```
import functools
import operator
from typing import List, Dict

from log_level import LogLevel


class Simulator:
    """
        The Simulator class handles the simulation of our high level model of the fpga functionality of the stencil chain
        design (see paper example ref# TODO).

        interface for FPGA-like execution (gets called from the scheduler)

        - read:
                - saturation phase: read unconditionally
                - execution phase: read all inputs iff they are available
        - execute:
                - saturation phase: do nothing
                - execution phase: if input read, execute stencil using the input
        - write:
                - saturation phase: do nothing
                - execution phase: write result from execution to output buffers
                    --> if output buffer overflows: assumptions about size was wrong!
    """

    def __init__(self,
                 input_config_name: str,
                 input_nodes: Dict,
                 input_config: Dict,
                 kernel_nodes: Dict,
                 output_nodes: Dict,
                 dimensions: List,
                 write_output: bool,
                 log_level: int) -> None:
        """
        Create new Simulator class with given initialization parameters.
        :param input_config_name: name of the input file
        :param input_nodes: dict of all input nodes
        :param input_config: input config dict
        :param kernel_nodes: dict of all kernel nodes
        :param output_nodes: dict of all output nodes
        :param dimensions: global problem size dimensions
        :param write_output: flag for defining whether or not to write the result to a file
        :param log_level: flag for console output logging
        """
        # save params
        self.input_config_name: str = input_config_name
        self.dimensions: List = dimensions
```

```
    self.input_nodes: Dict = input_nodes
    self.input_config: Dict = input_config
    self.kernel_nodes: Dict = kernel_nodes
    self.output_nodes: Dict = output_nodes
    self.write_output: bool = write_output
    self.log_level: int = log_level

def step_execution(self):
    """
    Execute one step/cycle (read & execute & write).
    """
    """
        try to read all kernel inputs
    """
    # read output nodes
    for output in self.output_nodes:
        try:
            self.output_nodes[output].try_read()
        except Exception as ex:  # error
            self.diagnostics(ex)
    # read kernel nodes
    for kernel in self.kernel_nodes:
        try:
            # reset the internal state to be ready for a new computation
            self.kernel_nodes[kernel].reset_old_compute_state()
            self.kernel_nodes[kernel].try_read()
        except Exception as ex:  # error
            self.diagnostics(ex)
    """
        try to execute all kernels
    """
    # execute kernel nodes
    for kernel in self.kernel_nodes:
        try:
            self.kernel_nodes[kernel].try_execute()
        except Exception as ex:
            self.diagnostics(ex)

    """
        try to write all kernel outputs
    """
    # write input nodes
    for input in self.input_nodes:
        try:
            self.input_nodes[input].try_write()
        except Exception as ex:
            self.diagnostics(ex)
    # write kernel nodes
    for kernel in self.kernel_nodes:
```

```
            try:
                self.kernel_nodes[kernel].try_write()
            except Exception as ex:
                self.diagnostics(ex)
        """
            update performance metrics
        """
        for kernel in self.kernel_nodes:
            try:
                self.kernel_nodes[kernel].update_performance_metric()
            except Exception as ex:
                self.diagnostics(ex)

    def initialize(self):
        """
        Initialize the input nodes with data given in the config file.
        """
        # loop over all input nodes
        if self.log_level >= LogLevel.BASIC.value:
            print("Initialize simulator input arrays.")
        for input in self.input_nodes:
            # import data
            self.input_nodes[input].init_input_data(self.input_config)

    def finalize(self):
        """
        Do the necessary post-processsing after the simulator completed the step execution.
        """
        # check if write flag set
        if self.write_output:
            # save data to files
            for output in self.output_nodes:
                self.output_nodes[output].write_result_to_file(self.input_config_name)
        # output kernel performance metric
        if self.log_level >= LogLevel.BASIC.value:
            for kernel in self.kernel_nodes:
                self.kernel_nodes[kernel].print_kernel_performance()

    def get_result(self):
        """
        Returns the result stored in all output nodes.
        """
        # add all output node data to the dictionary
        result_dict = dict()
        for output in self.output_nodes:
            result_dict[output] = self.output_nodes[output].data_queue.export_data()
        return result_dict

    def all_done(self) -> bool:
```

```
    """
    Check if all nodes completed their execution.
    :return:
    """
    # compute the total problem size
    total_elements = functools.reduce(operator.mul, self.dimensions)
    # check if all input nodes completed their write
    for input in self.input_nodes:
        if self.input_nodes[input].program_counter < total_elements:
            return False
    # check if all kernel nodes completed their execution
    for kernel in self.kernel_nodes:
        if self.kernel_nodes[kernel].program_counter < total_elements:
            return False
    # check if all output nodes completed their read
    for output in self.output_nodes:
        if self.output_nodes[output].program_counter < total_elements:
            return False
    return True

def simulate(self):
    """
    Run the main simulation loop
    """
    # init
    if self.log_level >= LogLevel.BASIC.value:
        print("Initialize simulation.")
    self.initialize()
    # run simulation
    if self.log_level >= LogLevel.BASIC.value:
        print("Run simulation.")
    PC = 0
    while not self.all_done():
        if self.log_level >= LogLevel.FULL.value:  # output program counter of each node
            print("Execute next step. Current global program counter: {}.".format(PC))
        # execute
        self.step_execution()
        # increment program counter
        PC += 1
        if self.log_level >= LogLevel.FULL.value:  # output program counter of each node
            for input in self.input_nodes:
                print("input:{}, PC: {}".format(input, self.input_nodes[input].program_counter))
            for kernel in self.kernel_nodes:
                print("kernel:{}, PC: {}".format(kernel, self.kernel_nodes[kernel].program_counter))
            for output in self.output_nodes:
                print("output:{}, PC: {}".format(output, self.output_nodes[output].program_counter))
    # write completion message
    if self.log_level >= LogLevel.BASIC.value:
        print("Simulation completed after {} cycles.".format(PC))
```

```
    # finalize the simulation
    if self.log_level >= LogLevel.BASIC.value:
        print("Finalize simulation.")
    self.finalize()
    if self.log_level >= LogLevel.BASIC.value:
        print("Create simulator report.")
        self.report()
    if self.log_level >= LogLevel.BASIC.value:
        print("Write result report.")
        for out in self.output_nodes:
            print("name: {}, data: {}".format(out, self.output_nodes[out].data_queue.export_data()))

def report(self):
    if self.log_level >= LogLevel.BASIC.value:
        print("Create simulator report.")
    self.diagnostics(None)

def diagnostics(self, exception):
    if exception is not None:
        print("Error: Exception {} has been risen. Run diagnostics.".format(exception.__traceback__))
    if self.log_level >= LogLevel.BASIC.value:
        print("Run diagnostics of {}.".format(self.input_config_name))
    # print info about all inputs
    for input in self.input_nodes:
        print("input:{}, PC: {}".format(input, self.input_nodes[input].program_counter))
    # call debug diagnostics output of all kernels
    for kernel in self.kernel_nodes:
        self.kernel_nodes[kernel].diagnostics(exception)
    # print info about all outputs
    for output in self.output_nodes:
        print("output:{}, PC: {}".format(output, self.output_nodes[output].program_counter))
```