ringwormHDL

1

Generated by Doxygen 1.8.20

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 Class Documentation

2.1 generate.generate Class Reference

Public Member Functions

- None __init__ (self)
- str RO_not (self, int n, str in_p, str out_p, writeToFile newFile)
- str RO_nand (self, int n, str in_p, str out_p, writeToFile newFile)
- str RO_nor (self, int n, str in_p, str out_p, writeToFile newFile)
- str onChipMem (self, str d, str wren, str clk, str write_addr, str read_addr, str q, writeToFile newFile)

Public Attributes

- **modules**
- **inputs**
- **outputs**

2.1.1 Detailed Description

```
generate - class for generating test structures (ring oscillators)
```

2.1.2 Constructor & Destructor Documentation

2.1.2.1 __init__() None generate.generate.__init__ (

self)

```
__init__: stores names of all generated modules so that they can be connected in the top module
self.modules - list of module names (List[str])
self.inputs - list of list of module input names (List[List[str]]), where the index = module number
self.outputs - list of list of module output names (List[List[str]]), where the index = module number
```

2.1.3 Member Function Documentation

2.1.3.1 onChipMem() str generate.generate.onChipMem (

self,
str *d*,
str *wren*,
str *clk*,
str *write_addr*,
str *read_addr*,
str *q*,
writeToFile *newFile*)

```
onChipMem: Generates M10K on chip memory (for DE1-SoC, may need to be tweaked depending on which FPGA board is us
returns module name
d - input data signal (str)
wren - write enable signal (str)
clk - clock signal (str)
write_addr - write address (str)
read_addr - read address (str)
q - output (str)
newFile - writeToFile object
```

2.1.3.2 RO_nand() `str generate.generate.RO_nand (`

```
 self,
int n,
str in_p,
str out_p,
writeToFile newFile )
```

```
RO_nand: Generates a ring oscillator circuit of size n with NAND gates, where n is a positive odd integer.
Writes module to file specified, returns module name
n - number of stages in RO (int)
in_p - name of input enable port (str)
out_p - name of output clock port (str)
newFile - writeToFile object
```

2.1.3.3 RO_nor() `str generate.generate.RO_nor (`

```
 self,
int n,
str in_p,
str out_p,
writeToFile newFile )
```

```
RO_nor: Generates a ring oscillator circuit of size n with NOR gates, where n is a positive odd integer.
Writes module to file specified, returns module name
n - number of stages in RO (int)
in_p - name of input enable port (str)
out_p - name of output clock port (str)
newFile - writeToFile object
```

2.1.3.4 RO_not() `str generate.generate.RO_not (`

```
 self,
int n,
str in_p,
str out_p,
writeToFile newFile )
```

```
RO_not: Generates a ring oscillator circuit of size n with NOT gates, where n is a positive odd integer.
Writes module to file specified, returns module name
n - number of stages in RO (int)
in_p - name of input enable port (str)
newFile - writeToFile object
```

The documentation for this class was generated from the following file:

- C:/Users/Ivan Cheng/Desktop/git/ringwormHDL/ringwormHDL/generate.py

2.2 verilogModule.verilogModule Class Reference

Public Member Functions

- None __init__ (self, str moduleName)
- str parameter (self, str newParam, int i)
- str port (self, str newPort, str dir, int w=1)
- str logic (self, str newLogic, int width=1, str synthParam="")
- str assign (self, str l, str r, int lw=0, int lwd=0, int rw=0, int rwd=0, str synthParam="")
- str notGate (self, str a, str b, str synthParam="")
- str andGate (self, str o, str a, str b, str synthParam="")
- str nandGate (self, str o, str a, str b, str synthParam="")
- str norGate (self, str o, str a, str b, str synthParam="")
- int alwaysSequential (self, dict signals={})
- int nbAssign (self, int i, str l, str r)
- int ifStatement (self, int i, str cond, str cmd)
- int elseStatement (self, int i, str cmd)

Public Attributes

- **moduleName**
- **parameters**
- **ports**
- **sequential**
- **logics**
- **combinational**

2.2.1 Detailed Description

```
verilogModule - class for generating SystemVerilog modules
```

2.2.2 Constructor & Destructor Documentation

2.2.2.1 __init__()

```
None verilogModule.verilogModule.__init__ (
       self,
      str moduleName )
```

```
__init__: takes string "name" and instantiates a Verilog/SystemVerilog module with that name
*dicts are ordered in python 3.6+, use OrderedDict if older version*
self.parameters - a dict of all the module parameters
self.ports - a dict of all the module ports, where 0 = input and 1 = output
self.sequential - a list of llists of all sequential logic, where the 0th element in each list of lists is the fi
self.logic - a dict of all logic defines (wire/regs), where the order is based on input order
self.combinational - a dict of all combinational logic, where all are syntehsized with the tag /*synthesis keep*/
self.generate - a dict of all generate loops, which can contail sequential or combinational logic (this might nee
```

2.2.3 Member Function Documentation

2.2.3.1 alwaysSequential()

```
int verilogModule.verilogModule.alwaysSequential (
        self,
        dict signals = {} )
```

```
alwaysSequential: creates a sequential always block with input edge conditions
returns index in sequential list that all commands for this always block should be input in
signals: name of signals and "p/n" for posedge/negedge, leave empty for * wild card (dict)
```

2.2.3.2 andGate()

```
str verilogModule.verilogModule.andGate (
        self,
        str o,
        str a,
        str b,
        str synthParam = "" )
```

```
andGate: creates an AND gate with synthesis parameter
returns dict key
o - output port (str)
a - input port 1 (str)
b - input port 2 (str)
synthParam - synthesis parameter (str)
```

2.2.3.3 assign()

```
str verilogModule.verilogModule.assign (
        self,
        str l,
        str r,
        int lw = 0,
        int lwd = 0,
        int rw = 0,
        int rwd = 0,
        str synthParam = "" )
```

```
assign: takes lhs and rhs values, along with their bit widths, and creates an assign statement
default to 1 bit wire, can specify width manually
returns dict key /*synthesis keep*/, or "error" if not valid
l - lh WIRE ONLY (str)
r - rh wire/reg (str)
lw - left bit [_:0] (int)
lwd - right bit [7:_] (int)
rw - left bit [_:0] (int)
rwd - right bit [7:_] (int)
synthParam - synthesis parameter (str)
```

2.2.3.4 elseStatement() `int verilogModule.verilogModule.elseStatement (`

```
 self,
int i,
str cmd )
```

```
elseStatement: generates an if statement for use in a sequential always block
returns index of command in list of list
i - index of sequential always block (int)
cmd - command to execute (str)
```

2.2.3.5 ifStatement() `int verilogModule.verilogModule.ifStatement (`

```
 self,
int i,
str cond,
str cmd )
```

```
ifStatement: generates an if statement for use in a sequential always block
returns index of command in list of list
i - index of sequential always block (int)
cond - condition for if statement (str)
cmd - command to execute if condition is true (str)
```

2.2.3.6 logic() `str verilogModule.verilogModule.logic (`

```
 self,
str newLogic,
int width = 1,
str synthParam = "" )
```

```
logic: use to create a logic (wire/reg) "w" bits wide
returns dict key, "error" if could not add
newLogic - name (str)
width - width (int)
synthParam - synthesis parameter (str)
```

2.2.3.7 nandGate() `str verilogModule.verilogModule.nandGate (`

```
 self,
str o,
str a,
str b,
str synthParam = "" )
```

```
nandGate: creates a NAND gate with synthesis parameter
returns dict key
o - output port (str)
a - input port 1 (str)
b - input port 2 (str)
synthParam - synthesis parameter (str)
```

2.2.3.8 nbAssign() `int verilogModule.verilogModule.nbAssign (`

```
    self,
    int i,
    str l,
    str r )
```

```
nbAssign: creates non blocking assignment statement for use in sequential always blocks
for now, default to lhs bit width == rhs bit width
returns index of command in list of list
i - index of sequential always block (int)
l - lhs (str)
r - rhs (str)
```

2.2.3.9 norGate() `str verilogModule.verilogModule.norGate (`

```
    self,
    str o,
    str a,
    str b,
    str synthParam = "" )
```

```
norGate: creates a NOR gate with synthesis parameter
returns dict key
o - output port (str)
a - input port 1 (str)
b - input port 2 (str)
synthParam - synthesis parameter (str)
```

2.2.3.10 notGate() `str verilogModule.verilogModule.notGate (`

```
    self,
    str a,
    str b,
    str synthParam = "" )
```

```
notGate: creates a NOT gate with synthesis parameter
returns dict key
a - input port (str)
b - output port (str)
synthParam - synthesis parameter (str)
```

2.2.3.11 parameter() `str verilogModule.verilogModule.parameter (`

```
    self,
    str newParam,
    int i )
```

```
parameter: takes parameter name, value, and adds them to the module
returns dict key, "error" if could not add
newParam - name of parameter (str)
i - value of parameter (int)
```

2.2.3.12 port()

```
str verilogModule.verilogModule.port (
    self,
    str newPort,
    str dir,
    int w = 1 )
```

```
ports: takes port name, direction, and adds them to the module
returns dict key, "error" if could not add
p - name of port (str)
dir - direction of port, use "i" to designate input, and "o" to designate output (str)
w - width of port (int)
```

The documentation for this class was generated from the following file:

- C:/Users/Ivan Cheng/Desktop/git/ringwormHDL/ringwormHDL/verilogModule.py

2.3 writeToFile.writeToFile Class Reference

Public Member Functions

- None __init__ (self, str fileName)
- None writeSubModule (self, verilogModule module)
- None writeTopModule (self, verilogModule modules)

Public Attributes

- **fileName**

2.3.1 Detailed Description

```
writeToFile - class for creating/writing the generated SystemVerilog to a .sv file
```

2.3.2 Constructor & Destructor Documentation

2.3.2.1 __init__()

```
None writeToFile.writeToFile.__init__ (
    self,
    str fileName )
```

```
initialization: creates new file with specified file name
fileName: name of SystemVerilog file (str)
```

2.3.3 Member Function Documentation

2.3.3.1 writeSubModule()

```
None writeToFile.writeToFile.writeSubModule (
        self,
        verilogModule module )
```

```
writeSubModule: writes the module to the file, takes file name from writeToFile instance
generation format:
module <name>
    #(<parameters>)
    (<ports>);
    <logic decl.>;
    <combinational logic>;
    <sequential logic>;
endmodule
module - verilogModule object
```

2.3.3.2 writeTopModule()

```
None writeToFile.writeToFile.writeTopModule (
        self,
        verilogModule modules )
```

```
writeTopModule: writes generated modules into a top_module for synthesis
generation format:
module <name>
    (<ports>);
    <module instantiations>;
endmodule
modules - generate object, from which we take the lists of module names/ports
```

The documentation for this class was generated from the following file:

- C:/Users/Ivan Cheng/Desktop/git/ringwormHDL/ringwormHDL/writeToFile.py

Index