An Open Source Alternative to Wire Bonding

1st Pascal Sossou
Science, Math, and Computer Science Magnet House
Poolesville High School
Poolesville, U.S.A.
pascalsossou1@gmail.com

2nd Timothy Edwards
SVP of Analog and Platform
Open Circuit Design, efabless.com
San Jose, U.S.A
tim@opencircuitdesign.com

***Abstract*—Before the end of the millennium, the infrastructure concerned with circuit chip design and development was widely accessible and affordable to many people involved; the relevant software was free and, and chips were cheap to manufacture. However, through the rapid development of electronics in the past twenty years, a vast majority of software and design tools became proprietary. The license costs for these became prohibitively expensive for all except large companies and research institutions, thus confining the chip design industry within the borders of privatized interests. Consequently, many university research programs, small companies, and individuals concerned with chip design have no choice but to work with obsolete technologies and methods. This paper discusses a recent initiative to 'democratize' the chip design industry through software automation. Specifically, we overview a desktop application that automates wire bonding chips to their respective packages, detail its functionality, and discuss the supporting infrastructure from which the application derives its resources from. We will also discuss any relevant and existing open source tools and the shortcomings of the application.**

***Index Terms*—pads, pins, paddle, QFN packages,**

I. Introduction

Integrated circuits (IC's) are sets of electronic circuits placed on semiconductor materials (usually silicon). The integration of a large number of transistors in a small space creates circuits that are faster, cheaper, and more portable than circuits made of discrete electrical components. Traditionally, IC technicians place chips inside packages, and form physical connections between the two through wire bonding, however, large companies employ machines to complete these steps automatically. This allows for their digital communications to occur. However, the process can become extremely complicated and tedious, as without software tools, all planning must be done by hand, often in a 'trial and error' approach.

II. Generating Diagrams

A. QFN Package Layout

The primary objective of the application was to create diagrams for quad flat no-lead (QFN) packages - bidirectionally symmetric packages with a constant number of pins for each of its four sides. Given the number of chip pads, as derived from analysis of a chip's LEF file, the program determines how many pins are required, and then generates an image of a corresponding QFN (Fig. 1). There is a challenge however; some chips have irregular pad designs which introduces complications with wire placement.

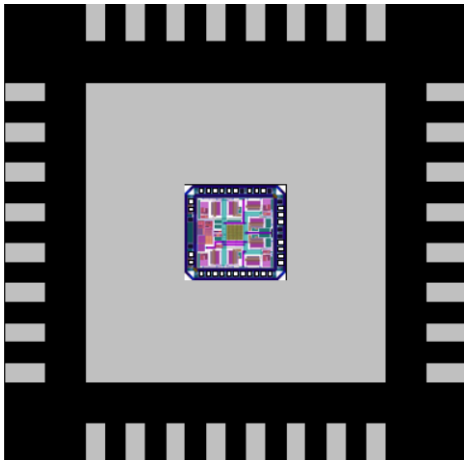

Fig. 1. An example QFN package for the hydra chip

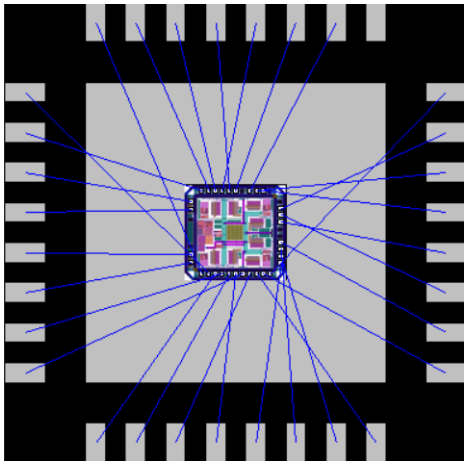

Fig. 2. Initial wire bond solution

B. Wire bonding Algorithm

For chips with simpler and symmetric pads, developing an algorithm is trivial. However, for more unique arrangements, a rudimentary approach fails, and we must adopt a clever method to arrive at a possible solution. The algorithm begins the wire bonding process by iterating the pads of one side of a chip in a center-out strategy. Suppose the list 1,2,3,4,5 represents five pads amongst a particular side of a chip. The

algorithm iterates the list in the order 3,4,2,5,1, where each pad is connected to the closest available pin that has not already been wire bonded. However, we are not done, as with some pad arrangements, we that there exists many crossovers, as observed in Fig. 2. To fix this, we can iterate a list containing all the line objects, and recursively check to see if they intersect with any other lines (before and after swapping), as we elaborate below (Algorithm 1). Employing this strategy, the application arrives at a more presentable solution (Fig. 3)

Algorithm 1 Rectify Crossovers

Require: No two wires intersect
  for $w_1 \in wires$ **do**
    for $w_2 \in wires \wedge w_2 \neq w_1$ **do**
      if $\neg(w_1$ intersects $w_2)$ **then**
        continue
      end if
    else
      swap w_1, w_1
      Algorithm1()
    end for
    return
  end for

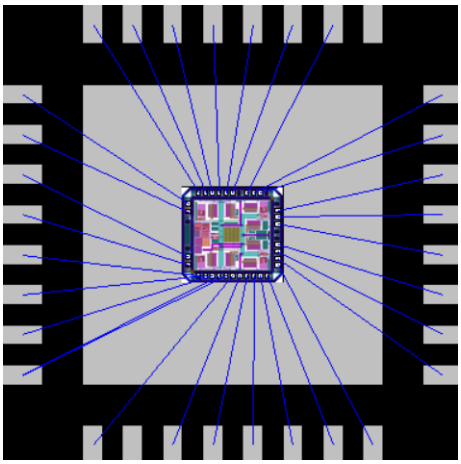

Fig. 3. Final wire bond solution

TABLE I
ALGORITHMIC ACCURACY OF LEGITIMATE WIRE PLACEMENTS

Chip Name	Number of Pads	Percentage of Correct Wire Bond Placements
frequency_divid	12	1.0
hydra	31	.84
strIve	42	.90
strIve2	42	.90
raven	42	.90
ravenna	47	.92
openram$_tc_1kb$	52	.85

C. Editing Tools

After the initial diagram has been generated by the application, the user has a variety of options regarding editing and saving the image produced. The user can drag, shift, and delete any wire, as needed, and also increase the number of pins in the package. Additionally, the user can save the image in the form of various file types including png, jpg, and pdf. The application also checks every individual wire, and colors them red if they are placed in an illegitimate position [2], so that the user may identify and rectify wires as necessary.

D. Algorithmic Efficacy

The construction an algorithm that places wires with a one-hundred-percent accuracy is difficult. There are too many discrepancies and variations, regarding chip dimensions and pad placements, to efficiently account for every single possibility. This may or may not be a problem, as the wire bonder only needs to make minimal adjustments to create a valid diagram. The script most likely can be optimized, but its time complexity also shouldn't be a major concern as the number of wires the application handles is relatively small in size as compared to a setting in which the time required to compute a solution is impractical. Table 1 quantifies the reliability of the algorithm to determine the majority of wire bond placements.

III. SUPPORTING INFRASTRUCTURE

A. Potential Open Source Databases

Google has partnered with Skywater Technology in the recent *Open Source Shuttle Program* initiative, with plans to launch the *FOSS 120nm Production PDK* as soon as November, 2020, the implications of which would allow anyone to create and monetize their own chips.

B. Application Resources

The wire bond application depends primarily on formats produced by the Magic VSLI Layout tool, but could theoretically come from any other tool capable of producing a LEF abstract view of a chip top-level and, although not necessary, a graphical image for a visual reference point for the user.

IV. CONCLUSION

Historically, wire bonding software has remained proprietary [4], and since any information regarding current wire bonding technologies is not available in the public domain, it is difficult to compare the application with any other. Nevertheless, the wire bonding algorithm incontrovertibly outperforms hand-drawn diagrams - shedding hours of time for IC packaging technicians.

REFERENCES

[1] Air Cavity QFN Packages. (n.d.). Retrieved August 30, 2020, from http://www.icproto.com/cap-ic-open-cavity-qfn.html
[2] General Rules for Bonding and Packaging [Manual]. (n.d.). https://d1rkab7tlqy5f1.cloudfront.net/EWI/Onderzoek/Else
[3] An Introduction to Tkiner. (n.d.). Retrieved August 30, 2020, from https://effbot.org/tkinterbook/
[4] New Technologies and New Applications for Wire Bonding. (n.d.). Retrieved September 3, 2020, from https://www.researchgate.net/publication/319847493_New_Technologies_and_New_Applications_for_Wire_Bonding
[5] Packaging Options. (n.d.). The Mosis Service. Retrieved August 30, 2020, from https://www.mosis.com/pages/products/assembly/index
[6] Simlink HFSS. (n.d.). Retrieved August 30, 2020, from http://www.cad-design.com/software/3rd$_party_simulation_links_hfss.html$
[7] Repository: https://github.com/letter108/Wirebonding