Software - Logic Simulator Project

First interim report

S. Arulselvan, F. Freddi, A. I. Lam

Easter Term 2018

1 Introduction

This report introduces the reader to the general approach to the logic simulator problem. In the first few sessions of the projects we described the EBNF syntax of our logic simulator and we identified the possible semantic errors that might arise, considering how these should be handled. Two examples of how some of the most common circuit designs are represented using definition files consistent with the defined syntax. The team <- WHAT?

2 Teamwork

TODOShrin add here: add a brief description of how we decided to split the work between us and which tasks we are expecting to take more time than the others. Add brief comment to the gantt chart and refer to it in the appendix
add comment on expected rework and margin of time kept in case something goes wrong

3 Specification of the language

3.1 Error identification

3.1.1 Identification of syntax errors and EBNF

The EBNF is the stucture defining what rules the description of our circuit need to conform to and when syntax errors are thrown.

```
capitalletter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K"
              | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
              | "W" | "X" | "Y" | "Z" ;
lowerletter= "a" | "b"  | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l"
           | "m" | "n" | "o" | "p"  | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x"
           | "y" | "z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

structure=devicelist,connectionlist,monitorlist,"END";

deviceslist= "DEVICE" , { devicedef } ;
devicedef= devicetype, device, { "," , device} , ";" ;
devicetype= capitalletter, {capitalletter};
device=devicename, [ "(" , digit , { digit } , ")" ];
devicename=(lowerletter | capitalletter), { lowerletter | capitalletter | digit };

connectionlist="CONNECT", {connection};
connection= output "->",input,{input},";";
input=devicename,".",capitalletter,{ capitalletter | digit };
output=decivename, [".", capitalletter, {capitalletter}];
```

```
monitorlist="MONITOR", output, { "," , output};
```
Listing 1: EBNF of the logic syntax

3.1.2 Identification of semantic errors

Considering the EBNF syntax design, further types of semantic errors can be identified:

1. a set of capital letters given for *devicetype* is not a valid recognised *devicetype*
2. a device name is used more than once.
3. the device name corresponds to a keyword or to a device type (considered case insensitive)
4. a device is defined with parameters when the devicet type does not allow it (e.g. XOR(5))
5. a device is defined with parameters out of the parameter range (e.g. SWITCH(4))
6. multiple outputs going to the same input
7. an input is not mentioned in the connections
8. input index to a device must be less than or equal to the number of inputs initialised in DEFINE
9. input/output name in CONNECT does not match the possible inputs of the device type
10. in CONNECT and MONITOR, the *devicename* has not been defined in DEFINE
11. the same signal is monitored twice

3.2 Error handling

Our approach to error handling starts from the syntax errors. If a syntax error is found, it is output and the parsing is continued on the rest of the file starting from the next semicolumn checking for the remaining syntax errors. If no syntax errors are found, we move to semantic error checking.

3.2.1 Syntax errors

All the syntax error are approached similarly displaying SYNTAX ERRORS followed by the line number, the involved code and an arrow underneath indicating where in line syntax error has occurred. Example:

```
SYNTAX ERROR   Line 42: SWITCH 5n2;
                                ^
```

3.2.2 Semantic errors

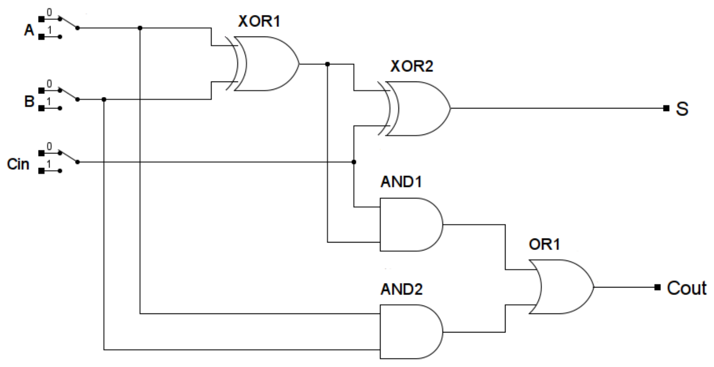
Each type of semantic error is handled individually and, despite not being mentioned in the table, line number and the code generating the error will be included at beginning of the message. For the error number, refer to Section 3.1.2.

Error #	Handling
1	Invalid device type: “devicetype”
2	Device name “devicename” was already used in line “line”
3	Cannot use keyword, “devicename”, as a device name
4	“devicetype” type devices do not take parameters
5	SWITCH type device can only take 0 or 1 as parameter
6	Input “input” was already used in line line
7	Input “input” was never used
8	“input” cannot be used the “num_inputs” input “devicename” device
9	“input”/“output” is not valid for “devicetype” device “devicename”
10	Undefined device name: “devicename”
11	“signal” is already monitored: line “line”

Table 1: Error messages for each type of possible semantic error identified

4 Example definition files

Two circuits are given in Figures 1 and 2, by circuit diagrams and by the language defined above.



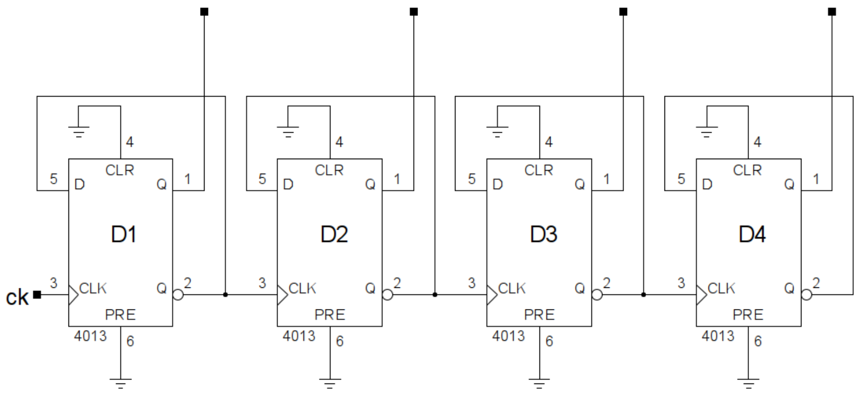
(a) Circuit diagram for a full adder

```
# full adder
DEVICE
    XOR xor1, xor2;
    AND and1, and2;
    OR  or1;
    SWITCH a(0), b(0), cin(0);
CONNECT
    a -> xor1.I1, and2.I1;
    b -> xor1.I2, and2.I2;
    cin -> xor2.I2, and1.I1;
    xor1 -> xor2.I1, and1.I2;

    and1 -> or1.I1;
    and2 -> or1.I2;
MONITOR
    xor2, # s
    or1   # cout
END
```

(b) Full adder described by our language

Figure 1: A full adder, described equivalently by a standard circuit diagram and by our language.



(a) Circuit diagram for a ripple counter

```
# ripple counter
DEVICE
    DTYPE d1, d2, d3, d4;
    CLOCK ck / 2;
    SWITCH gnd(0);
CONNECT
    gnd -> d1.SET, d1.CLEAR, d2.SET, d2.CLEAR, d3.SET, d3.CLEAR,
           d4.SET, d4.CLEAR;

    d1.QBAR -> d1.DATA;
    d2.QBAR -> d2.DATA;
    d3.QBAR -> d3.DATA;
    d4.QBAR -> d4.DATA;

    ck      -> d1.CLK;
    d1.QBAR -> d2.CLK;
    d2.QBAR -> d3.CLK;
    d3.QBAR -> d4.CLK;
MONITOR
    d1.Q, d2.Q, d3.Q, d4.Q
END
```

(b) Ripple counter described by our language

Figure 2: A 4-bit ripple counter. Note that the circuit diagram is slightly simplified by implementing `gnd` directly as ground instead of a switch set to `0`, to avoid clutter in the diagram.

Appendices

A Gantt chart

TODOShrin add here