Hardware Security and Trust

STMT : A tool for Computing Testability Mesurments

Tarbiat-Modares University
Spring 2020

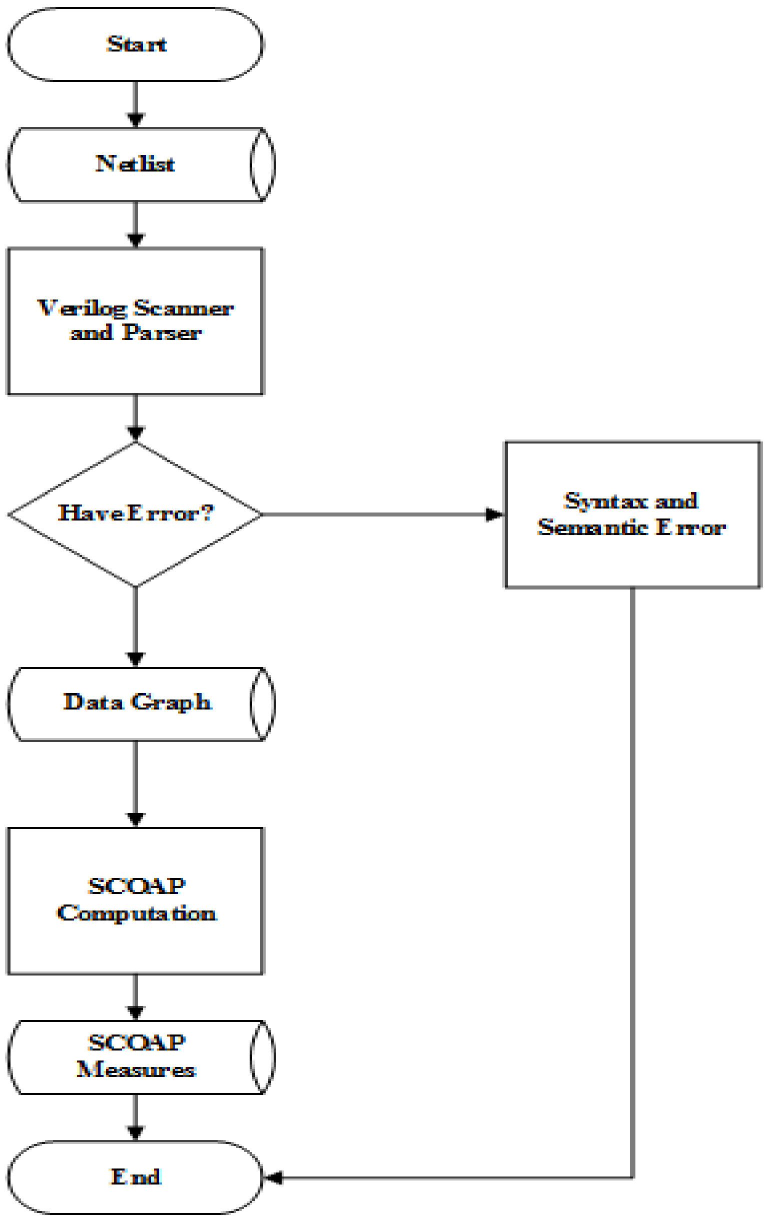
A. Shafieinejad
shafieinejad@modares.ac.ir

Table of Contents

STMT

Architecture:

- ✓ Scanner
- ✓ Parser
- ✓ Data graph
- ✓ SCOAP Computations



STMT

Scanner

- Token formats are written in *.l file
- Converted to *.c file using win_flex (lexical analyzer)

Parser

- Verilog Language Syntax is written in *.y file
- Converted to *.cpp file using win_bison
  - LALR Parser Generator

STMT

Scanner source (verilog_s.l)

- **Keywords**
  - "module" {return T_MODULE;}
  - "endmodule" {return T_ENDMODULE;}
  - "input" {yylval.ival = ST_IN; return T_SIG_TYPE;}
  - "output" {yylval.ival = ST_OUT; return T_SIG_TYPE;}
  - "wire" {yylval.ival = ST_WIRE; return T_SIG_TYPE;}
  - "reg" {yylval.ival = ST_REG; return T_SIG_TYPE;}
  - "assign" {return T_ASSIGN_STR;}
- **Operators**
  - "(" {return T_LEFT;}
  - ")" {return T_RIGHT;}

STMT

Verilog Parser Source (verilog_p.y)

- **Grammar Rules**
  - start-var: λ | module-statement start-var
  - module-statement : module-def-statement statement-list T_ENDMODULE
  - module-def-statement : T_MODULE T_IDENTIFIER

```
 {      st.h_module = create_module($2);
        st.hModules[st.nModule++] = st.h_module;
        st.state = MODULE_DEFINE;
        nNode = 0; }  T_LEFT sig_list T_RIGHT T_SEMICOLON
statement-list : λ | statement T_SEMICOLON statement-list
statement : λ    | signal-assign-statement  | signal-def-statement
           | component-instantiate-statement
signal-def-statement :      signal-type sig_list
sig_list : λ | sig_name   | sig_name T_COMMA sig_list

......
```

STMT

Data Structures

- **Module**

```
struct Module {
        char name[MAX];
        int nIn, nOut, nWires, nNode;
        signal      *p_inputs[MAX_INPUT];
        signal      *p_outputs[MAX_OUTPUT];
        signal      *p_wires[MAX_WIRES];
        node        *p_nlist[MAX_NODE];
        signal  gnd, vcc;
}
```

- **Singal**

```
struct signal {
        char name[MAX];
        SType type;
        node *generator;
        TestabilityState st;
};
```

STMT

Data Structures

- **Node**

```
struct node
{
        int stdType;
        char name[MAX];
        EType type;
        // handle for user-defined module
        void *hModule;
        int nIn, nOut;
        signal *In[MAX_INPUT];
        signal *Out[MAX_OUTPUT];
        bool CC_Evaluated;
}
……
```

STMT

Data Structures

- **StdCell**

```
struct StdCell
{
        EType e;
        char name[64];
};
struct StdCellInfo
{
        int nStdType;
        StdCell cells[80];
        int nIn, nOut;
        // clk and reset are determined in FF SC and SO
        portInfo in_ports[MAX_PORT], out_ports[MAX_PORT];
}
……
```

STMT

Example for StdCell

- **FullAdder in Library 180 nm**

```
{ STD180,{ { ADDFXL,"ADDFXL" },{ ADDFX1,"ADDFX1" },{
ADDFX2,"ADDFX2" },{ ADDFX4,"ADDFX4" }, …},
        3, 2,
        { { PRIMARY_IN, "A" },{ PRIMARY_IN, "B" },{ PRIMARY_IN, "CI" } },
        { { PRIMARY_OUT, "CO" },{ PRIMARY_OUT, "S" } }
```

- **FullAdder in Library 90 nm**

```
{ STD90,{ { FADDX1,"FADDX1" },{ FADDX2,"FADDX2" } } ,
3,2,
{ { PRIMARY_IN, "A" },{ PRIMARY_IN, "B" },{ PRIMARY_IN, "CI" } },
{ { PRIMARY_OUT, "CO" },{ PRIMARY_OUT, "S" } }
```

STMT

Example for StdCell

- **D-FlipFloo in Library 180 nm**

  { STD180,{ { DFFRHQXL, "DFFRHQXL" },{ DFFRHQX1, "DFFRHQX1" },{ DFFRHQX2, "DFFRHQX2" },{ DFFRHQX4, "DFFRHQX4" } },

  3, 1,

  { { PRIMARY_IN, "D" },{ CLOCK, "CK" },{ ASYNC_RESET, "RN" } },

  { { PRIMARY_OUT, "Q" } }

- **D-FlipFloo in Library 90 nm**

  { STD90,{ { DFFASX1,"DFFASX1" },{ DFFASX2,"DFFASX2" },{ DFFNASX1, "DFFNASX1" },{ DFFNASX2, "DFFNASX2" },

  { LASX1, "LASX1" },{ LASX2, "LASX2" } } ,

  3,2,

  { { PRIMARY_IN, "D" },{ PRIMARY_IN, "SETB" },{ CLOCK, "CLK" } },

  { { PRIMARY_OUT, "Q" },{ PRIMARY_OUT, "QN" } }

STMT

How to consider assignment?

- **Sig1 <= Sig2;**

Note that, all of testability measurements of sig1 and sig2 are the same.

We define it as Dummy element

```
{ STD180,{ { DUMMY,"C_ASSIGN" }} ,
1,1,
{ { PRIMARY_IN, "IN" } },
{ { PRIMARY_OUT, "OUT" } }
},
```

It is considered as a Verilog code like this:

```
DUMMY   d1 (.IN(sig1), .OUT(sig2) );
```

STMT : Initialization Algorithm

```
input: Module m
output: NULL
1.  for each signal s in m do begin
2.      if (s is primary input) then
3.          s.cc = 1;
4.          s.sc = 0;
5.          s.co = s.so = ∞;
6.      else if (s in primary output)
7.  then
8.          s.cc = s.sc = ∞;
9.          s.co = s.so = 0;
11.     else
12.         s.cc = s.sc = ∞;
13.         s.co = s.so = ∞;
14.     end if
15. end for
```

STMT : Controllability Algorithm

First Candidates:

- The nodes connected to Primary Inputs

Next Candidates:

- The nodes connected to recently updated node

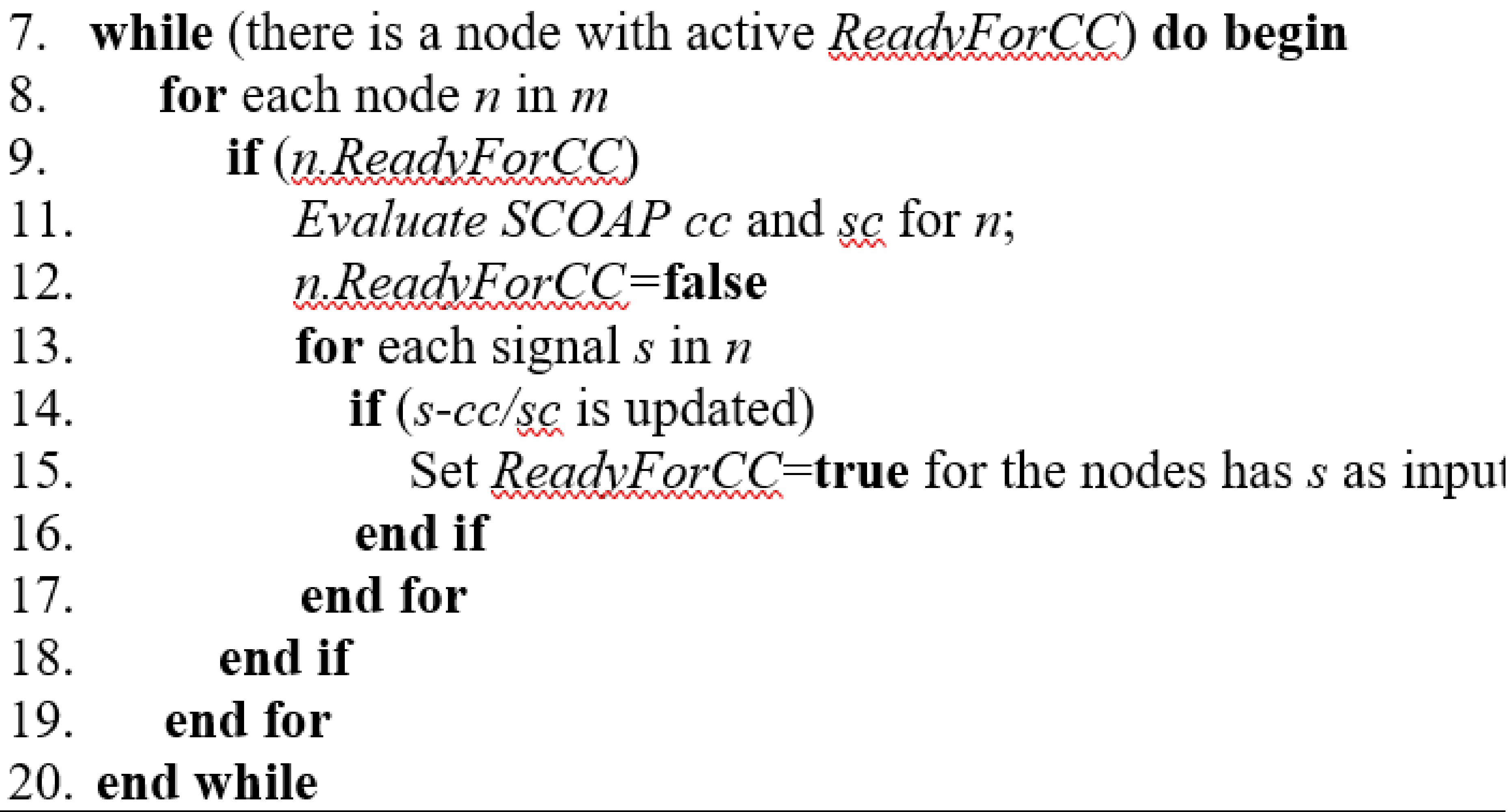
Loop Condition:

- There is a node which the CC/SC values are updated

STMT : Controllability Algorithm

```
input:   Module m
output:  CC/SC values of signals
-----------------------------------------------------------
1.    for each node n in m
2.        if (any input of n is a primary input) then
3.            n.ReadyForCC = true;
4.        else
5.            n.ReadyForCC = false;
6.    end for
```

STMT : Controllability Algorithm (2)



```
7.  while (there is a node with active ReadyForCC) do begin
8.      for each node n in m
9.          if (n.ReadyForCC)
11.             Evaluate SCOAP cc and sc for n;
12.             n.ReadyForCC=false
13.             for each signal s in n
14.                 if (s-cc/sc is updated)
15.                     Set ReadyForCC=true for the nodes has s as input
16.                 end if
17.             end for
18.         end if
19.     end for
20. end while
```

STMT : Observability Algorithm

First Candidates:

- The nodes connected to Primary Outputs
- Generator nodes of the primary outputs

Next Candidates:

- The nodes connected to recently updated node (CO/SO)
- Simply identified by the *generator* field of the signal

Loop Condition:

- There is a node which the CO/SO values are updated
- Exit in the case of NO Change

STMT : Observability Algorithm (1)

```
input:  Module m
output: CO/SO values of signals
------------------------------------------------
1.   for each node $n$ in $m$
2.       $n.ReadyForCO$ = false
3.   end for
4.   for each output signal $s$ of $m$
5.       node $g$ = generator node of signal $s$
6.       $g.ReadyforCO$ = true
7.   end for
```

STMT : Observability Algorithm (2)

```
8.  while (there is a node with active ReadyForCO) do begin
9.      for each node n in m do begin
11.         if (n.ReadyforCO) then
12.             Evaluate SCOAP co and so for n;
13.             n.ReadyforCO = false
14.             for each input signal r of n
15.                     if (r.co/so is updated and r is not a primary input)
16.                         generator(r).ReadyForCO = true
17.                 end if
18.             end for
19.         end if
20.     end for
21. end while
```