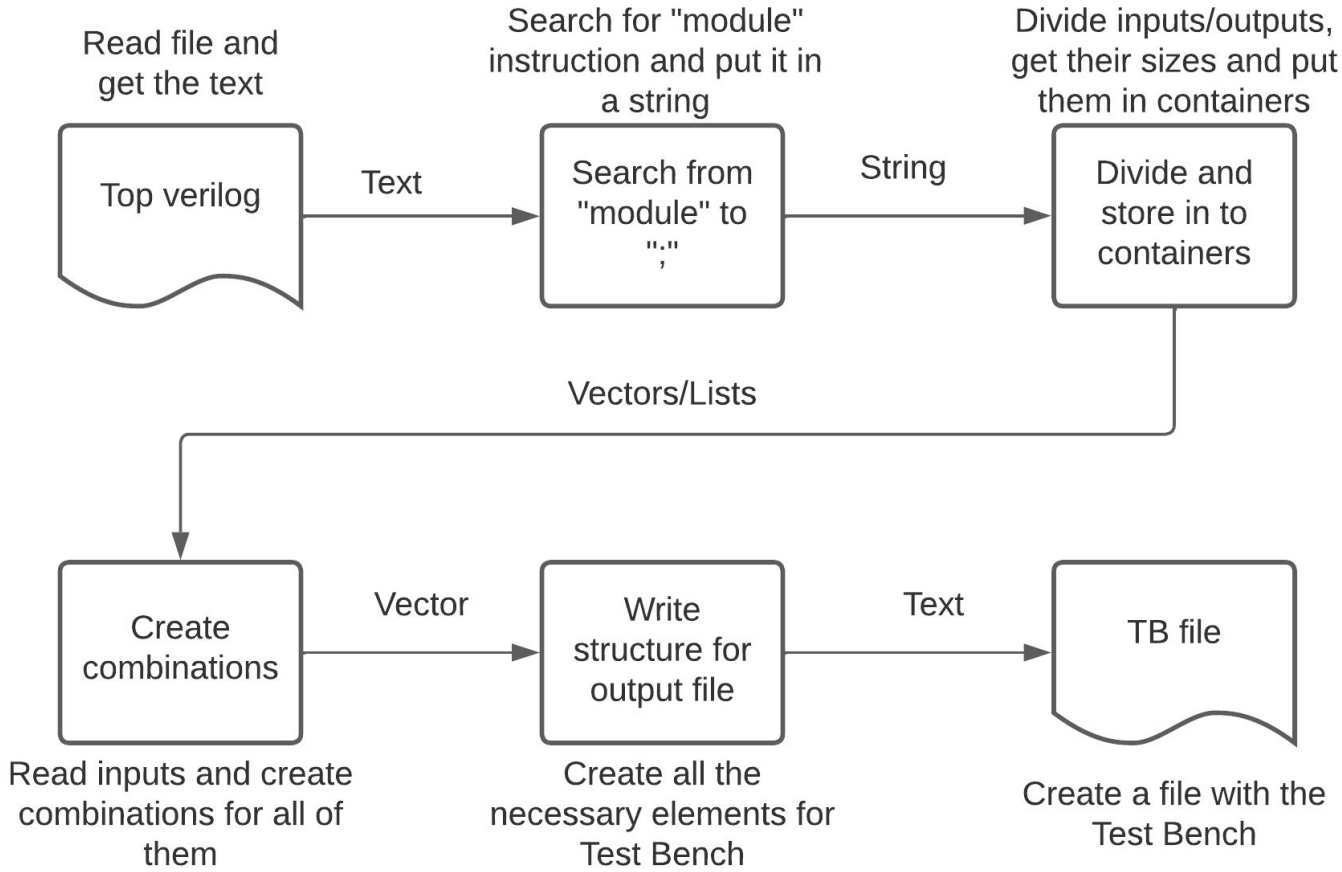
Test Bench Project

Verificación Pre-silicio Primavera 2022

Josue Isaias Gomez Cosme
Bruno Hernández López
Jose Alberto Gomez Diaz
Christian Aaron Ortega Blanco


Read file and get the text
Top verilog
Text
Search for "module" instruction and put it in a string
Search from "module" to ";"
String
Divide inputs/outputs, get their sizes and put them in containers
Divide and store in to containers
Vectors/Lists
Create combinations
Read inputs and create combinations for all of them
Vector
Write structure for output file
Create all the necessary elements for Test Bench
Text
TB file
Create a file with the Test Bench

Abstract

1a

Functions used;
match():
search():
sub():
replace():
rstrip():
find():
append():
join():
Libraries;
re,array,match.

2b

Functions used;
open():
getline():
find():
erase():
substr():
find():
pushback():
at():
Libraries;iostream,
fstream,string,vect
or,math.h.

3c

Functions used;
regex.search():
getline():
regex_macth():
erase():
substr():
find():
pushback():
at():
Libraries;iostream,
fstream,string,vect
or,math.h,regex.

Structure

Read file and delete the unnecessary text

```
while (!archivo.eof()) {
    getline(archivo, texto);
    if (texto.find("//")!=string::npos){        //Comments inside the design are removed
        int t1 = texto.length();                   //
        int ecom = texto.find("//");              //
        texto.erase(ecom,t1-ecom);
    }
    if (texto.find('`')!=string::npos){         //The libraries inside the design are removed
        int t1 = texto.length();                   //
        int ecom = texto.find('`');               //
        texto.erase(ecom,t1-ecom);
    }
    if (texto.find('#')!=string::npos){         //Parameters inside the design are removed
        int t1 = texto.length();                   //
        int ecom = texto.find('#');               //
        texto.erase(ecom,t1-ecom);
    }
```

Extract module statement

```
string text = Prueba;
string temp=Prueba.substr(Prueba.find("module"),Prueba.find('('));
temp.erase(0,Prueba.find("module")+6);
module_name.append(temp);
int a = Prueba.find("(");
int b = a + 1;
int c = Prueba.find(")", a);

for (int i = a; i <= c; i++) {
    Apend.push_back(Prueba[i]);
}
```

Containers

```
string module=lectura();

vector<string> inputs;  //inputs vector
vector<string> outputs; //outputs vector

vector<int> bitsint;       //vector with the number of bits per input
vector<int> bitsout;        //vector with the number of bits per output

vector<string> bitrgint;       //vector with range of input buses
vector<string> bitrgout;       //vector with range of output buses
string inouts="";
```

Identify if it is input or output

```
if(instruction.find("input")!=-1){
    instruction.replace(0,instruction.find("input")+5,"reg ");//word "input" is replaced by "reg"
    inputs.push_back(instruction);//adds to array of inputs
    string temp=instruction;
    temp.replace(0,(temp.find(']')+1),"");
    if (temp.substr(0,3)=="reg")temp.replace(0,3,"");
    inouts+=temp;
    inouts+=",";
    linout=true; //flag was input
}
```

Variables without identifier

```
if(linout){ //if the last variable was input
    string temp=inputs.back();
    if(temp.find(']')!=string::npos){
        temp=temp.substr(0,temp.find(']')+1);
    }
    else{
        temp=temp.substr(0,2);
    }
    if(temp.find(']')!=-1) temp=temp.substr(0,temp.find(']')+1);
    else temp="reg ";
    inouts+=instruction;
    inouts+=",";
    temp+=" "+instruction; //concatenate variable name and its values
    inputs.push_back(temp);
}
```

Testbench structure

```
1  module FA_TB ();
2
3      reg A_TB, B_TB,Cin_TB;
4      wire S_TB,Cout_TB;
5
6    FA DUT (A_TB, B_TB,Cin_TB,S_TB,Cout_TB);
7
8  initial begin
9
10     $dumpvars; $dumpfile("dump.vcd");
11
12     A_TB <= 0;
13     B_TB <= 0;
14     Cin_TB <= 0;
15
16   $monitor("%b %b %b %b %b", A_TB,B_TB,Cin_TB, S_TB,Cout_TB);
17
18   for (int i=0; i < 8; i=i+1)
19     begin
20         {A_TB,B_TB,Cin_TB} = i;
21         #10;
22     end
23 end
24
25 endmodule
```

Testbench name

Variables to test

Variables assignation

Initialization of program and waveform generation

Inputs initialization

Values display

Inputs combinations

Final statements

inputs

Input reg [1:0] A
Input [2:0] B
Input reg C
Input D

outputs

Output W
Output X
Output Y
Output Z

bitsint

2
3
1
1

module_name

inouts

A, B, C, D, W, X, Y, Z

```
int combinaciones=0;
string input_comb;
vector<string> arr= inputs;
 for (int i=0; i<arr.size(); i++) {
    arr[i].erase(0,arr[i].find("reg")+3);
    if(arr[i].find(']')!=-1)arr[i].erase(0,arr[i].find(']')+1);
    input_comb.append(arr[i]);
    input_comb.append(",");
}
input_comb.erase(input_comb.length()-1,input_comb.length());
```

arr=inputs

input_comb

A, B, C, D

Beginning of text file generation for testbench

```
ofstream myfile;
myfile.open ("testBench.txt");
myfile << "`timescale 1ns/100ps\n";

myfile <<"module automatic_TB ();\n";
myfile << "/*        variables          */\n";

for(int i=0;i<inputs.size();i++){
  myfile << inputs[i]<<";\n";
}
  for(int i=0;i<outputs.size();i++){
  myfile <<  outputs[i]  <<";\n";
}
myfile << "\n";
myfile << module_name <<"\n";  //module name
myfile << "DUT("<< inouts <<");\n";  /// module variables
myfile << "initial begin\n";
myfile << "$dumpvars; $dumpfile(\"dump.vcd\");\n";
myfile << "\n$monitor(\"";
for (int i=0;i<inputs.size();i++) myfile << "%b ";
for (int i=0;i<outputs.size();i++) myfile << "%b ";
myfile << "\",";
myfile << inouts;
myfile << ");\n";
```

Inputs combinations logic

```
for (int i=0; i<bitsint.size(); i++) {
    combinaciones+=bitsint[i];
}
combinaciones=pow (2, combinaciones);
```

[2, 1, 2]

combinaciones= 2+1+2 = 5

combinaciones = 2^5 = 32

```
myfile << "\nfor(int i=0;i<"<<combinaciones<<";i++)begin";
myfile << "\n     {"<<input_comb<<"}=i;";
myfile << "\n      #1;";
myfile << "\nend";
```

Final testbench elements

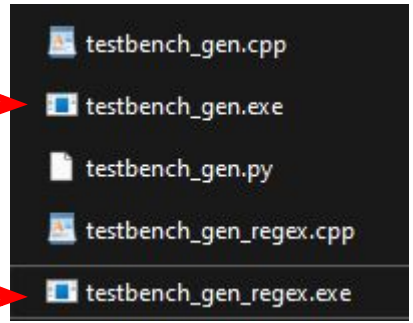
```
myfile << "$finish();\n";
myfile << "end\n";
myfile << "endmodule\n";
myfile << "/*  ----------------module end------------------       */\n";
myfile << "\n";


myfile.close();
```

Results:
compile C++ programs

compile a c++ script will create an executable file.

Automatic Generator

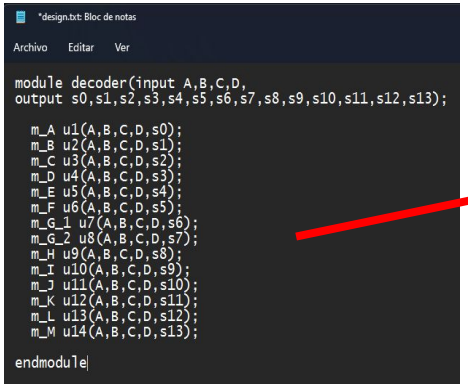
Automatic Generator
with regular expressions



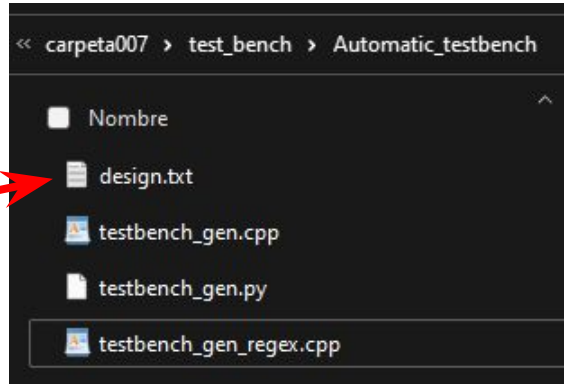
Results:

preparing the design file

to create a new Testbench file, its necessary to add a Design.txt file with the top level code in the program folder.



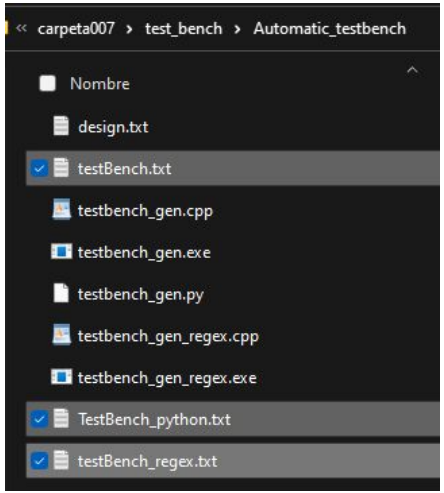
a) TopLevel example



b) Programs folder

Results:
running programs

the execution of the .exe files will return an automatic generated TestBench file of the given design.txt located on the programs folder.




```
`timescale 1ns/100ps
module automatic_TB ();
/*      variables         */
reg  A;
reg  B;
reg  C;
reg  D;
wire  s0;
wire  s1;
wire  s2;
wire  s3;
wire  s4;
wire  s5;
wire  s6;
wire  s7;
wire  s8;
wire  s9;
wire  s10;
wire  s11;
wire  s12;
wire  s13;

 decoder
DUT(  A,B,C,D,  s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13);
initial begin
$dumpvars; $dumpfile("dump.vcd");

$monitor("%b %b %b %b %b %b %b %b %b %b %b %b %b %b %b %b %b ",  A,B,C,D,  s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13);

for(int i=0;i<16;i++)begin
     {  A,  B,  C,  D}=i;
      #1;
end


$finish();
end
endmodule
/*  ---------------module end-----------------       */
```

Results:

eda result

testbench.sv

```
`timescale 1ns/100ps
module automatic_TB ();
/*        variables          */
reg  A;
reg  B;
reg  C;
reg  D;
wire  s0;
wire  s1;
wire  s2;
wire  s3;
wire  s4;
wire  s5;
wire  s6;
wire  s7;
wire  s8;
wire  s9;
wire  s10;
wire  s11;
wire  s12;
wire  s13;

 decoder
DUT(  A,B,C,D,  s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13);
initial begin
$dumpvars; $dumpfile("dump.vcd");

  $monitor("%b %b %b %b : %b %b %b %b %b %b %b %b %b %b %b %b %b %b ",  A,B,C,D,
s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13);
```

design.sv

Not_gate.sv | And_gate.sv | Or_gate.sv | And_gate_2.sv | Or_gate_2.sv | m_A.sv | m_B.sv | m_C.sv | m_D.sv | m_E.sv | m_F.sv | m_G_1.sv | m_G_2.sv | m_H.sv | m_I.sv | m_J.sv | m_K.sv | m_L.sv | m_M.sv

```
module decoder(A,B,C,D,s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13) ;SV/Verilog Desig
  input A,B,C,D;
  output s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13;
  m_A u1(A,B,C,D,s0);
  m_B u2(A,B,C,D,s1);
  m_C u3(A,B,C,D,s2);
  m_D u4(A,B,C,D,s3);
  m_E u5(A,B,C,D,s4);
  m_F u6(A,B,C,D,s5);
  m_G_1 u7(A,B,C,D,s6);
  m_G_2 u8(A,B,C,D,s7);
  m_H u9(A,B,C,D,s8);
  m_I u10(A,B,C,D,s9);
  m_J u11(A,B,C,D,s10);
  m_K u12(A,B,C,D,s11);
  m_L u13(A,B,C,D,s12);
  m_M u14(A,B,C,D,s13);
endmodule
```

Log | Share

```
# KERNEL: 0 0 0 0 : 1 0 0 1 1 1 0 0 0 0 0 0 0 0
# KERNEL: 0 0 0 1 : 0 0 0 0 1 1 1 0 0 0 0 0 1 0
# KERNEL: 0 0 1 0 : 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
# KERNEL: 0 0 1 1 : 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
# KERNEL: 0 1 0 0 : 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
# KERNEL: 0 1 0 1 : 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
# KERNEL: 0 1 1 0 : 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
# KERNEL: 0 1 1 1 : 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0
# KERNEL: 1 0 0 0 : 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0
# KERNEL: 1 0 0 1 : 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0
# KERNEL: 1 0 1 0 : 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0
```

Differences between codes

python script with regular expressions

```
man = open('design.txt')                    #Se abr
patterns=['module ','input ','output ']
Lines= []
Final=''

for linea in man:
    linea = linea.rstrip()
    for pattern in patterns:
        if re.search(pattern,linea):
            linea = linea.strip()
            #linea = linea.rstrip(",")
            linea = linea.replace("reg","")
            linea = linea.replace("wire","")
            linea = linea.replace(";","")
            linea = linea.replace(")","")
            if re.search('\,',linea):
                linea = linea.rstrip(linea[1])
            Lines.append(linea)
            break
print(Lines)
```

c++ script with regular expressions

```
//int par=module.find('(');
string t;
smatch m;
//std::regex_match (s , m, std::regex("\\b(su
string temp=Prueba;
regex e ("module\\s+(\\w+)\\s*+\\(([^)]+)");
while (regex_search (temp,m,e)) {
    cout<<m[2];
    t.append(m[2]);
    cout << endl;
    temp = m.suffix().str();
}
```

c++ script

```
string text = Prueba;
string temp=Prueba.substr(Prueba.find("module"),Prueba.find('('));
temp.erase(0,Prueba.find("module")+6);
module_name.append(temp);
int a = Prueba.find("(");
//cout << a << endl;
//int b = a + 1;
int c = Prueba.find(")", a);
//cout << c << endl;


for (int i = a; i <= c; i++) {
    Apend.push_back(Prueba[i]);
}
```

RESULT

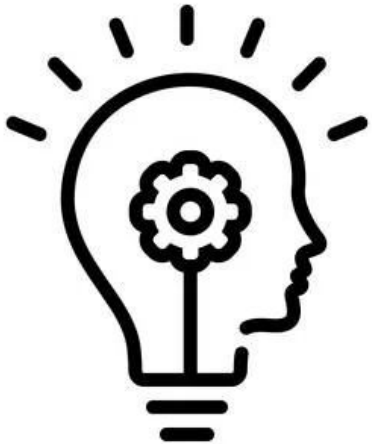
python**Lines="module(input A,output B)"**
c++(regex)**t="(input A,output B)"**
c++**Prueba="(input A,output B)"**

Recommendations

It is possible to find issues in the generated file, they could happen if the format of the Design file were full of module instantiations or the module variables were wrongly declared. Depending on which program we chose, the format will have more or less flexibility, for example, the python script could fail if the declarations of the variables begin with a tabulation or a blank space, otherwise, the simple C++ script is going to successfully make the TestBench no matter how many blank spaces are. The C++ script with regular expressions will return the most polished format.

Conclusions

Each programming language has its rules, pros and cons, but it is essential to find that each and every one of them can have the same structure.



References

1. https://regex101.com/
2. https://en.cppreference.com/w/cpp/regex
3. https://sg.com.mx/content/view/545
4. https://pythex.org/
5. https://www.regextester.com/102475
6. https://docs.python.org/3/howto/regex.html
7. https://docs.python.org/3/library/re.html