
	Academic Year:	2022/2023	Term:	second term	
	Course Code:	Eective4	Course Title:	EDA	

**Cairo University**  
**Faculty of Engineering**  
**Electronics and Communications Engineering Department –**  
**4<sup>th</sup> Year**

**AUTOMATIC TESTBENCH GENERATOR**  
**PROJECT**

Name	Sec	BN
احمد سامح عبدالصبور منصور	1	4
ايهاب مصطفى فرغلي صالح	1	46
بسنت عاطف جاد الكريم	1	47
عبدالرحمن احمد عواد محمود	2	43
مصطفى خالد فاروق عبدالشافى	4	18
منى منصور امين محمد	4	30

## 1- Overview

In this project we designed an Automatic Test Bench generator tool. The input is a Verilog DUT, the results are Testbench that initialize variables using initial block, generate clock, instantiate the input DUT into the Testbench, and use always/initial with appropriate time delays to create test stimulus.

## 2- Verilog Code Parser

The tool is able to parse Verilog code of the DUT to extract Input Ports, Variables, If Condition Expression, Case Conditions, ..

- o Verilog Module.
- o Input, Output Ports.
- o Registers, Wires.
- o Continuous Assignments.
- o Always Block (Combinational, Clock Based).
- o If , Case
- o Non-Blocking Assignment.
- o Logical Operator.

### 2.1 Module name, Input, Output Ports, Parameters

- The output of the parser for extracting all of these is shown in figure (2.1.1).
- It describes the module name, parameters, output signals which include the logic sequence of the output and the input signals which will be randomized.

```
design_info = {
  'module_name': 'ALU',
  'parameters': '{WIDTH: 16}',
  'input_ports': '{A: '[WIDTH-1:0]', 'B: '[WIDTH-1:0]', 'ALU_FUN': '[1:0]'}',
  'output_ports': '{Arith_Flag: '', 'Logic_Flag': '', 'ALU_OUT': '[WIDTH-1:0]'}',
  'input_signals': [
    {'name': 'A', 'range': (0, 65535)},
    {'name': 'B', 'range': (0, 65535)},
    {'name': 'ALU_FUN', 'range': (0, 3)},
  ],
  'output_signals': [
    {'name': 'y', 'logic': '1 if a && b 2 if a else 3'},
    {'name': 'ALU_OUT_Comb', 'logic': "A+B if (ALU_FUN) == 2'b00 else A-B if (ALU_FUN) == 2'b01 else A & B if (ALU_FUN) == 2'b10 else A | B if (ALU_FUN) == 2'b11 else 16'b0"},
  ],
}
```

figure(2.1.1)

### 2.2 Always Block (combinational, Clock based), Non-Blocking Assignment, If and Case

- The output of the parser for extracting all of these is shown in figure (2.2.1).
- It describes the clock edge (positive or negative), RST signal, the if conditions, the case statement and the two types of always blocks.

```

always_1:
  sequential: True
  sensitivity:
    {'signal_name': 'clk', 'edge_type': 'posedge'}
    {'signal_name': 'RST', 'edge_type': 'negedge'}
  if:
    {'condition': '(!RST)', 'statements': ['y <= 0;']}
  else if:
    {'condition': '(a && b)', 'statements': ['y <= 1;']}
    {'condition': '(a)', 'statements': ['y <= 2;']}
  else:
    {'statements': ['y <= 3;']}
  case statement:
always_2:
  sequential: False
  if:
  else if:
  else:
  case statement:
    {'(ALU_FUN)': {'"2'b00": 'A+B', '"2'b01": 'A-B', '"2'b10": 'A & B', '"2'b11": 'A | B', 'default': "16'b0"}}}

```

figure(2.2.1)

## 2.3 Continuous Assignments and Logical Operators

- The output of the parser for extracting the assign statement and the logical operators is shown in figure (2.3.1).

```

],
'assign statement': '{'C': 'A & B', 'D': 'A | B'}',

```

figure(2.3.1)

## 3- Parser Testing For 3 Verilog Cases

### 3.1 Parser Results for Verilog Case1

- This case is ALU which implements some ALU functions.

```

design_info = {
  'module_name': 'ALU',
  'parameters': {'WIDTH': 16},
  'input_ports': {'A': '[WIDTH-1:0]', 'B': '[WIDTH-1:0]', 'ALU_FUN': '[1:0]'},
  'output_ports': {'ALU_OUT': '[WIDTH-1:0]'},
  'input_signals': [
    {'name': 'A', 'range': (0, 65535)},
    {'name': 'B', 'range': (0, 65535)},
    {'name': 'ALU_FUN', 'range': (0, 3)},
  ],
  'output_signals': [
    {'name': 'ALU_OUT', 'logic': "A+B if (ALU_FUN) == 2'b00 else A-B if (ALU_FUN) == 2'b01 else A & B if (ALU_FUN) == 2'b10 else A | B if (ALU_FUN) == 2'b11 else 16'b0"},
  ],
}

```

## 3.2 Parser Results for Verilog Case2

- This case is a MUX.

```
design_info = {
  'module_name': 'MUX',
  'parameters': '{'WIDTH': 8}',
  'input_ports': '{'A': '[WIDTH-1:0]', 'B': '[WIDTH-1:0]', 'sel': '', 'add': '', 'sub': '', 'in_data': ''
}',
  'output_ports': '{'result': '[WIDTH:0]}'',
  'clock': '{'signal_name': 'CLK', 'edge_type': 'posedge'}',
  'reset': '{'signal_name': 'RST', 'edge_type': 'negedge'}',
  'input_signals': [
    {'name': 'A', 'range': (0, 255)},
    {'name': 'B', 'range': (0, 255)},
    {'name': 'sel', 'range': (0, 1)},
    {'name': 'add', 'range': (0, 1)},
    {'name': 'sub', 'range': (0, 1)},
    {'name': 'in_data', 'range': (0, 1)},
  ],
  'output_signals': [
    {'name': 'result', 'logic': 'A if in_dataB if sel A+B if add A-B if sub else 0'},
  ],
}
```

## 3.3 Parser Results for Verilog Case3

- This case is a SHIFT module.

```
design_info = {
  'module_name': 'Shift',
  'parameters': '{'WIDTH': 8}',
  'input_ports': '{'A': '[WIDTH-1:0]', 'B': '[WIDTH-1:0]}'',
  'output_ports': '{'OUT1': '[WIDTH:0]', 'OUT2': '[WIDTH:0]', 'OUT3': '[WIDTH:0]}'',
  'clock': '{'signal_name': 'CLK', 'edge_type': 'posedge'}',
  'reset': '{'signal_name': 'RST', 'edge_type': 'negedge'}',
  'input_signals': [
    {'name': 'A', 'range': (0, 255)},
    {'name': 'B', 'range': (0, 255)},
  ],
  'output_signals': [
    {'name': 'OUT1', 'logic': 'A>>2'},
    {'name': 'OUT2', 'logic': 'B2'},
    {'name': 'OUT3', 'logic': 'A+B'},
  ],
}
```

## 4- Stimulus Generator

The code mainly consists of 2 functions , (Test Cases Generator , Evaluate).

### 4.1 Test Cases Generator Function

This function takes 2 inputs : design info (DUT parser) , number of test cases.

- Design info (DUT parser) is the information of the DUT which is the output of the parser.
- Number of test cases can be any number 10,100,1000, etc..

The function mainly loops on the number of test cases and it generates random values for the inputs within their ranges and stores these values in a list or dict. (Ex. If x is a 4-bit input , then x can take any value from 0 → 15 which are the random values that will be generated and stored in the list).

### 4.2 Evaluate Function

This function takes 2 inputs : the inputs' random values generated from test cases generator function, the expression of the output. (Ex. If the output is an ALU output, then the expression may be addition , subtraction , AND or XOR..).

The function mainly takes inputs' random values and the expression of the output then, evaluates the expected value of the output according to the random value of the input and the expression of the output. (Ex. If expression of the output C is addition and inputs A, B have random values of 3,5 respectively, then evaluate function will return the expected value of C which is 8).

The output of the Evaluate function is the expected value of the output which we compare with the output of the DUT. The random expected values of the output are then stored in a list or dict.

## 5- Testbench Writer

This function generates the testbench. It writes the word “timescale” , module name, initial block for initialization, clock generation, instantiation, tests generator and monitor.

This function writes these all in a file.

## 6- Testbench Generator Testing For 3 Verilog Cases

### 6.1 Results for Verilog Case1

This case is ALU which implements some ALU functions described in figure (6.1.1)

```
case(ALU_FUN)
2'b00:  ALU_OUT <= A+B    ;
2'b01:  ALU_OUT <= A-B    ;
2'b10:  ALU_OUT <= A & B  ;
2'b11:  ALU_OUT <= A | B  ;
default: ALU_OUT <= 16'b0 ;
endcase
```

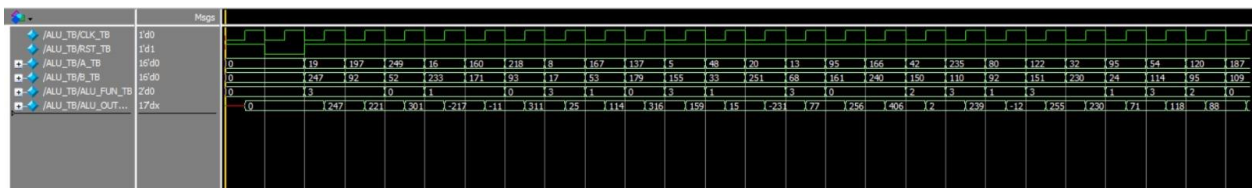
figure(6.1.1)

The dict of the design is shown in figure (6.1.2).

```
# Example usage:
design_info = {
    "module_name": "ALU",
    "input_ports": {"A": "[WIDTH-1:0]", "B": "[WIDTH-1:0]", "ALU_FUN": "[1:0]"},
    "output_ports": {"ALU_OUT": "[WIDTH:0]"},
    "parameter": {"WIDTH": 16},
    "clock": "CLK",
    "reset": ("RST", "negedge"),
    "input_signals": [
        {'name': 'A', 'range': (0, 255)},
        {'name': 'B', 'range': (0, 255)},
        {'name': 'ALU_FUN', 'range': (0, 3)},
    ],
    "output_signals": [
        {'name': 'ALU_OUT', 'logic': 'A+B if ALU_FUN == 0 else A-B if ALU_FUN ==1 else A&B if ALU_FUN==2 else A|B if ALU_FUN==3 else 0'},
    ]
}
```

figure(6.1.2)

The waveform of the result is shown in figure (6.1.3). which describes the output of the ALU according to the random values of A,B and the ALU function.



figure(6.1.3)

## 6.2 Results for Verilog Case2

This case is a MUX is described in figure (6.2.1)

```
always @(posedge CLK or negedge RST)
begin
    if(!RST) begin
        result <= 'b0;
    end
    if(in_data) begin
        result <= A;
    end
    else if (sel) begin
        result <= B;
    end
    else if (add) begin
        result <= A+B;
    end
    else if (sub) begin
        result <= A-B;
    end
    else begin
        result <= 0;
    end
end
end
```

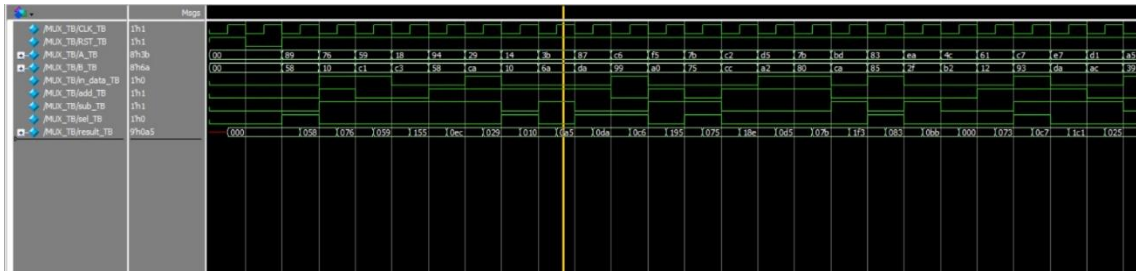
figure(6.2.1)

The dict of the design is shown in figure (6.2.2).

```
# Example usage:
design_info = {
    "module_name": "MUX",
    "input_ports": {"A": "[WIDTH-1:0]", "B": "[WIDTH-1:0]", "in_data": "", "add": "", "sub": "", "sel": ""},
    "output_ports": {"result": "[WIDTH:0]"},
    "parameter": {"WIDTH": 8},
    "clock": "CLK",
    "reset": ("RST", "negedge"),
    "input_signals": [
        {'name': 'A', 'range': (0, 255)},
        {'name': 'B', 'range': (0, 255)},
        {'name': 'in_data', 'range': (0, 1)},
        {'name': 'add', 'range': (0, 1)},
        {'name': 'sub', 'range': (0, 1)},
        {'name': 'sel', 'range': (0, 1)},
    ],
    "output_signals": [
        {'name': 'result', 'logic': 'A if in_data==1 else B if sel==1 else A+B if add==1 else A-B if sub==1 else 0'},
    ]
}
```

figure(6.2.2)

The waveform of the result is shown in figure (6.2.3). which describes the output of the MUX according to the random values of A,B and the Selection lines.



figure(6.2.3)

## 6.3 Results for Verilog Case3

This case is SHIFT which is described in figure (6.3.1)

```
always @(posedge CLK or negedge RST)
begin
    if(!RST) begin
        OUT1 <= 'b0;
    end
    else begin
        OUT1 <= A>>2;
    end
end
always @(posedge CLK or negedge RST)
begin
    if(!RST) begin
        OUT2 <= 'b0;
    end
    else begin
        OUT2 <= B<<2;
    end
end
end
```

figure(6.3.1)

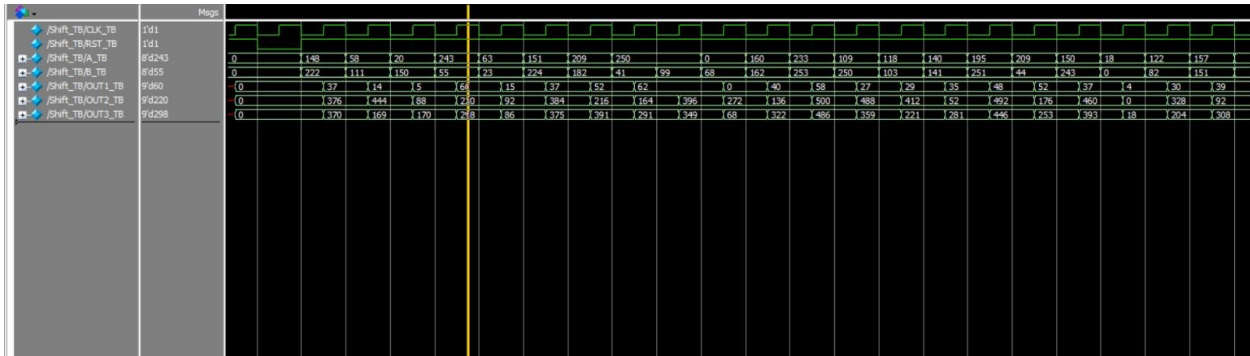
The dict of the design is shown in figure (6.3.2).

```
design_info = {
    "module_name": "Shift",
    "input_ports": {"A": "[WIDTH-1:0]", "B": "[WIDTH-1:0]"},
    "output_ports": {"OUT1": "[WIDTH:0]", "OUT2": "[WIDTH:0]", "OUT3": "[WIDTH:0]"},
    "parameter": {"WIDTH": 8},
    "clock": "CLK",
    "reset": ("RST", "negedge"),
    'input_signals': [
        {'name': 'A', 'range': (0, 255)},
        {'name': 'B', 'range': (0, 255)},
    ],
    'output_signals': [
        {'name': 'OUT1', 'logic': 'A>>2'},
        {'name': 'OUT2', 'logic': 'B<<2'},
        {'name': 'OUT3', 'logic': 'A+B'},
    ],
}
```

figure(6.3.2)



The waveform of the result is shown in figure (6.3.3). which describes the output of the SHIFT.



figure(6.3.3)

## 7- GUI Python

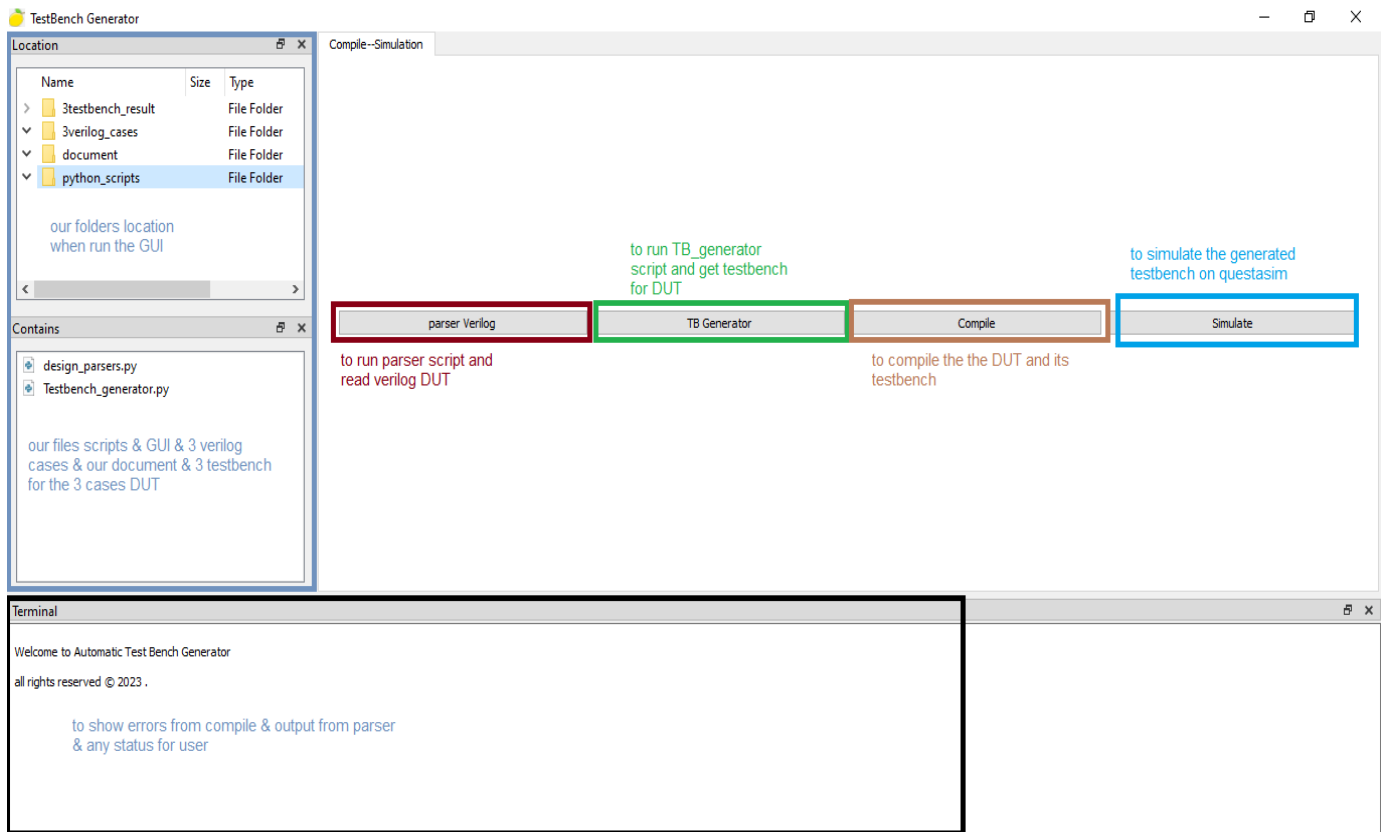


Figure6(testbench generator tool)

## 7- Appendix

### **The recommended design format to be used in the tool**

```
module Shift #( parameter WIDTH = 8)
(
    input      CLK,
    input wire  RST,
    input  [WIDTH-1:0] A,    // if the input or output is parametrized,
    input  [WIDTH-1:0] B,    // its width must be written in this way [WIDTH-1:0]
    output reg [WIDTH-1:0] OUT1,
    output reg [WIDTH-1:0] OUT2,
    output reg [WIDTH-1:0] OUT3
);

always @(posedge CLK or negedge RST)
begin
    if(!RST) begin        // begin have to be with the same line with condition line
        OUT1 <= 'b0;      // if, case statements should include 1 output only
    end
    else begin            // in if_else statement, there must be (else)
        OUT1 <= A>>2;
    end
end
always @(posedge CLK or negedge RST)
begin
    if(!RST) begin
        OUT2 <= 'b0;
    end
    else begin
        OUT2 <= B<<2;
    end
end

always @(posedge CLK or negedge RST)
begin
    if(!RST) begin
        OUT3 <= 'b0;
    end
    else begin
        OUT3 <= A+B;
    end
end
endmodule
```