied no application but for some reason still wants to run.

RUNNING = 0
Runs immediately without waiting for barrier and then exits.

SYNC = 1
Calls `spin1_start(SYNC_WAIT)` and then eventually `spin1_exit()`.

SYSTEM = 4
Runs immediately without waiting for barrier and never ends.

USES_SIMULATION_INTERFACE = 2
Calls `simulation_run()` and `simulation_exit()` / `simulation_handle_pause_resume()`.

class spinn_front_end_common.utilities.utility_objs.**LivePacketGatherParameters**(*port=None*, *hostname=None*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIE 2>*, *right_shift=0*, *payload_as_time_sta use_payload_pre payload_prefix=Non payload_right_shift= number_of_packets_s label=None*, *board_address=*

Bases: `object`

Parameter holder for LPGs so that they can be instantiated at a later date.

Raises ***ConfigurationException*** – If the parameters passed are known to be an invalid combination.

board_address

get_iptag_resource()
Get a description of the IPtag that the LPG for these parameters will require.

Return type IPtagResource

hostname

key_prefix

label

message_type

number_of_packets_sent_per_time_step

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.**PowerUsed**
Bases: object

Describes the power used by a simulation.

active_cores
Enumeration of the coordinates of the cores that can report active energy usage.

Return type iterable(tuple(int, int, int))

active_routers
Enumeration of the coordinates of the routers that can report active energy usage.

Return type iterable(tuple(int, int))

add_core_active_energy(*x*, *y*, *p*, *joules*)
Adds energy for a particular core. It can be called multiple times per core.

Only intended to be used during construction of this object.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **p** (*int*) –
- **joules** (*float*) – the energy to add for this core, in Joules.

add_router_active_energy(*x*, *y*, *joules*)
Adds energy for a particular router. It can be called multiple times per router.

Only intended to be used during construction of this object.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **joules** (*float*) – the energy to add for this router, in Joules.

baseline_joules
Baseline/idle energy used, in Joules. This is used by things like the frames the SpiNNaker boards are held in, the cooling system, etc.

Return type float

booted_time_secs
Time taken when the machine is booted, in seconds.

Return type float

chip_energy_joules
Energy used by all SpiNNaker chips during active simulation running, in Joules.

Return type float

data_gen_joules
Energy used during the data generation phase, in Joules. Assumes that the SpiNNaker system has been shut down.

Return type float

data_gen_time_secs
Time taken by data generation phase, in seconds.

Return type float

exec_time_secs
Time taken by active simulation running, in seconds.

Return type float

fpga_exec_energy_joules
Energy used by all FPGAs during active simulation running, in Joules. This is *included* in the total FPGA energy.

Return type float

fpga_total_energy_joules
Energy used by all FPGAs in total, in Joules.

Return type float

get_core_active_energy_joules(*x*, *y*, *p*)
Energy used (above idle baseline) by a particular core, in Joules.

Unused cores always report 0.0 for this.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **p** (*int*) –

Return type float

get_router_active_energy_joules(*x*, *y*)
Energy used (above idle baseline) by a particular router, in Joules.

Unused routers always report 0.0 for this.

Parameters

- **x** (*int*) –
- **y** (*int*) –

Return type float

loading_joules
Energy used during data loading, in Joules.

Return type float

loading_time_secs
Time taken by data loading, in seconds.

Return type float

mapping_joules
Energy used during the mapping phase, in Joules. Assumes that the SpiNNaker system has been shut down.

Return type float

mapping_time_secs
Time taken by the mapping phase, in seconds.

Return type float

num_chips
The total number of chips used.

Return type int

num_cores
The total number of cores used, including for SCAMP.

Return type int

num_fpgas
The total number of FPGAs used.

Return type int

num_frames
The total number of frames used.

Return type int

packet_joules
Energy used by packet transmission, in Joules.

Return type float

saving_joules
Energy used during data extraction, in Joules.

Return type float

saving_time_secs
Time taken by data extraction, in seconds.

Return type float

total_energy_joules
Total of all energy costs, in Joules.

Return type float

total_time_secs
Time taken in total, in seconds.

Return type float

class spinn_front_end_common.utilities.utility_objs.**ProvenanceDataItem**(*names*, *value*, *report=False*, *message=''*)

Bases: object

Container for provenance data

Parameters

- **names** (*list(str)*) – A list of strings representing the naming hierarchy of this item
- **value** (*int or float or str*) – The value of the item
- **report** (*bool*) – True if the item should be reported to the user
- **message** (*str*) – 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

Return type str

names
The hierarchy of names of this bit of provenance data

Return type list(str)

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

Return type bool

value
The value of the item

Return type int or float or str

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

Bases: object

Represents a status information report from dropped packet reinjection.

Parameters

- **data** (*bytes*) – 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

Return type bool

is_reinjecting_multicast
True if re-injection of multicast packets is enabled

Return type bool

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

Return type bool

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

Return type bool

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

Return type int

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

Return type int

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

Return type 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

Return type int

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

Return type int

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

Return type int

router_wait1_timeout
The WAIT1 timeout value of the router, in cycles.

Return type int

router_wait1_timeout_parameters
The WAIT1 timeout value of the router as mantissa and exponent.

Return type tuple(int,int)

router_wait2_timeout
The WAIT2 timeout value of the router, in cycles.

Return type int

router_wait2_timeout_parameters
The WAIT2 timeout value of the router as mantissa and exponent.

Return type tuple(int,int)

Submodules

spinn_front_end_common.utilities.constants module

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

A listing of what SpiNNaker specific EIEIO commands there are.

BUFFER_READ = 0

BUFFER_WRITE = 1

spinn_front_end_common.utilities.constants.**CLOCKS_PER_US = 200**
The number of clock cycles per micro-second (at 200Mhz)

spinn_front_end_common.utilities.constants.**DATA_SPECABLE_BASIC_SETUP_INFO_N_BYTES = 80**
The number of words in the AbstractDataSpecable basic setup information. This is the amount required by the pointer table plus a SARK allocation.

spinn_front_end_common.utilities.constants.**DEFAULT_BUFFER_SIZE_BEFORE_RECEIVE = 16384**
The default size of a recording buffer before receive request is sent

spinn_front_end_common.utilities.constants.**DSE_DATA_STRUCT_SIZE = 16**
size of the on-chip DSE data structure required, in bytes

spinn_front_end_common.utilities.constants.**MAX_DATABASE_PATH_LENGTH = 50000**
Database file path maximum length for database notification messages. Note that this is *not* sent to SpiNNaker and so is not subject to the usual SDP limit.

spinn_front_end_common.utilities.constants.**MAX_POSSIBLE_BINARY_SIZE = 33792**
the ITCM max limit for a binary

spinn_front_end_common.utilities.constants.**MAX_SAFE_BINARY_SIZE = 32768**
the ITCM max safe limit for a binary

spinn_front_end_common.utilities.constants.**MAX_SIZE_OF_BUFFERED_REGION_ON_CHIP = 1048576**
max size expected to be used by the reverse ip_tag multicast source during buffered operations.

spinn_front_end_common.utilities.constants.**NOTIFY_PORT = 19999**
The default local port that the toolchain listens on for the notification protocol.

spinn_front_end_common.utilities.constants.**PARTITION_ID_FOR_MULTICAST_DATA_SPEED_UP = 'DATA**
partition IDs preallocated to functionality

spinn_front_end_common.utilities.constants.**SARK_PER_MALLOC_SDRAM_USAGE = 8**
The number of bytes used by SARK per memory allocation

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

SDP port handling output buffering data streaming

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

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

An enumeration.

SDP_CLEAR_IOBUF_CODE = 9

SDP_NEW_RUNTIME_ID_CODE = 7

SDP_STOP_ID_CODE = 6

SDP_UPDATE_PROVENCE_REGION_AND_EXIT = 8

spinn_front_end_common.utilities.constants.**SDRAM_BASE_ADDR = 1879048192**
start of where SDRAM starts (either unbuffered or buffered)

spinn_front_end_common.utilities.constants.**SIMULATION_N_BYTES = 12**
The number of bytes used by the simulation interface. This is one word for the machine_time_step, one for the SDP port, and one for the application hash.

spinn_front_end_common.utilities.constants.**SYSTEM_BYTES_REQUIREMENT = 92**
The number of bytes used by the DSG and simulation interfaces

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.**CantFindSDRAMToUseException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when malloc and sdram stealing cannot occur.

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: Exception

Raised when the front end detects an error

spinn_front_end_common.utilities.globals_variables module

spinn_front_end_common.utilities.globals_variables.**app_provenance_file_path**()
Returns the path to the directory that holds all app provenance files

This will be the path used by the last run call or to be used by the next run if it has not yet been called.

Rtpye str

Raises **ValueError** – if the system is in a state where path can't be retrieved

spinn_front_end_common.utilities.globals_variables.**config**()
Provides access to the configuration used or a best effort to the config to be used

If called before setup will return the values found in the config file BEFORE any cleanup alterations that setup could do.

After setup will return the simulation config even if end has been called

Return type *ConfigHandler*

spinn_front_end_common.utilities.globals_variables.**get_generated_output**(*output*)
Get one of the simulator outputs by name.

Parameters **output** (*str*) – The name of the output to retrieve.

Returns The value (of arbitrary type, dependent on which output), or *None* if the variable is not found.

Raises **ValueError** – if the system is in a state where outputs can't be retrieved

spinn_front_end_common.utilities.globals_variables.**get_not_running_simulator**()
Get the current simulator object and verify that it is not running.

Return type *SimulatorInterface*

spinn_front_end_common.utilities.globals_variables.**get_simulator**()
Get the current simulator object.

Return type *SimulatorInterface*

spinn_front_end_common.utilities.globals_variables.**has_simulator**()
Check if a simulator is operational.

Return type bool

spinn_front_end_common.utilities.globals_variables.**provenance_file_path**()
Returns the path to the directory that holds all provenance files

This will be the path used by the last run call or to be used by the next run if it has not yet been called.

Rtpye str

Raises **ValueError** – if the system is in a state where path can't be retrieved

spinn_front_end_common.utilities.globals_variables.**run_report_directory**()
Returns the path to the directory that holds all the reports for run

This will be the path used by the last run call or to be used by the next run if it has not yet been called.

Rtpye str

Raises **ValueError** – if the system is in a state where path can't be retrieved

spinn_front_end_common.utilities.globals_variables.**set_failed_state**(*new_failed_state*)
Install a marker to say that the simulator has failed.

Parameters **new_failed_state** (FailedState) – the failure marker

spinn_front_end_common.utilities.globals_variables.**set_simulator**(*new_simulator*)
Set the current simulator object.

Parameters **new_simulator** (SimulatorInterface) – The simulator to set.

spinn_front_end_common.utilities.globals_variables.**system_provenance_file_path**()
Returns the path to the directory that holds all provenance files

This will be the path used by the last run call or to be used by the next run if it has not yet been called.

Rtpye str

Raises **ValueError** – if the system is in a state where path can't be retrieved

spinn_front_end_common.utilities.globals_variables.**unset_simulator**(*to_cache_simulator=None*)
Destroy the current simulator.

Parameters **to_cache_simulator** (SimulatorInterface) – a cached version for allowing data extraction

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]*

Return type CoreSubsets

spinn_front_end_common.utilities.helpful_functions.**convert_time_diff_to_total_milliseconds**
Convert between a time diff and total milliseconds.

Parameters **sample** (*timedelta*) –

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

- **vertices** (*iterable(MachineVertex)*) – the vertices to convert to core subsets
- **placements** (*Placements*) – the placements object

Returns the CoreSubSets of the vertices

Return type CoreSubsets

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 tuple(dict(*ExecutableType*,CPUState), dict(*ExecutableType*,CPUState))

spinn_front_end_common.utilities.helpful_functions.**flood_fill_binary_to_spinnaker**(*executable_t*
bi-
nary,
txrx,
app_id)

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

Parameters

- **executable_targets** (*ExecutableTargets*) – the executable targets object
- **binary** (*str*) – the (name of the) binary to flood fill
- **txrx** (*Transceiver*) – spinnman instance
- **app_id** (*int*) – the application ID to load it as

Returns the number of cores it was loaded onto

Return type int

spinn_front_end_common.utilities.helpful_functions.**generate_unique_folder_name**(*folder*,
file-
name,
ex-
ten-
sion)

Generate a unique file name with a given extension in a given folder

Parameters

- **folder** (*str*) – where to put this unique file
- **filename** (*str*) – the name of the first part of the file without extension
- **extension** (*str*) – extension of the file

Returns file path with a unique addition

Return type str

spinn_front_end_common.utilities.helpful_functions.**get_defaultable_source_id**(*entry*)

Hack to support the source requirement for the router compressor on chip.

Parameters **entry** (*MulticastRoutingEntry*) – the multicast router table entry.

Returns return the source value

Return type int

spinn_front_end_common.utilities.helpful_functions.**get_ethernet_chip**(*machine*,
board_address)

Locate the chip with the given board IP address

Parameters

- **machine** (*Machine*) – the SpiNNaker machine

- **board_address** (*str*) – the board address to locate the chip of.

Returns The chip that supports that board address

Return type Chip

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*

Get the data speed up gatherer that can be used to talk to a particular chip. This will be on the same board.

Parameters

- **machine** (*Machine*) – The machine descriptor
- **placement_x** (*int*) – The X coordinate of the reference chip
- **placement_y** (*int*) – The Y coordinate of the reference chip
- **packet_gather_cores_to_ethernet_connection_map** (*dict(tuple(int,int), DataSpeedUpPacketGatherMachineVertex)*) –

Return type *DataSpeedUpPacketGatherMachineVertex*

spinn_front_end_common.utilities.helpful_functions.**locate_memory_region_for_placement**(*placem*

re-
gion,
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

Returns the address

Return type int

spinn_front_end_common.utilities.helpful_functions.**n_word_struct**(*n_words*)

Manages a precompiled cache of structs for parsing blocks of words. Thus, this:

```
data = n_word_struct(n_words).unpack(data_blob)
```

Is much like doing this:

```
data = struct.unpack("<{}I".format(n_words), data_blob)
```

except quite a bit more efficient because things are shared including the cost of parsing the format.

Parameters **n_words** (*int*) – The number of *SpiNNaker words* to be handled.

Returns A struct for working with that many words.

Return type Struct

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"

Parameters

- **config** (*ConfigParser*) – The configuration to look things up in.
- **section** (*str*) – The section name
- **item** (*str*) – The item name.

Return type str or 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"

Parameters

- **config** (*ConfigParser*) – The configuration to look things up in.
- **section** (*str*) – The section name
- **item** (*str*) – The item name.

Return type bool or None

spinn_front_end_common.utilities.helpful_functions.**read_config_float**(*config*, *section*, *item*)

Get the float value of a config item, returning None if the value is "None"

Parameters

- **config** (*ConfigParser*) – The configuration to look things up in.
- **section** (*str*) – The section name
- **item** (*str*) – The item name.

Return type float or 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"

Parameters

- **config** (*ConfigParser*) – The configuration to look things up in.
- **section** (*str*) – The section name
- **item** (*str*) – The item name.

Return type int or 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** (*int*) – chip x
- **y** (*int*) – chip y
- **address** (*int*) – base address of the SDRAM chip to read
- **length** (*int*) – length to read
- **data_format** (*str*) – the format to read memory (see `struct.pack()`)
- **transceiver** (*Transceiver*) – the SpinnMan interface

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

spinn_front_end_common.utilities.sqlite_db module

class spinn_front_end_common.utilities.sqlite_db.**Isolation**

Bases: `enum.Enum`

Transaction isolation levels for *SQLiteDB.transaction()*.

DEFERRED = 'DEFERRED'

Standard transaction type; postpones holding a lock until required.

EXCLUSIVE = 'EXCLUSIVE'

Take a write lock immediately. This is the strongest lock type.

IMMEDIATE = 'IMMEDIATE'

Take the lock immediately; this may be a read-lock that gets upgraded.

class spinn_front_end_common.utilities.sqlite_db.**SQLiteDB**(*database_file=None*, ***, *read_only=False*, *ddl_file=None*, *row_factory=<class 'sqlite3.Row'>*, *text_factory=<class 'memoryview'>*, *case_insensitive_like=True*)

Bases: `spinn_utilities.abstract_context_manager.AbstractContextManager`

General support class for SQLite databases. This handles a lot of the low-level detail of setting up a connection.

Basic usage (with the default row type):

```
with SQLiteDB("db_file.sqlite3") as db:
    with db.transaction() as cursor:
        for row in cursor.execute("SELECT thing FROM ..."):
            print(row["thing"])
```

This class is designed to be either used as above or by subclassing. See the SQLite SQL documentation for details of how to write queries, and the Python `sqlite3` module for how to do parameter binding.

Parameters

- **database_file** (*str*) – The name of a file that contains (or will contain) an SQLite database holding the data. If omitted, an unshared in-memory database will be used (suitable only for testing).
- **read_only** (*bool*) – Whether the database is in read-only mode. When the database is in read-only mode, it *must* already exist.
- **ddl_file** (*str or None*) – The name of a file (typically containing SQL DDL commands used to create the tables) to be evaluated against the database before this object completes construction. If `None`, nothing will be evaluated. You probably don't want to specify a DDL file at the same time as setting `read_only=True`.
- **row_factory** (*Callable or None*) – Callable used to create the rows of result sets. Either `tuple` or `sqlite3.Row` (default); can be `None` to use the DB driver default.
- **text_factory** (*Callable or None*) – Callable used to create the Python values of non-numeric columns in result sets. Usually `memoryview` (default) but should be `str` when you're expecting string results; can be `None` to use the DB driver default.
- **case_insensitive_like** (*bool*) – Whether we want the `LIKE` matching operator to be case-sensitive or case-insensitive (default).

close()
Finalises and closes the database.

connection
The underlying SQLite database connection.

> **Warning:** If you're using this a lot, consider contacting the SpiNNaker Software Team with details of your use case so we can extend the relevant core class to support you. *Normally* it is better to use *transaction()* to obtain a cursor with appropriate transactional guards.

Return type Connection

Raises **AttributeError** – if the database connection has been closed

pragma(*pragma_name*, *value*)
Set a database `PRAGMA`. See the SQLite PRAGMA documentation for details.

Parameters

- **pragma_name** (*str*) – The name of the pragma to set.
- **value** (*bool or int or str*) – The value to set the pragma to.

transaction(*isolation_level=None*)
Get a context manager that manages a transaction on the database. The value of the context manager is a `Cursor`. This means you can do this:

```
with db.transaction() as cursor:
    cursor.execute(...)
```

Parameters **isolation_level** (Isolation) – The transaction isolation level; note that this sets it for the connection! Can usually be *not* specified.

Return type ContextManager(Cursor)

spinn_front_end_common.utilities.system_control_logic module

spinn_front_end_common.utilities.system_control_logic.**run_system_application**(*executable_cores*, *app_id*, *transceiver*, *provenance_file_path*, *executable_finder*, *read_algorithm_iobuf*, *check_for_success_function*, *cpu_end_states*, *needs_sync_barrier*, *filename_template*, *binaries_to_track=None*, *progress_bar=None*, *logger=None*)

Executes the given _system_ application. Used for on-chip expanders, compressors, etc.

Parameters

- **executable_cores** (*ExecutableTargets*) – the cores to run the executable on
- **app_id** (*int*) – the app-id for the executable
- **transceiver** (*Transceiver*) – the SpiNNMan instance
- **provenance_file_path** (*str*) – the path for where provenance data is stored
- **executable_finder** (ExecutableFinder) – finder for executable paths
- **read_algorithm_iobuf** (*bool*) – whether to report IOBUFs
- **check_for_success_function** (*callable*) – function used to check success; expects *executable_cores*, *transceiver* as inputs
- **cpu_end_states** (*set(CPUState)*) – the states that a successful run is expected to terminate in
- **needs_sync_barrier** (*bool*) – whether a sync barrier is needed
- **filename_template** (*str*) – the IOBUF filename template.
- **binaries_to_track** (*list(str)*) – A list of binary names to check for exit state. Or *None* for all binaries

- **progress_bar** (*ProgressBar or None*) – Possible progress bar to update. end() will be called after state checked
- **logger** (*Logger*) – If provided and IOBUF is extracted, will be used to log errors and warnings

Raises **SpinnmanException** – If one should arise from the underlying SpiNNMan calls

Module contents

class spinn_front_end_common.utilities.**FailedState**

Bases: spinn_front_end_common.utilities.simulator_interface.SimulatorInterface

Marks that the simulator has failed (and replaces said simulator).

> **Warning:** Any method invoked on this object may fail with *ConfigurationException*.

add_socket_address(*socket_address*)

Add the address of a socket used in the run notification protocol.

Parameters **socket_address** (*SocketAddress*) – The address of the socket

Return type None

buffer_manager

The buffer manager being used for loading/extracting buffers

Return type *BufferManager*

config

Provides access to the configuration for front end interfaces.

Return type *ConfigHandler*

continue_simulation()

Continue a simulation that has been started in stepped mode

has_ran

Whether the simulation has executed anything at all.

Return type bool

machine

The python machine description object.

Return type Machine

machine_time_step

The machine timestep, in microseconds.

Return type int

no_machine_time_steps

The number of machine time steps.

Return type int

placements

Where machine vertices are placed on the machine.

Return type Placements

run(*run_time*, *sync_time=0.0*)
Run a simulation for a fixed amount of time

Parameters

- **run_time** (*int*) – the run duration in milliseconds.
- **sync_time** (*float*) – If not 0, this specifies that the simulation should pause after this duration. The continue_simulation() method must then be called for the simulation to continue.

stop()
End running of the simulation.

stop_run()
End the running of a simulation that has been started with run_forever

tags

Return type Tags

time_scale_factor

Return type int

transceiver
How to talk to the machine.

Return type Transceiver

use_virtual_board

Return type bool

verify_not_running()
Verify that the simulator is in a state where it can start running.

class spinn_front_end_common.utilities.**SimulatorInterface**
Bases: object

add_socket_address(*socket_address*)
Add the address of a socket used in the run notification protocol.

Parameters **socket_address** (*SocketAddress*) – The address of the socket

Return type None

buffer_manager
The buffer manager being used for loading/extracting buffers

Return type *BufferManager*

config
Provides access to the configuration for front end interfaces.

Return type *ConfigHandler*

continue_simulation()
Continue a simulation that has been started in stepped mode

has_ran
Whether the simulation has executed anything at all.

Return type bool

machine
The python machine description object.

Return type Machine

machine_time_step
The machine timestep, in microseconds.

Return type int

no_machine_time_steps
The number of machine time steps.

Return type int

placements
Where machine vertices are placed on the machine.

Return type Placements

run(*run_time*, *sync_time=0.0*)
Run a simulation for a fixed amount of time

Parameters

- **run_time** (*int*) – the run duration in milliseconds.
- **sync_time** (*float*) – If not 0, this specifies that the simulation should pause after this duration. The continue_simulation() method must then be called for the simulation to continue.

stop()
End running of the simulation.

stop_run()
End the running of a simulation that has been started with run_forever

tags

Return type Tags

time_scale_factor

Return type int

transceiver
How to talk to the machine.

Return type Transceiver

use_virtual_board

Return type bool

verify_not_running()
Verify that the simulator is in a state where it can start running.

class spinn_front_end_common.utilities.**IOBufExtractor**(*transceiver*, *executable_targets*, *executable_finder*, *app_provenance_file_path*, *system_provenance_file_path*, *from_cores='ALL'*, *binary_types=None*, *recovery_mode=False*, *filename_template='iobuf_for_chip_{}_{}_processor_id_*
suppress_progress=False)

Bases: object

Extract the logging output buffers from the machine, and separates lines based on their prefix.

Parameters

- **recovery_mode** (*bool*) –
- **filename_template** (*str*) –
- **suppress_progress** (*bool*) –
- **transceiver** (*Transceiver*) –
- **executable_targets** (*ExecutableTargets*) –
- **executable_finder** (ExecutableFinder) –
- **app_provenance_file_path** (*str or None*) –
- **system_provenance_file_path** (*str or None*) –
- **from_cores** (*str*) –
- **binary_types** (*str*) –

extract_iobuf()

Perform the extraction of IOBUF

Returns error_entries, warn_entries

Return type tuple(list(str),list(str))

1.1.1.5 spinn_front_end_common.utility_models package

Module contents

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

Bases: pacman.model.graphs.application.abstract.abstract_one_app_one_machine_vertex.AbstractOneAppOneMachineVertex

A utility for sending commands to a vertex (possibly an external device) at fixed times in the simulation or in response to simulation events (e.g., starting and stopping).

Parameters

- **label** (*str*) – The label of this vertex
- **constraints** (*iterable(AbstractConstraint)*) – Any initial constraints to this vertex

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(MultiCastCommand)*) – The commands to send when the simulation starts or resumes from pause
- **pause_stop_commands** (*iterable(MultiCastCommand)*) – The commands to send when the simulation stops or pauses after running
- **timed_commands** (*iterable(MultiCastCommand)*) – The commands to send at specific times

- **vertex_to_send_to** (*AbstractVertex*) – The vertex these commands are to be sent to

edges_and_partitions()
Construct application edges from this vertex to the app vertices that this vertex knows how to target (and has keys allocated for).

Returns edges, partition IDs

Return type tuple(list(ApplicationEdge), list(str))

class spinn_front_end_common.utility_models.**CommandSenderMachineVertex**(*label*, *constraints*, *app_vertex=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_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

Machine vertex for injecting packets at particular times or in response to particular events into a SpiNNaker application.

Parameters

- **label** (*str*) – The label of this vertex
- **constraints** (*iterable(AbstractConstraint)*) – Any initial constraints to this vertex
- **app_vertex** (CommandSender) –

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS
Bases: enum.Enum

An enumeration.

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(MultiCastCommand)*) – The commands to send when the simulation starts or resumes from pause
- **pause_stop_commands** (*iterable(MultiCastCommand)*) – the commands to send when the simulation stops or pauses after running

- **timed_commands** (*iterable(MultiCastCommand)*) – The commands to send at specific times
- **vertex_to_send_to** (*AbstractVertex*) – The vertex these commands are to be sent to

edges_and_partitions()
Construct machine edges from this vertex to the machine vertices that this vertex knows how to target (and has keys allocated for).

Returns edges, partition IDs

Return type tuple(list(MachineEdge), list(str))

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

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **machine_time_step** (*int*) –
- **time_scale_factor** (*int*) –
- **routing_infos** (*RoutingInfo*) – the routing infos

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*

classmethod get_n_command_bytes(*commands*)

Parameters **commands** (*list(MultiCastCommand)*) –

Return type int

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge partition that comes out of this vertex.

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

Returns A list of constraints

Return type list(AbstractConstraint)

get_timed_commands_bytes()

Return type int

resources_required
The resources required by the vertex

Return type ResourceContainer

class spinn_front_end_common.utility_models.**ChipPowerMonitor**(*label*, *n_samples_per_recording*, *sampling_frequency*, *constraints=None*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, pacman.model.partitioner_interfaces.legacy_partitioner_api.LegacyPartitionerAPI

Represents idle time recording code in a application graph.

Parameters

- **label** (*str*) – vertex label
- **constraints** (*iterable(AbstractConstraint)*) – constraints for the vertex
- **n_samples_per_recording** (*int*) – how many samples to take before recording to SDRAM the total
- **sampling_frequency** (*int*) – 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.

Returns The created machine vertex

Return type MachineVertex

get_resources_used_by_atoms(*vertex_slice*, *machine_time_step*, *time_scale_factor*)

Get the separate resource requirements for a range of atoms.

Parameters

- **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from
- **machine_time_step** (*int*) –
- **time_scale_factor** (*int*) –

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

Return type ResourceContainer

n_atoms

The number of atoms in the vertex

Returns The number of atoms

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.

static binary_file_name()
Return the string binary file name

Return type str

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

Returns starttype enum

Return type *ExecutableType*

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

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **spec** – data spec
- **machine_time_step** (*int*) – machine time step
- **time_scale_factor** (*int*) – time scale factor
- **data_n_time_steps** (*int*) – timesteps to reserve data for

Return type None Supports the application vertex calling this directly

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** (*Placement*) – the location on machine to get data from
- **buffer_manager** (BufferManager) – the buffer manager that might have data

Returns results, an array with 1 dimension of uint32 values

Return type ndarray

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** (*Transceiver*) – the SpiNNMan instance
- **placement** (*Placement*) – the placement object of the core to find the address of

Returns the base address of the recording region

Return type int

static get_resources(*time_step*, *time_scale_factor*, *n_samples_per_recording*, *sampling_frequency*)
Get the resources used by this vertex

Parameters

- **time_step** (*int*) –
- **time_scale_factor** (*int*) –
- **n_samples_per_recording** (*int*) –
- **sampling_frequency** (*float*) –

Return type ResourceContainer

n_samples_per_recording
how may samples to take between making recording entries

Return type int

resources_required
The resources required by the vertex

Return type ResourceContainer

sampling_frequency
how often to sample, in microseconds

Return type int

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.abstract.abstract_one_app_one_machine_vertex.AbstractOneAppOneMachineVertex

The gatherer for the data speed up protocols. Gatherers are only ever deployed on chips with an ethernet connection.

Parameters

- **x** (*int*) – Where this gatherer is.
- **y** (*int*) – Where this gatherer is.

- **extra_monitors_by_chip** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) – UNUSED
- **ip_address** (*str*) – How to talk directly to the chip where the gatherer is.
- **report_default_directory** (*str*) – Where reporting is done.
- **write_data_speed_up_reports** (*bool*) – Whether to write low-level reports on data transfer speeds.
- **constraints** (*iterable(AbstractConstraint)*) –

class spinn_front_end_common.utility_models.**DataSpeedUpPacketGatherMachineVertex**(*x*, *y*, *extra_monitors_*, *ip_address*, *report_default_d*, *write_data_sp*, *app_vertex=N*, *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

Machine vertex for handling fast data transfer between host and SpiNNaker. This machine vertex is only ever placed on chips with a working Ethernet connection; it collaborates with the *ExtraMonitorSupportMachineVertex* to write data on other chips..

Parameters

- **x** (*int*) – Where this gatherer is.
- **y** (*int*) – Where this gatherer is.
- **extra_monitors_by_chip** (*dict(tuple(int,int), ExtraMonitorSupportMachineVertex)*) – UNUSED
- **ip_address** (*str*) – How to talk directly to the chip where the gatherer is.
- **report_default_directory** (*str*) – Where reporting is done.
- **write_data_speed_up_reports** (*bool*) – Whether to write low-level reports on data transfer speeds.
- **constraints** (*iterable(AbstractConstraint) or None*) –
- **app_vertex** (*ApplicationVertex or None*) – The application vertex that caused this machine vertex to be created. If None, there is no such application vertex.

BASE_KEY = 4294967289

base key (really nasty hack to tie in fixed route keys)

BASE_MASK = 4294967291

to use with multicast stuff (reinjection acks have to be fixed route)

END_FLAG_KEY = 4294967286

END_FLAG_KEY_OFFSET = 3

FIRST_DATA_KEY = 4294967287

FIRST_DATA_KEY_OFFSET = 2

FLAG_FOR_MISSING_ALL_SEQUENCES = 4294967294

IN_REPORT_NAME = 'speeds_gained_in_speed_up_process.rpt'
report name for tracking performance gains

NEW_SEQ_KEY = 4294967288

NEW_SEQ_KEY_OFFSET = 1

OUT_REPORT_NAME = 'routers_used_in_speed_up_process.rpt'
report name for tracking used routers

TRAFFIC_TYPE = 2

TRANSACTION_ID_KEY = 4294967285

TRANSACTION_ID_KEY_OFFSET = 4

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

Returns the biggest sequence num expected

Return type int

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

Parameters

- **transceiver** (*Transceiver*) – the spinnMan interface
- **placements** (*Placements*) – the placements object

generate_data_specification(*spec*, *placement*, *machine_graph*, *routing_info*, *tags*, *mc_data_chips_to_keys*, *machine*, *app_id*, *router_timeout_key*)
Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **machine_graph** (*MachineGraph*) – (injected)
- **routing_info** (*RoutingInfo*) – (injected)
- **tags** (*Tags*) – (injected)
- **mc_data_chips_to_keys** (*dict(tuple(int,int),int)*) – (injected)
- **machine** (*Machine*) – (injected)
- **app_id** (*int*) – (injected)
- **router_timeout_key** (*dict(tuple(int,int),int)*) – (injected)

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

Parameters

- **extra_monitor** (ExtraMonitorSupportMachineVertex) – the extra monitor used for this data
- **extra_monitor_placement** (*Placement*) – placement object for where to get data from
- **memory_address** (*int*) – the address in SDRAM to start reading from
- **length_in_bytes** (*int*) – the length of data to read in bytes
- **fixed_routes** (*dict(tuple(int,int),FixedRouteEntry)*) – the fixed routes, used in the report of which chips were used by the speed up process

Returns byte array of the data

Return type bytearray

get_local_provenance_data()
Get an iterable of provenance data items.

Returns the provenance items

Return type iterable(*ProvenanceDataItem*)

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

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan instance
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set
- **placements** (*Placements*) – placements object

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

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan instance
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set
- **placements** (*Placements*) – placements object

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 gatherer and function to call for writing data onto SpiNNaker.

Parameters

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

Returns a write function of either a LPG or the spinnMan

Return type callable

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** (*int*) – chip x for data
- **y** (*int*) – chip y for data
- **base_address** (*int*) – the address in SDRAM to start writing memory
- **data** (*bytes or bytearray or memoryview or str*) – the data to write (or filename to load data from, if `is_filename` is True; that's the only time this is a str)
- **n_bytes** (*int*) – how many bytes to read, or None if not set
- **offset** (*int*) – where in the data to start from
- **cpu** (*int*) –
- **is_filename** (*bool*) – whether data is actually a file.

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** (*Transceiver*) – the SpiNNMan instance
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set
- **placements** (*Placements*) – placements object

set_router_wait1_timeout(*timeout*, *transceiver*, *placements*)
Set the wait1 field for a set of routers.

Parameters

- **timeout** (*tuple(int,int)*) –
- **transceiver** (*Transceiver*) –
- **placements** (*Placements*) –

set_router_wait2_timeout(*timeout*, *transceiver*, *placements*)
Set the wait2 field for a set of routers.

Parameters

- **timeout** (*tuple(int, int)*) –
- **transceiver** (*Transceiver*) –
- **placements** (*Placements*) –

classmethod static_resources_required()

Return type ResourceContainer

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** (*list(DataSpeedUpPacketGatherMachineVertex)*) – All the gatherers that are to be set
- **transceiver** (*Transceiver*) – the SpiNNMan instance
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set
- **placements** (*Placements*) – placements object

Returns a context manager

unset_cores_for_data_streaming(*transceiver*, *extra_monitor_cores*, *placements*)
Helper method for restoring the router timeouts to normal after being in a state usable for data streaming.

Parameters

- **transceiver** (*Transceiver*) – the SpiNNMan instance
- **extra_monitor_cores** (*list(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set
- **placements** (*Placements*) – placements object

update_transaction_id_from_machine(*txrx*)
Looks up from the machine what the current transaction ID is and updates the data speed up gatherer.

Parameters **txrx** (*Transceiver*) – SpiNNMan instance

class spinn_front_end_common.utility_models.**ExtraMonitorSupport**(*constraints*)
Bases: pacman.model.graphs.application.abstract.abstract_one_app_one_machine_vertex.AbstractOneAppOneMachineVertex

Control over the extra monitors.

Parameters **constraints** (*iterable(AbstractConstraint)*) – The constraints on the vertex

class spinn_front_end_common.utility_models.**ExtraMonitorSupportMachineVertex**(*constraints*, *app_vertex*, *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

Machine vertex for talking to extra monitor cores. Supports reinjection control and the faster data transfer protocols.

Usually deployed once per chip.

Parameters

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

clear_reinjection_queue(*transceiver*, *placements*, *extra_monitor_cores_to_set*)

Clears the queues for reinjection

Parameters

- **transceiver** (*Transceiver*) – the spinnMan interface
- **placements** (*Placements*) – the placements object
- **extra_monitor_cores_to_set** (*iterable(ExtraMonitorSupportMachineVertex)*) – Which extra monitors need to clear their queues.

generate_data_specification(*spec*, *placement*, *routing_info*, *machine_graph*, *data_in_routing_tables*, *mc_data_chips_to_keys*, *app_id*, *machine*, *router_timeout_keys*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **routing_info** (*RoutingInfo*) – (injected)
- **machine_graph** (*MachineGraph*) – (injected)
- **data_in_routing_tables** (*MulticastRoutingTables*) – (injected)
- **mc_data_chips_to_keys** (*dict(tuple(int,int),int)*) – (injected)
- **app_id** (*int*) – (injected)

- **machine** (*Machine*) – (injected)

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** (*Transceiver*) – the spinnMan interface
- **placements** (*Placements*) – the vertex placements

Returns the reinjection status for this vertex

Return type *ReInjectionStatus*

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** (*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 dict(tuple(int,int), *ReInjectionStatus*)

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

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

placement

Return type Placement

reinject_fixed_route

Return type bool

reinject_multicast

Return type bool

reinject_nearest_neighbour

Return type bool

reinject_point_to_point

Return type bool

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

Parameters

- **transceiver** (*Transceiver*) – the spinnMan interface
- **placements** (*Placements*) – the placements object
- **extra_monitor_cores_to_set** (*iterable(ExtraMonitorSupportMachineVertex)*) – which monitors control the routers to reset the counters of

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*) – placements object
- **extra_monitor_cores_for_data** (*iterable(ExtraMonitorSupportMachineVertex)*) – the extra monitor cores to set the packets of
- **transceiver** (*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.

set_router_wait1_timeout(*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. This sets the timeout for the time between when a packet arrives and when it starts to be emergency routed. (Actual emergency routing is disabled by default.)

Parameters

- **timeout** (*tuple(int,int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (*Transceiver*) – the spinnman interface

- **placements** (*Placements*) – vertex placements
- **extra_monitor_cores_to_set** (*iterable(ExtraMonitorSupportMachineVertex)*) – which monitors control the routers to set the timeout of

set_router_wait2_timeout(*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. This sets the timeout for the time between when a packet starts to be emergency routed and when it is dropped. (Actual emergency routing is disabled by default.)

Parameters

- **timeout** (*tuple(int, int)*) – The mantissa and exponent of the timeout value, each between 0 and 15
- **transceiver** (*Transceiver*) – the spinnMan instance
- **placements** (*Placements*) – vertex placements
- **extra_monitor_cores_to_set** (*iterable(ExtraMonitorSupportMachineVertex)*) – which monitors control the routers to set the timeout of

static static_get_binary_file_name()
The name of the binary implementing this vertex.

Return type str

static static_get_binary_start_type()
The type of the binary implementing this vertex.

Return type *ExecutableType*

static static_resources_required()
The resources required by this vertex.

Return type ResourceContainer

transaction_id

update_transaction_id()

update_transaction_id_from_machine(*txrx*)
looks up from the machine what the current transaction id is and updates the extra monitor.

Parameters **txrx** – SpiNNMan instance

Return type None

class spinn_front_end_common.utility_models.**LivePacketGather**(*lpg_params*, *constraints=None*)
Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, pacman.model.partitioner_interfaces.legacy_partitioner_api.LegacyPartitionerAPI

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.

Parameters

- **lpg_params** (LivePacketGatherParameters) –
- **constraints** (*iterable(AbstractConstraint)*) –

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.

Returns The created machine vertex

Return type MachineVertex

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

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utility_models.**LivePacketGatherMachineVertex**(*lpg_params*, *constraints=None*, *app_vertex=None*, *label=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`

Used to gather multicast packets coming from cores and stream them out to a receiving application on host. Only ever deployed on chips with a working Ethernet connection.

Parameters

- **lpg_params** (LivePacketGatherParameters) –
- **app_vertex** (LivePacketGather) –
- **label** (*str*) –
- **constraints** (*iterable(AbstractConstraint)*) –

TRAFFIC_IDENTIFIER = 'LPG_EVENT_STREAM'
Used to identify tags involved with the live packet gatherer.

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

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **machine_time_step** (*int*) –
- **time_scale_factor** (*int*) –
- **tags** (*Tags*) –

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

classmethod get_dtcm_usage()
Get the DTCM used by this vertex

Return type int

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

Parameters

- **transceiver** (*Transceiver*) – the SpinnMan interface object
- **placement** (*Placement*) – the placement of the object

Returns the provenance items

Return type iterable(*ProvenanceDataItem*)

classmethod get_sdram_usage()
Get the SDRAM used by this vertex

Return type int

is_in_injection_mode
Whether this vertex is actually in injection mode.

Return type bool

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 or None*) – The payload of the command
- **time** (*int or None*) – 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 *ConfigurationException* – If the repeat or delay are out of range

delay_between_repeats

Return type int

is_payload

Whether 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

Return type bool

is_timed

Whether this command is a timed command.

Return type bool

key

Return type int

payload

The payload of the command, or None if there is no payload.

Return type int or None

repeat

Return type int

time

The time within the simulation at which to send the command, or None if this is not a timed command

Return type int or None

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*, *enable_injection=False*, *splitter=None*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, pacman.model.partitioner_interfaces.legacy_partitioner_api.LegacyPartitionerAPI, spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints, 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** (*int*) – The number of keys to be sent via this multicast source
- **label** (*str*) – The label of this vertex
- **constraints** (*iterable(AbstractConstraint)*) – Any initial constraints to this vertex
- **max_atoms_per_core** (*int*) –
- **board_address** (*str or None*) – 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** (*int or None*) – 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** (*int*) – The SDP port to listen on for incoming event packets (defaults to 1)

- **receive_tag** (*IPTag*) – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** (*float*) – The estimated rate of packets that will be sent by this source
- **virtual_key** (*int*) – The base multicast key to send received events with (assigned automatically by default)
- **prefix** (*int*) – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** (*EIEIOPrefix*) – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** (*bool*) – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** (*ndarray(ndarray(numpy.int32)) or list(ndarray(int32)) or None*) – An array of arrays of times at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** (*str or None*) – The ID of the partition containing the edges down which the events are to be sent
- **reserve_reverse_ip_tag** (*bool*) – Extra flag for input without a reserved port
- **enable_injection** (*bool*) – Flag to indicate that data will be received to inject
- **splitter** (*None or AbstractSplitterCommon*) – the splitter object needed for this vertex

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.

Returns The created machine vertex

Return type MachineVertex

enable_recording(*new_state=True*)

get_outgoing_partition_constraints(*partition*)

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

Parameters **partition** (*AbstractOutgoingEdgePartition*) – 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

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

send_buffer_times
When messages will be sent.

Return type ndarray(ndarray(numpy.int32)) or list(ndarray(int32)) or None

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

A model which allows events to be injected into SpiNNaker and converted in to multicast packets.

Parameters

- **label** (*str*) – The label of this vertex
- **vertex_slice** (*Slice or None*) – The slice served via this multicast source
- **app_vertex** (*ReverseIpTagMultiCastSource or None*) – The associated application vertex
- **n_keys** (*int*) – The number of keys to be sent via this mulitcast source (can't be None if vertex_slice is also None)
- **constraints** (*iterable(AbstractConstraint)*) – Any initial constraints to this vertex
- **board_address** (*str*) – 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** (*int*) – The port on the board that will listen for incoming event packets (default is to disable this feature; set a value to enable it, or set the *reserve_reverse_ip_tag parameter* to True if a random port is to be used)
- **receive_sdp_port** (*int*) – The SDP port to listen on for incoming event packets (defaults to 1)
- **receive_tag** (*int*) – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** (*float*) –
- **virtual_key** (*int*) – The base multicast key to send received events with (assigned automatically by default)

- **prefix** (*int*) – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** (*EIEIOPrefix*) – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** (*bool*) – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** (*ndarray*) – An array of arrays of time steps at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** (*str*) – The ID of the partition containing the edges down which the events are to be sent
- **reserve_reverse_ip_tag** (*bool*) – True if the source should set up a tag through which it can receive packets; if port is set to None this can be used to enable the reception of packets on a randomly assigned port, which can be read from the database
- **enable_injection** (*bool*) – Flag to indicate that data will be received to inject

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

Parameters **new_state** (*bool*) –

generate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *machine_graph*, *routing_info*, *first_machine_time_step*, *data_n_time_steps*, *run_until_timesteps*)
Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – The placement the vertex is located at
- **machine_time_step** (*int*) –
- **time_scale_factor** (*int*) –
- **machine_graph** (*MachineGraph*) –
- **routing_info** (*RoutingInfo*) –
- **first_machine_time_step** (*int*) –
- **data_n_time_steps** (*int*) –
- **run_until_timesteps** (*int*) –

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 partition that comes out of this vertex.

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

Returns A list of constraints

Return type list(AbstractConstraint)

get_provenance_data_from_machine(*transceiver*, *placement*)
Retrieve the provenance data.

Parameters

- **transceiver** (*Transceiver*) – How to talk to the machine
- **placement** (*Placement*) – Which vertex are we retrieving from, and where was it

Return type list(*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** (*Transceiver*) – the SpiNNMan instance
- **placement** (*Placement*) – the placement object of the core to find the address of

Returns the base address of the recording region

Return type int

get_region_buffer_size(*region*)

Parameters **region** (*int*) – Region ID

Returns Size of buffer, in bytes.

Return type int

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

Parameters

- **send_buffer_times** (*ndarray(ndarray(numpy.int32)) or list(ndarray(numpy.int32)) or None*) – When events will be sent
- **recording_enabled** (*bool*) – Whether recording is done
- **machine_time_step** (*int*) – What the machine timestep is
- **receive_rate** (*float*) – What the expected message receive rate is
- **n_keys** (*int*) – How many keys are being sent

Return type VariableSDRAM

is_in_injection_mode
Whether this vertex is actually in injection mode.

Return type bool

mask

Return type int or None

resources_required
The resources required by the vertex

Return type ResourceContainer

send_buffer_times
When events will be sent.

Return type ndarray(ndarray(numpy.int32)) or list(ndarray(numpy.int32)) or None

send_buffers

Return type dict(int,*BufferedSendingRegion*)

update_buffer (*first_machine_time_step*, *run_until_timesteps*)

Updates the buffers on specification of the first machine timestep. Note: This is called by injection.

Parameters

- **first_machine_time_step** (*int*) – The first machine time step in the simulation
- **run_until_timesteps** (*int*) – The last machine time step in the simulation

virtual_key

Return type int or None

1.1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

C

D

F

G

H

I

M

N

R

S

V

W