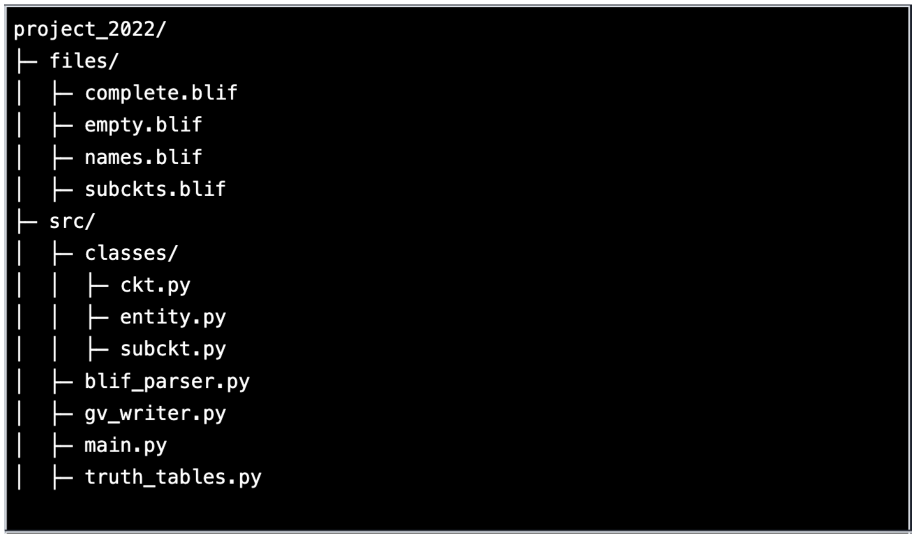
Blif Parser Guide

Matteo Alberici

April 2022

1 Introduction

This document is a guide for using the blif parser I implemented during the first part of my Bachelor project.
What my project does until now consists of receiving a blif file in input and converting it into a gv file. Moreover, it takes the newly generated gv file and creates the correspondent pdf file using **dot**.
The following is the file tree that represents the structure of my project:



```
project_2022/
├─ files/
│  ├─ complete.blif
│  ├─ empty.blif
│  ├─ names.blif
│  ├─ subckts.blif
├─ src/
│  ├─ classes/
│  │  ├─ ckt.py
│  │  ├─ entity.py
│  │  ├─ subckt.py
│  ├─ blif_parser.py
│  ├─ gv_writer.py
│  ├─ main.py
│  ├─ truth_tables.py
```

Figure 1: File Tree

Let's have a closer look to all the files the blif parser is composed of.

2 Source Files

2.1 Classes

In order to deal with a circuit and all the gates it is composed of, I decided to implement three different classes, described in the following paragraphs.

Entity

Entity is the super class that represents an object with a given set of **inputs** and a given set of **outputs**. It has only one class method, **generate()**, which is abstract. The purpose of this class is to define a blueprint which is common to both the sub-classes needed for parsing a blif file.

Ckt

Ckt is a sub-class of entity and represents a circuit object with a given set of inputs, a given set of outputs, and finally a list of sub-circuits the circuit itself is composed of. It has only one class method, **generate(**$file\_name$**)**, which overrides the one defined in the entity class. The method takes in input a blif file, retrieves from it the list of inputs and the list of outputs, and finally returns the circuit object. The list of sub-circuits will be filled while parsing the entire file.

Subckt

Subckt is another sub-class of the entity class and represents a sub-circuit object with a given input, a given output, and finally a given operator. It has only one class method, **generate(***_input*, *output*, *operator***)**, which overrides the one defined in the entity class. The method takes an input, an output, and an operator and simply generates a sub-circuit with the elements received in input.

2.2 Truth Tables

In a blif file, a sub-circuit can be represented by the following lines:

(1) .subckt *operator input*(*s*) *output*

(2) .names *input*(*s*) *output*

The lines starting with ".*names*" are followed by one or more lines defining the truth table obeyed by that sub-circuit.
In the file *truth_tables.py*, there are two dictionaries, one for the unary gates and one for the binary ones, and one function, **permute(***gate*, *combinations***)**, which evaluates all the possible ways to describe an operator. While parsing a blif file, the dictionaries are read in order to find and assign the correct operator.

Operator	**Input**	**Output**
zero	-	0
one	-	1
not	0	1
assign	1	1

Table 1: Unary operators table

Operator	**Input**	**Output**
zero	- -	0
and	1 1	1
not_imply	1 0	1
one_dc	1 -	1
zero_one	0 1	1
dc_one	- 1	1
xor	0 1 1 0	1 1
or	- 1 1 -	1 1
nor	0 0	1
equality	0 0 1 1	1 1
dc_zero	- 0	1
zero_dc	0 -	1
one	- -	1

Table 2: Binary operators table

Operator	**Input**	**Input 2**	**Input 3**	**Output**
not_zero_one	- 0 1 -	0 0 1 -	- 0 1 1	1 1
imply	0 - - 1	0 0 - 1	0 - 1 1	1 1
nand	0 - - 0	0 - 1 0	- 0 0 1	1 1

Table 3: Binary operators second table

2.3 Blif Parser

The file *blif_parser.py* defines the core function of the translator: **blif_parser(***file_name***)**. The function takes a blif file, generates a circuit, and then starts reading every line of the file, creating a sub-circuit for each line read. After all the sub-circuits defined by the file have been created, the parser renames the inputs and the outputs through the function **fix_syntax(***string***)**, in order to avoid errors caused by the gv syntax. Finally, the parser removes all the "*assign*" nodes which do not lead to an output of the circuit.

2.4 Gv Writer

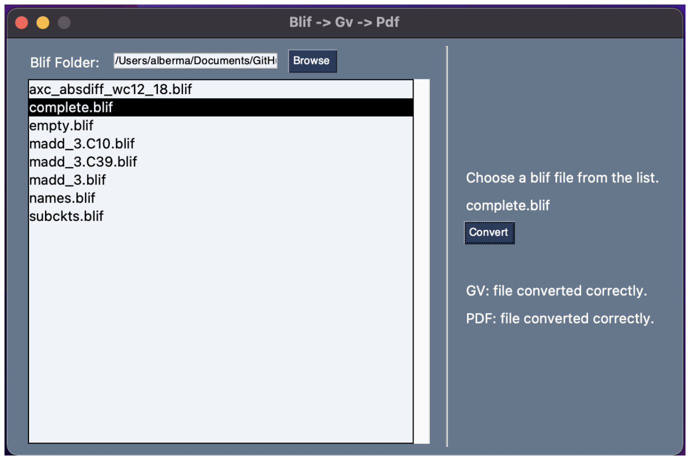
The file *gv_writer.py* defines the function which takes a file, creates a circuit object by parsing that file, and finally writes the correspondent gv file.

2.5 Main

The file *main.py* displays a simple GUI with some instructions in order to convert a blif file to a gv file and finally to the correspondent pdf file:

1. By clicking the button "Browse", the user chooses a folder containing at least one blif file
2. The user chooses a file among the ones displayed
3. By clicking the button "Convert", the file is first converted to a gv file, and then to a pdf file.

The GUI also displays error/success messages after every action made.



/Users/alberma/Documents/GitH

axc_absdiff_wc12_18.blif
complete.blif
empty.blif
madd_3.C10.blif
madd_3.C39.blif
madd_3.blif
names.blif
subckts.blif

Choose a blif file from the list.

complete.blif

GV: file converted correctly.

PDF: file converted correctly.

Figure 2: GUI