University of Cambridge

Engineering Tripos Part IIA

GF2: SOFTWARE

First Interim Report

Clinton Igwegbu, David de Oliveira, Steve Zhang May 18, 2019

1. Introduction

The aim of this project is to develop a logic simulation program in which the logic circuit is defined using a newly specified language. Most of the modules required for this are already complete and supplied: it is the job of the team to firstly specify the logic description language, and then to develop the scanner and parser for the language, and finally to create a user friendly GUI.

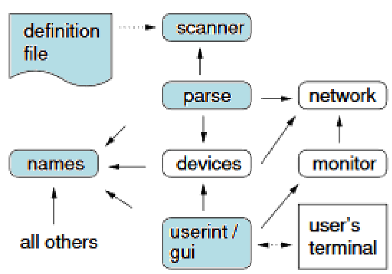

Figure 1: Structure of program, with arrows showing communication between modules; The modules in blue are all yet to be developed and implemented. Source: GF2 Software introductory presentation.

It has been decided that the main design criterion of the simulation program is that it should ensure a user-friendly experience. Keeping this in mind, the grammar definition will be simple, meaning not many syntax errors will be expected, but rather a host of semantic errors. Fortunately most of the semantic errors will not be too difficult to identify, some may even be correctable. Thus instead of stopping the simulation program if any error is encountered, it has been decided that the simulation program should endeavour to correct semantic errors that are minor by offering the user a suggestion for a possible correction. If the user accepts the suggestion, then the program will execute the simulation.

1.1 Teamwork and Planning

Maintaining as modular an approach as possible to the program, each team member will be the main driving force for one aspect of the project, which means that they will do most of the preliminary design and implementation of that part. The other team members will review their work, offering feedback on what works well, and if any possible improvements can be made, as well as writing tests to ensure the code functions as expected. Specific details of the planning and scheduling can be seen in the chart in Appendix A.

2. EBNF Syntax

One of the first parts of the project is to define the logic description language, which has been done using the Extended Backus-Naur Form for grammar definitions. Corresponding to the criterion of simplicity and user-friendliness, it has been decided that to write a definition file, all that is required is to specify a device list, followed by a connection list, followed by a monitor list.

```
DEVICELIST        = "DEVICE_LIST" , ":" , DEVICE , { ";" , DEVICE } ,
                        ";" , "END" ;

DEVICE            = DEVICETYPE , "'" , NAME , "'" , [ PROPERTY ] ;

DEVICETYPE        = GENERATOR | MULTIINPUTLE | OTHERLE ;
GENERATOR         = "CLOCK" | "SWITCH" ;
MULTIINPUTLE      = "AND" | "NAND" | "OR" | "NOR" ;
OTHERLE           = "XOR" | "DTYPE" ;

NAME              = letter , { letter | number } ;

PROPERTY          = NUMREPETITIONS | INISTATE | NUMINPUT ;
NUMREPETITIONS    = "NUM_REPETITIONS" , "=" , number ;
INISTATE          = "INITIAL_STATE" , "=" , ( "OFF" | "ON" ) ;
NUMINPUT          = "NUM_INPUT" , "=" , number ;

CONNECTIONLIST    = "CONNECTION_LIST" , ":" , CONNECTION ,
                        { ";" , CONNECTION } , ";" , "END" ;

CONNECTION        = SIGNAL , "->" , SIGNAL ;

SIGNAL            = INPUT | OUTPUT ;
INPUT             = NAME , "." , ( GATEINPUT | DTYPEINPUT ) ;
GATEINPUT         = "I" , [ number ] ;
DTYPEINPUT        = "DATA" | "CLK" | "SET" | "CLEAR" ;
OUTPUT            = NAME , [ "." , ( "Q" | "QBAR" ) ] ;

MONITORLIST       = "MONITOR_LIST" , ":" , MONITOR , { ";" , MONITOR } ,
                        ";" , "END" ;

MONITOR           = OUTPUT | INPUT ;

number            = digit , { digit } ;
letter            = "A" | .... | "z" ;
digit             = "0" | .... | "9" ;
```

As can be seen from the above grammar, writing a definition file is quite simple:

- The device list begins with the keyword DEVICE_LIST, followed by a colon, followed by a semi-colon separated list of devices. The list finishes with the END keyword.
  - A device is made up of the device type, a user defined name surrounded in single quote marks, followed by an optional property.
  - The delineation of GENERATOR, MULTIINPUTLE and OTHERLE is only included to convey the different types of devices available: OTHERLE takes no property when defining the device; the MULTIINPUTLE takes the NUMINPUT property; and GENERATOR takes either the NUMREPETITIONS or INISTATE properties, depending on whether it is a clock or switch respectively.
- The connection list then follows the device list. Similar to the device list, it commences with the keyword CONNECTION_LIST, is followed by a colon, followed by a semi-colon separated list of connections. The list again finishes with the keyword END.
  - Connections are made up of outputs connected to inputs. Though syntactically a reverse connection is allowed, this is semantically incorrect (see Section 3 for more details on errors).
- Lastly, a monitor list is defined, again in a similar way to the device list, and to the connection list.
  - Although both outputs and inputs are allowed to be defined as monitor points, only outputs are actually monitored by the Monitor module; the output corresponding to the connection with the input will be found and monitored if an input is specified in the monitor list.

2.2. Example Definition Files and Circuits

Below are example definition files which make concrete use of the above grammar. The first example is a ripple (down) counter using clocked D-Type flip flops. The second example is of a simple half-adder, which makes use of an AND gate and a XOR gate. These two examples are sufficient in showing use of all the features of the grammar.

2.2.1. Ripple counter

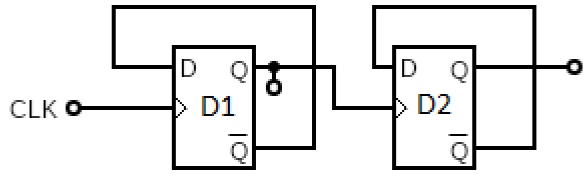

Figure 2: Ripple counter circuit diagram

```
DEVICELIST:        DTYPE 'D1';
                   DTYPE 'D2';
                   CLOCK 'CLK' NUM_REPETITIONS=10;
                   END

CONNECTIONLIST:    CLK -> D1.CLK;
                   D1.QBAR -> D1.DATA;
                   D1.Q -> D2.CLK;
                   D2.QBAR -> D2.DATA;
                   END

MONITORLIST:       D2.Q;
                   D1.Q;
                   END
```

2.2.2. Half adder

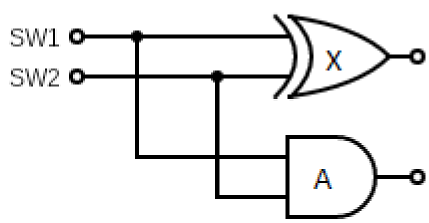

Figure 3: Half Adder circuit diagram

```
DEVICELIST:
XOR 'X'; AND 'A' NUM_INPUT=2;
SWITCH 'SW1' INITIAL_STATE=OFF; SWITCH 'SW2' INITIAL_STATE=ON; END

CONNECTIONLIST:
SW1 -> X.I1; SW1 -> A.I1;
SW2 -> X.I2; SW2 -> A.I2; END

MONITORLIST:
X; A; END
```

3. Error Handling

In order to make the program robust, error handling will need to be implemented. Syntax errors occur whenever one of the production rules in the defined grammar is broken. When a syntax error occurs, the scanner module will be able to scan to the next stopping symbol, and resume scanning from there. Error handling will consist of printing a message to the user, saying where the error takes place – by means of line number and a caret underneath the position of the error in the line – in the definition file. All errors will be collated and reported after parsing is complete.

Semantic errors – where the syntax is correct but the resulting statement is not meaningful – will be detected by parsing the received symbols from the scanner module. If these are detected, the user will be notified and instructed to make corrections to their defintion file. Some semantic errors are easy to detect, and even correct: the program will therefore offer suggestions of possible corrections for these easier-to-handle errors to the user, and attempt to proceed with the simulation.

Below are lists of all common semantic errors that can occur while using our language. The errors are categorised according to their origin, and are separated into DEVICELIST, CONNECTIONLIST and MONITORLIST. The example codes given are all counterexample to demonstrate the errors.

3.1. DEVICELIST originated errors

Error 1: number of inputs of MULTIINPUT gates must be $\leq$ 16

```
e.g.    NAND  'G1'  NUM_INPUT=17;
```

Error 2: Device name cannot be defined more than one time

```
e.g.    NAND  'G1'  NUM_INPUT=3;
        AND   'G1'  NUM_INPUT=5;
```

Error 3: Specifying wrong or redundant properties.

```
e.g.    DTYPE  'DTYPE1'  NUM_INPUT=5 ;
        SWITCH 'SW1'     INITIAL_STATE=2;
```

3.2. CONNECTIONLIST originated errors

Error 4: Input number must be less than the number specified in DEVICELIST

```
e.g.    /*In DEVICELIST*/         AND 'G1' NUM_INPUT=4;
        /*In CONNECTIONLIST*/     SW1  ->  G1.I6 ;
```

Error 5: Name of input/output must be specified previously in DEVICELIST

```
e.g.    /*In DEVICELIST*/       SWITCH 'SW1' INITIAL_STATE=OFF;
                                AND 'G1' NUM_INPUT=3; END
        /*In CONNECTIONLIST*/   SW1  ->  G10.I2 ;
                                WS1  ->  G1.I1 ;
```

Error 6: Input/output signals cannot be the same

```
e.g.    SW1  ->  SW1 ;
```

Error 7: Gate-input cannot be the signal input

```
e.g.    G1.I1  ->  G2.I1 ;
```

Error 8: Gate-output cannot go to output

```
e.g.    SW1  ->  G1 ;
```

Error 9: Switches cannot behave as inputs

```
e.g.    G1  ->  SW1 ;
```

Error 10: Clock cannot behave as an input

```
e.g.    G1  ->  CLK ;
```

3.3. MONITORLIST originated errors

Error 11: Name of device to which output/input belongs must be specified previously in DEVICELIST

```
e.g.    /*In DEVICELIST*/   AND 'G1' NUM_INPUT=5 ; END
        /*In MONITORLIST*/  G2;
```

Appendix A: Planning

		Week 1						Week 2						Week 3						Week 4						
		T	F	S/S	M	T	W	T	F	S/S	M	T	W	T	F	S/S	M	T	W	T	F	S/S	M	T	W	T
	Preliminary exercises	C/D/S	C/D/S	C/D/S	C/D/S	C/D/S																				
Phase 1	Discuss approach					C/D/S	C/D/S																			
	Define language					C/D/S	D	D	D	C/D/S																
	Define errors								S	C/D/S																
	Complete Names module									S	S	S	S	S												
	Test Names module												C/D	C/D	C/D											
	Complete Scanner module									S	S	S	S	S												
	Test Scanner module												C/D	C/D	C/D											
	Complete Parser module									D	D	D	D	D												
	Test Parser module												C/S	C/S	C/S											
	Test system with command line interface														D	D										
Phase 2	Design GUI									C	C	C	C	C/D/S												
	Build GUI														C	C	C	C								
	Implement GUI																	C	C	C						
	Test GUI																		D/S	D/S						
	Test entire system																				C/D/S					
	Maintainance																					C/D/S	C/D/S	C/D/S	C/D/S	

Figure 4: Timeline of project; Days highlighted in red correspond to report deadlines
C = Clinton; D = David; S = Steve

Appendix B: LL(1) Parse Table

		Terminals										
		letter	DEVICE_ LIST	[device type]	NUM_ REPS	INI_ STATE	NUM_IN	CON_ LIST	I	[dtype input]	MON_ LIST	other terminals
Non-terminals	DEVICELIST	-	1	-	-	-	-	-	-	-	-	-
	DEVICE	-	-	3	-	-	-	-	-	-	-	-
	PROPERTY	-	-	-	12	12	12	-	-	-	-	-
	NUMREPETITIONS	-	-	-	13	-	-	-	-	-	-	-
	INISTATE	-	-	-	-	14	-	-	-	-	-	-
	NUMINPUT	-	-	-	-	-	15	-	-	-	-	-
	CONNECTIONLIST	-	-	-	-	-	-	17	-	-	-	-
	CONNECTION	19	-	-	-	-	-	-	-	-	-	-
	SIGNAL	21	-	-	-	-	-	-	-	-	-	-
	INPUT	22	-	-	-	-	-	-	-	-	-	-
	GATEINPUT	-	-	-	-	-	-	-	23	-	-	-
	DTYPEINPUT	-	-	-	-	-	-	-	-	24	-	-
	OUTPUT	25	-	-	-	-	-	-	-	-	-	-
	MONITORLIST	-	-	-	-	-	-	-	-	-	27	-
	MONITOR	29	-	-	-	-	-	-	-	-	-	-
	NAME	10	-	-	-	-	-	-	-	-	-	-

Figure 5: LL(1) Parse table for the grammar defined in the report. The numbers in the table correspond to the line numbers of the production rules in Section 2. Some simplifications have been made: i.e. treating DEVICETYPE and DTYPEINPUT as terminals; collecting all terminals that aren't in the parse First set into "other terminals".