Python To VHDL Converter

Authors :

Name:	E-mail:
Mario Morcos	mariomorcoswassily@gmail.com
Mohamed Mahmoud	mohamed_hesham23@outlook.com
Sameh Maher Kalach	samehkalash0@gmail.com
Omar Khaled	school.omarkhaled@gmail.com
Mohamed Ahmed	mohameed.ahmedd077@gmail.com

Introduction

Welcome to **Python to VHDL Convertor** tool a Python-based tool designed to convert Python code into VHDL (VHSIC Hardware Description Language).It is considered to be a transpiler not a compiler as it doesn’t evaluate the Python code expressions it changes expressions written in Python to expressions written in VHDL code

This tool aims to bridge the gap between software and hardware design, providing a streamlined approach for engineers and developers to translate their Python algorithms and functionalities into hardware descriptions suitable for FPGA (Field-Programmable Gate Array) implementations.

The integration of Python and VHDL opens up possibilities for rapid prototyping, allowing software concepts to be efficiently transformed into hardware designs. Through this converter, users can harness the simplicity and flexibility of Python programming while leveraging the robustness and efficiency of VHDL for hardware development.

This documentation serves as a guide to understanding and utilizing **Python to VHDL Converter** tool effectively. It covers the usage instructions, supported Python constructs, VHDL output format, and examples demonstrating the conversion process. this tool aims to facilitate the transition and enhance the development workflow.

We hope that **Python to VHDL Converter** tool enables faster iterations and innovative hardware design exploration.

Please refer to this documentation for comprehensive insights into using the tool, and feel free to reach out with any questions, feedback, or suggestions.

Contents:

I. Basic Converter

Overview :

Basic Converter is the part which is responsible for converting both entities and architectures as it imports functionalities from other modules or files like logic converter, files, and components.

Classes:

A) **Entity**
B) **Architecture**
C) **Create Component**
D) **Input**
E) **Output**
F) **Signal**
G) **Package**

A) Entity

The entity class converts **entities** taken from the user by the help of the decorator (@) to allocate the entity class written by the user and change it to VHDL code

Attributes:

entity_class

Functions:

1) _init_

- This function is a constructor for **Entity class**
- It's parameters are **self** & **entity_class** .

2) convert_entity

- This function converts class Entity to VHDL code
- It's parameters are **self**
- It returns **str: VHDL** for class Entity

3) create_port

- This function creates **VHDL port**
- It's Parameters are **self**
- It returns **str: The VHDL code** for a port with inputs and output as block

B) Architecture

The architecture class converts **Architectures** taken from the user by the help of the decorator (@) to allocate the architecture class written by the user and change it to VHDL code however, the logic part in the in the architecture have to be passed to the Logic Convertor to achieve complete parsing of the architecture

Attributes:

file_name

architecture_class

Functions:

1) _init_
- This function is a constructor for **Architecture class**
- It's parameters are **self** & **architecture_class**

2) convert_arch
- This function converts class Architecture to VHDL code
- It's parameters are **self**
- It returns **str: architecture block** as VHDL code

3) create_constants
- This function converts **Python constants** to VHDL code
- It's parameters are **self**
- It returns **str: Constant** as VHDL code

C) Create Component

The **CreateComponent class** contains functions made to help in creating components based on entities and ports

Attributes:

none

Functions:

1) _init_
    - This function is a constructor for **CreateComponent class**
    - It's parameters are **self**

2) create_ component
    - This function converts python component to VHDL code
    - It's parameters are **self**
    - It returns **str: VHDL component**

3) create_port
    - This function creates **VHDL port**
    - It's parameters are **self**
    - It returns **str: The VHDL** code for a port with inputs and output as block

D) Input

The input class contains functions made to help in creating input blocks

Attributes:

name

type

Functions:

1) _init_
    - This function is a constructor for **Input class**
    - It's parameters are **self** & **name** & **type**

2) _str_

- This function is responsible for providing the program with the Input blocks in VHDL code
- It's parameters are **self**
- It returns **str: Input as VHDL**

E) Output

The **output class** contains functions made to help in creating output blocks

Attributes:

name

type

Functions:

1) _init_

- This function is a constructor for **Output class**

- It's parameters are **self** & **name** & **type**

2) _str_

This function is responsible for providing the program with the Output blocks in VHDL code

It's parameters **self**

It returns **str: Output as VHDL**

F) Signal

The **Signal class** contains functions made to help in creating signal blocks

Attributes:

name

type

Functions:

1) _init_

This function is a constructor for **Signal class**

It’s Parameters are **self** , **name** & **type**

2) _str_

This function is responsible for providing the program with the signal blocks in VHDL code

It’s parameters is **self**

It returns **str: Signal as VHDL**

G) Package

The Package class represents VHDL libraries and packages providing a method to retrieve them as required for VHDL code generation.

Functions:

1) _init_

This function is a constructor for **Package class**

It’s parameters is **self**

2) get_Packages

This function is responsible for returning standard libraries and packages in VHDL code

It’s parameters is **self**

It returns str: Standard libraries and packages used by default VHDL program

II. Logic Converter

Overview :

Converter to revolve around reading and processing files containing logic-related information. It extracts specific sections marked as logic-related and perform some further processing or formatting on these sections using the Capture class

Classes :

A) Infix

A) Class Infix

The Infix Class defines a class used for representing various infix operators Instances of this class are created but not used

Functions:

A) get_lines

This function reads the statements in the file

It’s parameter are **file_path**

It returns **list : List of statements** as VHDL code

B) find_leading_white_space

This function calculates number of leading white space

It’s parameter is **line**

It returns **int: Number of leading white space**

C) process_lines

This function strips new lines in statements

It's parameter is **list : lines**

It returns **list : Lines after being processed**

D) get_logic

This function finds lines if logic code inside the file

It returns **list : Logic lines** as VHDL code

E) parse_file

This function collects processed logic statements and send it back as a parameter to capture class and then returns it back as list of tokens

It returns **list : Parsed text** as VHDL code

III. Lexer

Overview :

Lexer is the lexical analyzer for text conversion into meaningful tokens based on categories. These tokens categorize various parts of the VHDL codebase for further processing and manipulation

Classes :

A) Tokens
B) Lexer

A) Tokens

The **Tokens class** represents a token with attributes for its name

Attributes:

- name
- replace_with
- type
- in_middle

Functions:

1) _init_

   This function is a constructor for **Tokens class**

   It's parameters are **self** & **token_name** & **replace_with** & **i n_middle=True** & **type=None**

2) _repr_

This function is responsible for representing token format

It’s parameters is **self**

It returns **Equivalent VHDL code** of the token

3) _str_

This function is responsible for displaying the tokens inside the program functions

It’s parameters is **self**

It returns **str: Representation** of the Token class

B) Lexer

The Lexer class iterates through each character in the input text.

Identifies and extracts numbers and strings based on specific rules.

Attributes:

text

pos

current_char

Functions:

1) _init_

This function is a constructor for Lexer class

It’s parameters are **self** and **text**

2) advance

This Function is responsible for advancing to the next character to the text

It’s parameter is **self**

3) make_tokens

This function is responsible for Tokenization of text by calling functions (make_number() and make_string())

It’s parameter is **self**

It returns **list(str): List of tokens**

4) make_number

This function is responsible for tokenization of numbers

It’s parameter is **self**

It returns **str: The tokenized number**

5) make_string

This function is Responsible for tokenization strings as tokens or strings

It’s parameter is **self**

It returns **list: Tokens**

6) string_in_tokens

This function loops on list of tokens and checks whether if a string inside list of tokens or not

It’s parameters are **self** & **string**

It returns **str : A token inside list of tokens in case the is True**

It returns **boolean : False in case there is no string inside list of tokens**

Functions:

A) **register_token**

This function adds token to tokens list

It’s parameter is **token_class**

B) **parse_text**

This function parse tokens as VHDL code

It’s parameter is **text**

It returns **str: VHDL statements**

IV. Filters

Overview :

Responsible for tokenization of blocks into statements then rearrange and parse them into VHDL code The code goes through lines of Python code and identifies patterns resembling VHDL constructs.

Classes :

A) **Condition_token**
B) **Match_Case_Token**
C) **While_loop_token**
D) **For_loop_token**
E) **Statemnet_filter**
F) **Process_filter**

G) If_condition_filter

H) Match_Case_condition_filter

I) For_loop_filter

J) While_loop_filter

K) Capture

L) Portmap_filter

A) Condition_token

This function is responsible for tokenization of **if block conditions** with relation to children

Attributes:

condition

type

statements

Functions:

1) _init_

This function is the constructor for **Condition_token class**

It's parameters are **self** & **type** & **condition** , **default value of the parameter = None**

2) add_statement

This function is responsible for adding child statement to the parent condition

It returns **void**

B) Match_Case_Token

This function is responsible for tokenization of **match case conditions** with relation to children

Attributes:

self

type

parameter

choice

Functions:

1) _init_

This function is the constructor for **Match_Case_Token class**

It's parameters are **self** & **type** , **the default parameter & choice value = None**

2) add_statement

This function is responsible for adding child statement to the parent match case condition

It's parameters are **self** & **statement**

It returns **void**

C) While_loop_token

This function is responsible for tokenization of **while loop conditions** with relation to children

Attributes:

condition

statements

Functions:

1) _init_

This function is the constructor for **While_loop_token class**

It's parameters are **self** & **condition**

2) add_statement

This function is responsible for adding child statement to the parent while loop condition

It's parameters are **self** & **statement**

it returns **void**

D) For_loop_token

This function is responsible for tokenization of **for loop condition** with relation to children

Attributes:

self

from_var

to_var

statements

Functions:

1) _init_

This function is the constructor for **For_loop_token class**

It's parameters are **self** & **parameter** & **to_var** & **from_var**

2) add_statement

This function is responsible for adding child statement to the parent for loop condition

It's parameters are **self** & **statement**

It returns **void**

E) **Statement_filter**

This function is responsible for handling specific Python code **statements** , store and parse them to VHDL code

Attributes:

line

Functions:

1) _init_

This function is the constructor for **Statement_filter class**

It's **parameters** are line & **self**

2) parse

This function is responsible for parsing statements into VHDL code

It's parameter is **self**

It returns **str: VHDL code**

F) Process_filter

This function is responsible for handling more complex constructs .It identifies Python functions decorated as **processes** and tokenizes their contents.

Attributes:

lines

process_regex_exp

sensitivity_list

tokens

Functions:

1) _init_

   This function the constructor for **Process_filter class**

   It's parameters are **self** & **lines**

2) tokenize

   This function is responsible for tokenization of given lines

   It's parameter is **self**

   It returns **void**

3) parse

   This function is responsible for parsing tokens into VHDL code

   It's parameter is **self**

   It returns **str: VHDL code**

4) is_process

   This function is responsible for asking whether if the given line is a **process** or not

   It's parameter is **line**

   It returns **boolean :**

G) If_condition_filter

This function is responsible for handling more complex constructs . It tokenizes **if block** into small statements the rearrange and parse them into VHDL code

Attributes:

lines

if_regex_exp

elif_regex_exp

else_regex_exp

tokens

Functions:

1) _init_

   This function is the constructor for **If_condition_filter class**

   It's parameters are **self** & **line**

2) tokenize

   This function is responsible for tokenization given if condition into tokens

   It's parameter is **self**

   It returns **void**

3) parse

   This function is responsible for converting tokens into VHDL code

   It's parameter is **self**

   It returns **str: VHDL code**

4) is_if

   This function is responsible for checking if the given line is if condition or not

   It's parameter is **line**

   It returns **boolean:**

H) Match_Case_Condition_Filter

This function is responsible for handling more complex constructs . It tokenizes **match case block** into tokens then rearrange and parse them into VHDL code

Attributes:

lines

match_case_regex_exp

case_regex_exp

tokens

Functions:

1) _init_

This function is the constructor for **Match_Case_Condition_Filter class**

It's parameters are **self** & **lines**

2) tokenize

This function is responsible for tokenization given lines into tokens

It's parameter is **self**

It returns **void**

3) parse

This function is responsible for converting tokens into VHDL code

It's parameter is **self**

It returns **str: VHDL code**

4) is_match_case

This function is responsible for checking if the given line is match case condition or not

It's parameter is **line**

It returns **boolean:**

I) For_loop_filter

This function is responsible for handling more complex constructs . It tokenizes **for loop block** into tokens then rearrange and parse them into VHDL code

Attributes:

lines

for_regex_exp

tokens

Functions:

1) _init_

   This function is the constructor for **For_loop_filter class**

   It's parameters are **lines** & **self**

2) tokenize

   This function is responsible for tokenization given lines into tokens

   It's parameter is **self**

   It returns **void**

3) parse

   This function is responsible for Converting tokens into VHDL code

   It's parameter is **self**

   It returns **str: VHDL code**

4) is_for

   This function is responsible for Checking if the given line is **for loop** or not

   It's parameter is **line**

   It returns **boolean:**

J) While_loop_filter

This function is responsible for handling more complex constructs . It tokenizes **while loop** block into tokens then rearrange and parse them into VHDL code

Attributes:

lines

while_regex_exp

tokens

Functions:

1) _init_

   This function is the Constructor for **While_loop_filter class**

   It’s parameters are **self** & **lines**

2) tokenize

   This function is responsible for Tokenization of given lines into tokens

   It’s parameter is **self**

   It returns **void**

3) parse

   This function is responsible for converting tokens into VHDL code

   It’s parameters is **self**

   It returns **str: VHDL code**

4) is_while

   This function is responsible for checking if the given line is **while loop** or not

   It’s parameter is **line**

   It returns **boolean:**

K) Capture

Takes Python code as a block. and categorizes them based on recognized structures (if conditions, loops, etc.) then calls filter classes then passes the recognized structures into them and handles the parsing of different code structures

Attributes:

lines

pos

current_lines

tokens

Functions:

1) _init_

   This function is the constructor for **Capture class**. It takes lines of Python code as input and initializes the parsing process

   It's parameters are **self** & **lines**

2) tokenize

   This function is responsible for Iterating through each line of code and categorizes them into respective tokens and send them to filters

   It's parameter is **self**

3) advance

   This function is responsible for advancing to the next line of the text

   It's parameter is **self**

4) get_leading_white_space

   This function is responsible for calculating the number of white spaces behind lines

   It's parameters are **self** & **line**

   It returns **int: number of white spaces**

5) is_child

   This function is responsible for checking relation between 1st parameter and 2nd parameter

   It's parameters are **self** & **parent_line** & **child_line**

It returns **boolean: True** if the 1st parameter is the parent of the 2nd parameter (Child line) , **False** if the 1st parameter is **not** the parent of the 2nd parameter (Child line)

6) capture_if

This function is responsible for capturing if structure lines then returning it as a list

It’s parameter is **self**

It returns **list: if structures**

7) capture_match_case

This function is responsible for capturing match case structure lines then returning it as a list

It’s parameter is **self**

It returns **list: match case structures**

8) capture_for

This function is responsible for capturing for loop structures lines then returning it as a list

It’s parameter is **self**

It returns **list: for loop structures**

9) capture_while

This function is responsible for capturing while loop structures lines then returning it as a list

It’s parameter is **self**

It returns **list: while loop structures**

10) capture_portmap

This function is responsible for capturing portmap structures lines then returning it as a string

It’s parameter is **self**

It returns **str: portmap structures**

11) capture_process

This function is responsible for capturing process structures lines then returning it as a list

It’s parameter is **self**

It returns **list: process structures**

12) get_tokens

It’s parameter is **self**

It returns **list: tokens**

13) parse

This function is responsible for parsing filter tokens

It’s parameter is **self**

It returns **str: VHDL code**

L) PortMap_filter

Handles more complex constructs . It tokenizes while port map block into tokens then rearrange and parse them into VHDL code

Attributes:

line

map_list

var

component_name

Functions:

1) portmap_regx_exp

It’s parameters are **self** & **component_name**

It returns **str: portmap regex exp**

2) tokenize

This function is responsible for tokenization given into tokens

It’s parameter is **self**

3) parse

This function is responsible for Parse tokens into VHDL code

It’s parameter is **self**

It returns **str: VHDL code**

4) is_portmap

This function is responsible for asking if the given line is portmap or not

It’s parameter is self

It returns **boolean:**

V. Tokens

Overview :

Defined various tokens within different categories like logic, bitwise, arithmetic, relational operators, assignment, and additional tokens. Each token is represented by an instance of the Token class imported from the lexer file

Classes :

A) Logic Tokens
B) Bitwise Tokens
C) Arithmetic Tokens

D)Relational Operators Tokens

E) Assignment Tokens

F) Additional Tokens

A) Logic Tokens

Contains:

1) and_tok
2) not_tok
3) or_tok
4) xor_tok
5) nand_tok
6) nor_tok
7) xor_tok

B) Bitwise Tokens

Contains:

1) srl_tok
2) sll_tok
3) sra_tok

4) sla_tok

C) Arithmetic Tokens

Contains:

1) plus_tok
2) minus_tok
3) mult_tok
4) div_tok
5) mod_tok
6) exp_tok

D) Relational Operators Tokens

Contains:

1) equalto_tok
2) notequalto_tok
3) lessthan_tok
4) greaterthan_tok
5) lessthanorequal_tok
6) greaterthanorequal_tok

E) Assignment Token

Contains:

ass_tok

F) Additional Tokens

Contains:

1) left_square_bracket_tok
2) right_square_bracket_tok
3) left_parentheses_tok
4) right_parentheses_tok
5) single_quotation_tok
6) double_quotation_tok

7) Coma_tok

VI. Initialization

Store Components and Files in a list. Initialization is responsible for initializing these lists every time the user imports the PythontoVHDLConverter library

Classes :

- components
- files

VII. Components

Built in library contains common components that the user can import and use in their project

Contents:

- decoder 2x4
- decoder 3x8
- demux 1x4
- demux 1x8
- DFlipFlop
- encoder4to2
- encoder8to3
- FullAdder
- HalfAdder

- JKFlipFlop
- mux4x1
- mux8x1
- TFlipFlop