SpiNNakerGraphFrontEnd Documentation

Jan 29, 2020

Contents

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

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

Contents:

CHAPTER 1

spinnaker_graph_front_end package

1.1 Subpackages

1.1.1 spinnaker_graph_front_end.utilities package

1.1.1.1 Submodules

1.1.1.2 spinnaker_graph_front_end.utilities.data_utils module

spinnaker_graph_front_end.utilities.data_utils.**generate_system_data_region**(*spec*, *region_id*, *machine_vertex*, *machine_time_step*, *time_scale_factor*)

1.1.1.3 spinnaker_graph_front_end.utilities.simulator_vertex module

class spinnaker_graph_front_end.utilities.simulator_vertex.**SimulatorVertex**(*label*, *binary_name*, *constraints=None*)

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary

A machine vertex that is implemented by a binary APLX that supports the spin1_api simulation control protocol.

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

Return type str

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

Return type ExecutableType

1.1.1.4 Module contents

class spinnaker_graph_front_end.utilities.**SimulatorVertex**(*label*, *binary_name*, *constraints=None*)
Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary

A machine vertex that is implemented by a binary APLX that supports the spin1_api simulation control protocol.

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

Return type str

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

Return type ExecutableType

1.2 Submodules

1.3 spinnaker_graph_front_end.spinnaker module

class spinnaker_graph_front_end.spinnaker.**GraphFrontEndSimulatorInterface**
Bases: spinn_front_end_common.utilities.simulator_interface.SimulatorInterface

The simulator interface exported by the graph front end. A very thin layer over the capabilities of the Front End Common package.

spinnaker_graph_front_end.spinnaker.**SPALLOC_CORES = 48**
The default number of cores to ask spalloc for

class spinnaker_graph_front_end.spinnaker.**SpiNNaker**(*executable_finder*, *host_name=None*, *graph_label=None*, *database_socket_addresses=None*, *dsg_algorithm=None*, *n_chips_required=None*, *extra_pre_run_algorithms=None*, *extra_post_run_algorithms=None*, *time_scale_factor=None*, *machine_time_step=None*, *default_config_paths=None*, *extra_xml_paths=None*)

Bases: spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase, *spinnaker_graph_front_end.spinnaker.GraphFrontEndSimulatorInterface*

The implementation of the SpiNNaker simulation interface.

CONFIG_FILE_NAME = 'spiNNakerGraphFrontEnd.cfg'

VALIDATION_CONFIG_NAME = 'validation_config.cfg'

The name of the configuration validation configuration file

is_allocated_machine

Is this an allocated machine? Otherwise, it is local.

run(*run_time*)

Run a simulation for a fixed amount of time

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

1.4 Module contents

class spinnaker_graph_front_end.**LivePacketGather**(*hostname=None*, *port=None*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*, *constraints=None*, *label=None*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary

A model which stores all the events it receives during a timer tick and then compresses them into Ethernet packets and sends them out of a SpiNNaker machine.

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

Create a machine vertex from this application vertex

Parameters

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

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

Parameters

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

Return type None

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

Return type str

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

Return type ExecutableType

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

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

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

Return type ResourceContainer

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

n_atoms
The number of atoms in the vertex

Return type int

class spinnaker_graph_front_end.**ReverseIpTagMultiCastSource**(*n_keys*, *label=None*, *constraints=None*, *max_atoms_per_core=92233720368547758*
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=None*, *reserve_reverse_ip_tag=False*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary, spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl

A model which will allow events to be injected into a SpiNNaker machine and converted into multicast packets.

Parameters

- **n_keys** (*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** (*numpy.ndarray(numpy.ndarray(numpy.int32)) or list(numpy.ndarray(numpy.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

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

Parameters

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

enable_recording(*new_state=True*)

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

Parameters

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

Return type None

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

Return type str

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

Return type ExecutableType

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

Parameters **partition** (*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

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

n_atoms
The number of atoms in the vertex

Return type int

send_buffer_times
When messages will be sent.

class spinnaker_graph_front_end.**MachineEdge**(*pre_vertex*, *post_vertex*, *traffic_type=<EdgeTrafficType.MULTICAST: 1>*, *label=None*, *traffic_weight=1*)

Bases: pacman.model.graphs.abstract_edge.AbstractEdge

A simple implementation of a machine edge.

Parameters

- **pre_vertex** (pacman.model.graphs.machine.MachineVertex) – the vertex at the start of the edge
- **post_vertex** (pacman.model.graphs.machine.MachineVertex) – the vertex at the end of the edge
- **traffic_type** (pacman.model.graphs.common.EdgeTrafficType) – The type of traffic that this edge will carry
- **label** (*str*) – The name of the edge
- **traffic_weight** (*int*) – the optional weight of traffic expected to travel down this edge relative to other edges (default is 1)

label
The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex
The vertex at the end of the edge

Return type pacman.model.graphs.abstract_vertex.AbstractVertex

pre_vertex
The vertex at the start of the edge

Return type pacman.model.graphs.abstract_vertex.AbstractVertex

traffic_type
The traffic type of the edge

Return type pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType

traffic_weight
The amount of traffic expected to go down this edge relative to other edges

spinnaker_graph_front_end.**setup**(*hostname=None*, *graph_label=None*, *model_binary_module=None*, *model_binary_folder=None*, *database_socket_addresses=None*, *user_dsg_algorithm=None*, *n_chips_required=None*, *extra_pre_run_algorithms=None*, *extra_post_run_algorithms=None*, *time_scale_factor=None*, *machine_time_step=None*)

Parameters

- **hostname** (*str*) – the hostname of the SpiNNaker machine to operate on (over rides the machine_name from the cfg file).
- **graph_label** (*str*) – a human readable label for the graph (used mainly in reports)
- **model_binary_module** (*python module*) – the module where the binary files can be found for the c code that is being used in this application; mutually exclusive with the model_binary_folder.
- **model_binary_folder** (*str*) – the folder where the binary files can be found for the c code that is being used in this application; mutually exclusive with the model_binary_module.
- **database_socket_addresses** (*list of SocketAddresses*) – set of SocketAddresses that need to be added for the database notification functionality. This are over and above the ones used by the LiveEventConnection
- **user_dsg_algorithm** (*str*) – an algorithm used for generating the application data which is loaded onto the machine. if not set, will use the data specification language algorithm required for the type of graph being used.
- **n_chips_required** (*int or None*) – if you need to be allocated a machine (for spalloc) before building your graph, then fill this in with a general idea of the number of chips you need so that the spalloc system can allocate you a machine big enough for your needs.
- **extra_pre_run_algorithms** (*list of str*) – algorithms which need to be ran after mapping and loading has occurred but before the system has ran. These are plugged directly into the work flow management.
- **extra_post_run_algorithms** (*list of str*) – algorithms which need to be ran after the simulation has ran. These could be post processing of generated data on the machine for example.

spinnaker_graph_front_end.**run**(*duration=None*)

Method to support running an application for a number of microseconds.

Parameters **duration** (*int*) – the number of microseconds the application should run for

spinnaker_graph_front_end.**stop**()

Do any necessary cleaning up before exiting. Unregisters the controller

spinnaker_graph_front_end.**read_xml_file**(*file_path*)

Reads a xml file and translates it into an application graph and machine graph (if required).

Parameters **file_path** – the file path in absolute form

Return type None

spinnaker_graph_front_end.**add_vertex_instance**(*vertex_to_add*)

Add an existing application vertex to the unpartitioned graph.

Parameters **vertex_to_add** (AbstractPartitionableVertex) – vertex instance to add to the graph

Return type None

spinnaker_graph_front_end.**add_vertex**(*cell_class*, *cell_params*, *label=None*, *constraints=None*)

Create an application vertex and add it to the unpartitioned graph.

Parameters

- **cell_class** (*class*) – the class object for creating the application vertex
- **cell_params** (*dict(str,object)*) – the input parameters for the class object
- **constraints** (list(AbstractConstraint) or None) – any constraints to be applied to the vertex once built
- **label** (*str or None*) – the label for this vertex

Returns the application vertex instance object

spinnaker_graph_front_end.**add_machine_vertex**(*cell_class*, *cell_params*, *label=None*, *constraints=None*)

Create a machine vertex and add it to the partitioned graph.

Parameters

- **cell_class** (*class*) – the class of the machine vertex to create
- **cell_params** (*dict(str,object)*) – the input parameters for the class object
- **constraints** (list(AbstractConstraint) or None) – any constraints to be applied to the vertex once built
- **label** (*str or None*) – the label for this vertex

Returns the machine vertex instance object

spinnaker_graph_front_end.**add_machine_vertex_instance**(*vertex_to_add*)

Add an existing machine vertex to the partitioned graph.

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

Return type None

spinnaker_graph_front_end.**add_edge**(*edge_type*, *edge_parameters*, *semantic_label*, *label=None*)

Create an application edge and add it to the unpartitioned graph.

Parameters

- **edge_type** – the kind (class) of application edge to create
- **edge_parameters** – dict of parameters to pass to the constructor
- **semantic_label** – the ID of the partition that the edge belongs to
- **label** – textual label for the edge, or None

Returns the created application edge

spinnaker_graph_front_end.**add_application_edge_instance**(*edge*, *partition_id*)

spinnaker_graph_front_end.**add_machine_edge**(*edge_type*, *edge_parameters*, *semantic_label*, *label=None*)

Create a machine edge and add it to the partitioned graph.

Parameters

- **edge_type** – the kind (class) of machine edge to create

- **edge_parameters** – dict of parameters to pass to the constructor
- **semantic_label** – the ID of the partition that the edge belongs to
- **label** – textual label for the edge, or None

Returns the created machine edge

spinnaker_graph_front_end.**add_machine_edge_instance**(*edge*, *partition_id*)

spinnaker_graph_front_end.**add_socket_address**(*database_ack_port_num*, *database_notify_host*, *database_notify_port_num*)

Adds a socket address for the notification protocol.

Parameters

- **database_ack_port_num** – port number to send acknowledgement to
- **database_notify_host** – host IP to send notification to
- **database_notify_port_num** – port that the external device will be notified on.

spinnaker_graph_front_end.**get_txrx**()

Gets the transceiver used by the tool chain.

spinnaker_graph_front_end.**has_ran**()

spinnaker_graph_front_end.**machine_time_step**()

spinnaker_graph_front_end.**get_number_of_available_cores_on_machine**()

Gets the number of cores on this machine that are available to the simulation.

spinnaker_graph_front_end.**no_machine_time_steps**()

spinnaker_graph_front_end.**timescale_factor**()

spinnaker_graph_front_end.**machine_graph**()

spinnaker_graph_front_end.**application_graph**()

spinnaker_graph_front_end.**routing_infos**()

spinnaker_graph_front_end.**placements**()

spinnaker_graph_front_end.**transceiver**()

spinnaker_graph_front_end.**graph_mapper**()

spinnaker_graph_front_end.**buffer_manager**()

Returns the buffer manager being used for loading/extracting buffers

spinnaker_graph_front_end.**machine**()

spinnaker_graph_front_end.**is_allocated_machine**()

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index