Automatically Building Digital Symbol Libraries

Arjun Rakheja
Poolesville High School
Poolesville, MD, USA
arjun.rakheja@gmail.com

Digital circuit symbols do not currently have commonly accepted open standards for the formats of such symbols. Consequently, digital cell libraries often come in proprietary formats. The purpose of this project is to automatically create a digital symbol library from a liberty file in the SkyWater pdk. An algorithm was developed in Python and a standard digital cell library was created in KiCad and XSCHEM. The algorithm is able to parse through the liberty file, store information on the cells, and successfully output a digital symbol library. In future work, steps will be taken to automatically create a spice netlist from the liberty file.

I. Importance of Problem

When circuit chips manufacturing and its software first came out, it was free to use. Eventually, the software became license, and only the large companies and rich research institutions could afford them. Eventually, the software became dominated by three large companies: Cadence, Mentor Graphics, and Synopsys. These manufacturers create Process Design Kits (PDK's) and sell them so the public can use their software. The research involved in this project is one of the many aspects involved to allow engineers to manufacture chips freely with open source software. One important part of chip manufacturing are digital standard cell libraries. These are collections of low-level logic functions. Most standard cell libraries have a limited number of cell types, including INVERT and BUFFER; then AND, OR, XOR, XNOR, which come in variations of 2, 3,and 4. Additionally, there are a few useful combinations of gates AOI (and-or-invert) and OAI (or-and-invert), MUX (multiplexer, usually 2- and 4-input), TRIINV and TRIBUF (tri-state inverter and buffer, and FA and HA (full- and half-adder circuits).

The cells in standard cell libraries are organized with fixed-height, variable-width full-custom cells. The libraries being fixed height is important. This enable the cells to be placed in a fixed number rows and an unlimited number of columns, making the process of automating the digital layout of the cell easier. The cells are typically optimized full-custom layouts, which minimize delays and area. However, the description of digital standard cell libraries are not accessible by the public, and even if they are they usually are not readable. descriptions of digital standard cell libraries often come in a proprietary format (such as Synopsys), a proprietary and binary, human unreadable format such as Open Access, or have no symbols at all. Instead, most open source schematic entry tools have their own unique symbol formats.

However, the liberty file format is very often used for every digital standard cell library offered as part of a foundry process design kit (PDK). The liberty file contains "functions" that describes the Boolean function of a digital logic gate as a Boolean equation. This liberty file can be parsed through to identify the different digital logic gates that are being used. Unfortunately, the liberty file format allows many ways for describing a Boolean function. For example, to represent to represent "A AND B", you can have "A & B", "A B", and possibly "A * B" as well.

Not much research has been done in this component of EDA development. Hirotaka Terai and his colleagues have done research on a standardized digital cell library for a single flux quantum circuit. Their research is similar to mine in that they are creating a cell library out of complicated and proprietary pieces of code. Additionally, Miguel Miranda and his colleagues have done research on statistical comparisons in Electronic Automation Design with the Monte Carlo computational algorithms.

II. Objective

The goal of this research was to take a disorganized, complicated liberty file and automatically create a digital symbol library based on a known set of standard cells. This set of known symbols was created in KiCad and XSCHEM. Cell information would be gathered and stored by parsing the liberty file to create a digital symbol library in KiCad and XSCHEM.

III. Methodology

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar. I used Python to create the liberty file parser. The first step was to read the liberty file into a list. I did this in my main function called parser(). This function opened the liberty file and created a list, where each element was a line in the file (in chronological order). I wrote a function called get_cell_names() which took the list of lines and scanned

through searching for the line with the cell name. This denotes the beginning of the cell. The function stored the cell names of each cell in the liberty file into a list called cell_names. Every time the function finds a cell name (the beginning of a cell) is stores that line's number in a list called cell_divisions. That way, I know the ends and beginnings of each cell's lines in the liberty.

The next step was to store the functions of each of the cells. Using cell_divisions, I created a function called get_function searched through only the liberty files lines belonging to that cell for it's information. I parsed through this group of lines to find the function line of the cell name. Since some cells have multiple functions, the list of functions, called cell_function, was a list of lists, where each element of the outer list was a list with the one or multiple functions in it. The function then returns this list of cell functions.

After, I had to store the footprint of the cell. The same methodology as the get_function function was used to create a function called get_footprint. This function parses through the group of lines from the liberty file (determined by the cell_divisions list) and stores the footprint of each cell in a list, and then returns this list. Since cells only have one footprint, this function is a little simpler than get_functions.

Then, the next step was to determine if the cell was a sequential gate, and what type was it. I divided this sequential category into two groups: flip flops and latches. To determine if the cells in the liberty file was a flip flop, I wrote a procedure called get_sequential. This procedure uses cell_divisions and searches the cell lines in the liberty file to find the string "ff (" which denotes the cell as a flip flop. Once it is determined that the cell is a flip flop, the parser finds the values for the following variables: Clock, preset, clear, and next state. Additionally, if the "ff (" string has two elements after the parentheses("IQ","IQ_N), that means that the flip flop has two outputs. All this information is appended to a temporary list, and this list is appended to a larger list called sequential_cells. Each element in this list contains the sequential information for the cell, and an empty list if the cell is not sequential. I wrote another procedure called get_latch, which determines whether the cell in the liberty file is a latch. This procedure uses cell_divisions to search for the string "LATCH (". This string denotes that the cell in the SkyWater liberty file is a Latch gate. Then the parser finds values for the following variables: data_in, clear, preset, and enable. Additionally, if the "LATCH (" string has two elements after the parentheses("IQ","IQ_N), it has two outputs. Same with the get_sequential procedure, all this information is store in a temporary list for each cell, and this list is appended to a larger list called latch_cells. Each element in latch_cells is a list that contains all the latch information, or an empty list.

After that, I wrote a procedure called get_pin that stores the pins of each cell. Again, using cell_divisions, I parsed the lines for each cell and look for the string " pin(", which contained a letter(or letter plus a number) representing the name of the pin. I then parsed through the lines after the pin to determine if it was an input or output. I stored the pins for each cell in two separate lists based on if they were an input or output (output is second). A list containing the inputs and output is appended to a larger list called pin_cells, where each element is two lists that represent the input and output pins of that cell.

Then, I combined all the information for each cell into a dictionary called cell_dict. Since all the functions, footprints, sequential information, latch information, and pins were parsed through chronologically, the indexes match with the index of the cell name. For example, if the second cell in the liberty file was called "sky_cell050", all the information of the cell will be in the second index of all these lists created in the previous procedures. In the procedure make cells, every key in cell_dict is the cell name and the value of each key is a list containing all the information that was parsed from the liberty file.

Finally, I wrote the map_cells procedure to determine what kind of gate each cell is. My mentor, Mr. Tim Edwards, and I created a text file called gate_list. This file contained the gate type and its properties to determine if that is the correct gate. By parsing through this file and comparing it to the information in the grand cell dictionary, I was able to identify the properties of non-sequential gates and find what type each cell was, and store it in a new dictionary called gate_type. For Latches and Flip Flops, a naming system was implemented. The base cell type name for Latches and Flip Flops was "LATCH" and "DFF" respectively. Then based on the data collected in get_sequential and get_latch, additional letters are appended to the cell type name. For example, if the value of "clear" is "!R", then you had a letter "S" to the cell type name. The function map_cells returns the dictionary gate_type, where the keys are the cell names from liberty file and the values are the cell's type (AND3, DFFS, LATCH, XNOR2, etc.)

Finally, after mapping the cells, I then created the symbol library of the liberty file. Using KiCad, I drew a symbol library of standard cells. The library information is stored in a KICAD_SYM file, which contains all the pins names and drawing positions of each symbol (Basically contains the directions for KiCad to draw and label a symbol). Using the dictionary created in map_cells, I took that information and found the lines in the KICAD_SYM file that code for the designated symbol, and write those lines to a new KICAD_SYM file for the liberty file I am parsing through. I then substitute the correct cell and pin names into the new KICAD_SYM file. When opening the KICAD_SYM file in KiCad, a symbol library comes up of all the symbols in the liberty file.

Repository

My open source repository is arjunr10/eda-symbol_libraries

(https://github.com/arjunr10/eda-symbol_libraries)

IV. Future

Future improvements I could do is to expand the gate_list text file to contain more cell types. Additionally, the next step would be to automatically output a spice netlist. I am still currently working on creating the symbol library in XSCHEM, but it is the same concept as creating it in KICAD.

ACKNOWLEDGMENT

I would like to acknowledge Mr. Timothy Edwards for mentoring me throughout this project. I would not have been able to develop this program without his guidance. I would also like to acknowledge Mr. Stefan Schippers for creating XSCHEM and KiCad developers team for creating KiCad.

REFERENCES

[1] Stambaugh, W. (n.d.). KiCad. Retrieved September 7, 2020, from https://kicad-pcb.org/about/kicad/*)*

[2] Schippers, S. (n.d.). XSCHEM : schematic capture and netlisting EDA tool. Retrieved September 7, 2020, from https://xschem.sourceforge.io/stefan/index.html

[3] Fairhead, H. (2018, September 28). Introduction to Boolean Logic. Retrieved June 14, 2020, from https://www.i-programmer.info/babbages-bag/235-logic-logic-everything-is-logic.htmlK. Elissa, "Title of paper if known," unpublished.

[4] Hurtarte, J. S. (n.d.). Chapter 6 - Electronic Design Automation. Understanding Fabless IC Technology, 55-64. https://doi.org/10.1016/B978-075067944-2/50007-2

[5] Martins, M. (n.d.). Open Cell Library in 15nm FreePDK Technology. Association for Computeing Machinery. https://dl.acm.org/doi/abs/10.1145/2717764.2717783M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] Microspot. (n.d.). *Electronic Symbol Libraries*. Microspot. Retrieved June 14, 2020, from https://www.microspot.com/products/libraries/electronic-symbol-liraries-2.htm

[7] Miranda, M. (n.d.). Fast and accurate statistical characterization of standard cell libraries. Microelectronics Reliability. https://doi.org/10.1016/j.microrel.2011.05.016

[8] Silvaco. (n.d.). Automatic Configuration From a Liberty File. Silvaco. Retrieved June 14, 2020, from https://www.silvaco.com/examples/accucell/section1/example2/index.html

[9] Simucad. (n.d.). Introduction to cell characterization [Infographic]. https://www.silvaco.com/content/training/Cell_Char_Intro.pdf

[10] Synopsys. (n.d.). Standard Cell Libraries. Synopsys. Retrieved June 14, 2020, from https://www.synopsys.com/dw/ipdir.php?ds=dwc_standard_cell

[11] Tehranipoor, M. (n.d.). Chapter 2 - A Quick Overview of Electronic Hardware. Hardware Security A Hands-On Learning Approach, 23-45. https://doi.org/10.1016/B978-0-12-812477-2.00007-1

[12] Teman, A. (n.d.). Digital VLSI Design Lecture 1. Retrieved June 14, 2020, from http://www.eng.biu.ac.il/temanad/files/2017/02/Lecture-4-Standard-Cell-Libraries.pdf

[13] Terai, H. (n.d.). A single flux quantum standard logic cell library. Physica C: Superconductivity. https://doi.org/10.1016/S0921-4534(02)01759-8

[14] University of California, Berkeley. (n.d.). Liberty Reference Manual (Version 2007.03). Retrieved June 14, 2020, from https://people.eecs.berkeley.edu/~alanmi/publications/other/liberty07_03.pdf

[15] Worthman, E. (2014, April 14). A Guide To Advanced Process Design Kits. Semiconductor Engineering. Retrieved June 14, 2020, from https://semiengineering.com/a-guide-to-advanced-process-design-kits/