Baltic J. Modern Computing, Vol. 9 (2021), No. 1, pp. 145–154
https://doi.org/10.22364/bjmc.2021.9.1.09

A Whole World of Circuitry

Visualizing Integrated Circuits using Geo Information Systems

Jannis STOPPE, Tino FLENKER

German Aerospace Center
Institute for the Protection of Maritime Infrastructures
27572 Bremerhaven, Germany

Abstract Circuit visualisation is crucial to get an overview of the results of hardware synthesis processes. However, visualisation tools are often old, tailored to specific use cases and generally have to deal with the issue of extremely large datasets while providing a smooth and interactive user experience. In order to provide a smooth user experience and smooth learning curve for arbitrarily large circuits, this paper introduces an approach that utilizes workflows and tools from the geoinformation systems community, bridging the gap between integrated circuit synthesis tools and state-of-the-art visualisation tools for large georeferenced datasets.

Keywords: EDA, GIS, Visualisation, Open Source

1 Introduction

The electronic design automation (EDA) industry is a broad umbrella term for companies and products that provide software to design chips. The tasks of EDA are, among others, the creation of circuit diagrams – especially when designing printed circuit boards – but also the design of state machines which are used in sequential circuits to represent internal behavior/memory (Rabaey et al., 2008; Wang et al., 2009).

The automated techniques used for the layout and placement of the integrated circuits (ICs), generate significantly large designs, which developers need to be able to inspect. Therefore, appropriate visualisation tools are crucial, which is the reason why all EDA tools come with various methods to visualise and browse a given design – this functionality is a core part to verify that a design was synthesized as expected.

In order to be able to properly understand a given design, the tools should provide a smooth and ubiquitous user experience – being intuitively usable on any device. However, EDA tools often are quite the opposite, as they are built as large, monolithic, desktop applications and provide only a reasonably responsive user interface because trillions of parts have to be browsed often by the drawing components.

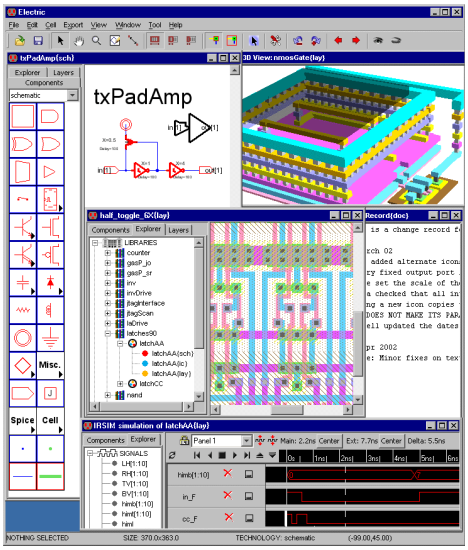

Figure 1. *Electric* (Rubin, 2012) was initially released in 1983, with the latest release (currently 9.07) being from 2016. Its C version was ported to Java in 2005.

Consequently, devices and workplaces are evolving in other directions: mobile computing has long become prevalent, giving people new interaction paradigms while at the same time giving application developers much more limited resources to work with. While other communities have long embraced these not-so-new-anymore developments, EDA tools are still made to be used on large workstations, locking the ability to browse designs in physical places and slowing down processes when people are elsewhere.

In this paper, we use geographic information system (GIS) tools to enable the visualisation of hardware designs that are not exclusively bound to workstations. The purpose is to provide a first proof-of-concept that EDA workflows may benefit from more modern development paradigms, especially when it comes to the ubiquity of tools. This work maps EDA data visualisation to modern GIS tools, allowing designers to browse their chips using state-of-the-art mapping applications from anywhere.

2 Circuit Visualisation

Circuit visualisation software is often old (Fourie and Volkmann, 2013; Fourie, 2018) and sparse. While there are several open source tools available, most of them date back two or more decades, often resulting in issues concerning portability, usability and build processes. Examples of open source software packages that visualise ICs are shown in Figs. 1 through 4.

While there are more commercial tool sets than open source solutions (such as EAGLE, CADSTAR, Allegro and more), they are often tied to licenses that are unsuitable for academic usage and/or quite expensive. Generally though, they often share the

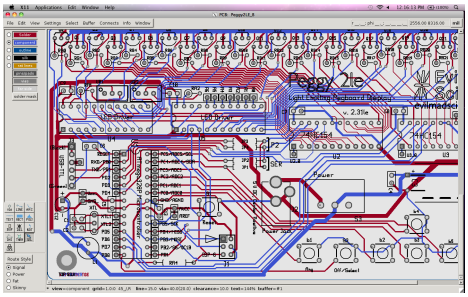

Figure 2. *gEDA* (Brorson, 2006) was initially released in 1998, with the latest release (1.8.2) being from 2013.

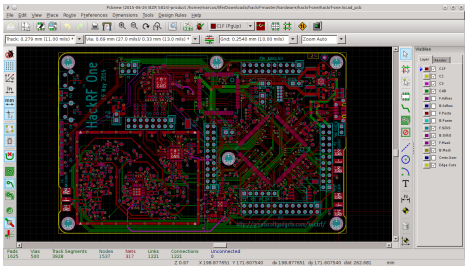

Figure 3. *KiCad* (*KiCAD*, 1992) was inirially released in 1992 but remains active to this day, with the latest release being a mere month old at the time of writing and its code repository showing consistent activity.

same shortcomings – usually being updated more regularly (as long as they are commercially viable and not discontinued) but with the additional point that some (such as e.g. AutoCAD) need to cover essentially all use cases of computer-aided design, far beyond the responsive and intuitive display of very large scale integration designs, resulting in steep learning curves and feature sets far beyond an individual designer's use case.

The bottom line is: while there are tools that at least visualise circuits as part of their (sometimes much broader) spectrum of available capabilities, the development – *especially of the open source solutions* – is slow at best or got stuck at worst. The tools are often part of larger, heavier application suites that include simulation engines, design tools, synthesis flows etc. Cross-platform often means that these heavyweight applications run on Windows and Linux, with no way to e.g. easily browse these designs on more modern portable devices. At the same time, porting these visualisation engines to more modern technologies is not a simple task, with each of them solving the task of drawing potentially trillions of parts (more or less) responsively in its own, special way.

With a handful of companies de facto dominating the market of EDA tools, there is not much competition to innovate and while open source tools may not exactly be stuck in dead-ends, the community may just be too underpopulated to be able to quickly pick up current technologies. Even simple common schemes such as user interactions that

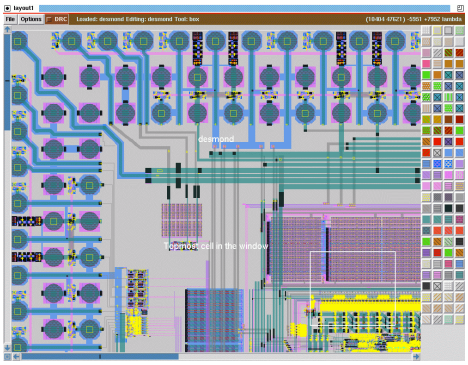

Figure 4. *Magic* (Ousterhout et al., 1985) was initially released in 1980, with the latest release being from 2017.

have been established over the last decade are often disregarded in tools that have been around much longer than that – with even basic interactions such as zooming out or dragging the viewport requiring the designer to read the documentation.

3 Geo Information Systems

Geographic information systems are an active field of both, research (Longley et al., 2010; Stefanakis, 2017; Netek et al., 2020) and commercial activities.

The rise of smart phones and location-based services has resulted in a large mass market for mapping applications and a broad variety of both, big and small applications and companies in both, commercial and open source communities.

Clients themselves are more short-lived and development cycles are faster in the GIS than in the EDA community. That implies that technologies are usually more recent and adapt more state-of-the-art approaches than applications that carry decades of history around. More importantly though, the short-livedness of the GIS tools has made a strict separation of concerns a necessity: servers that prepare and pass on the required data are less in numbers than the various front-ends that display it, but are in fact (mostly) exchangeable. These servers solve issues like translating a broad range of data with plethoras of different coordinate systems into data that can easily be processed by clients.

This smallest denominator – the prevalent separation of concerns in a server that distributes map data and a client (with potentially very limited resources) to consume and display it – separates how most GIS tools work from most EDA tools around. The data that is being passed to the clients mostly consists of "tiles": square-ish parts of the world map at particular zoom levels. While there are various formats available up to 3D models of terrain and cities (Kolbe et al., 2005), a lot of services merely hand out raster graphics for given areas.

Open source servers such as geoserver (Iacovella, 2017), mapserver (Vatsavai et al., 2006) or deegree (Fitzke et al., 2004) allow geographical services to be hosted. Likewise, open clients such as NASA's WorldWind (Bell et al., 2007), cesium (Cozzi and

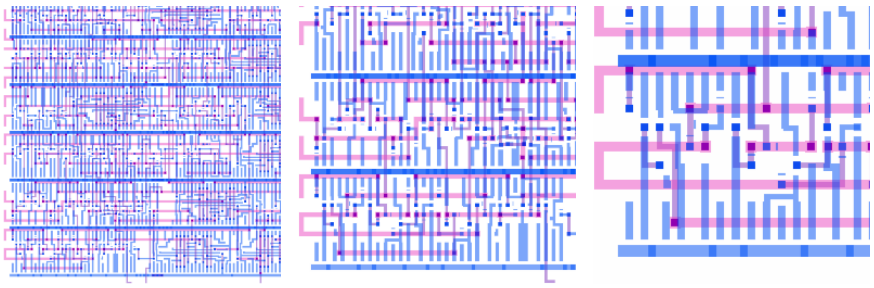

Figure 5. Pre-rendered tiles of circuits as they are generated and stored on the GIS back-end, zoom levels 1, 2 and 3 (respectively from left to right).

Bagnell, 2013) or leaflet (Crickard III, 2014), among others, allow the data that is served by those to be displayed and handled.

4 Approach and Workflow

This section first briefly describes our motivation for this approach and then goes into the workflow. It also goes into more detail about which tools were used and in what way.

With GIS tools being a more active field, and especially clients adopting new technologies (especially the web and mobile devices), this work illustrates how GIS technology can be used to assist developers in browsing their designs. This technology essentially **solves the same problem as EDA visualisation tools** – allowing users to browse large, location-referenced data sets that cannot be shown in their entirety due to both, their size and their detail.

As designers have usually set up their tools the way they prefer them, the approach should pick up these settings if possible. Instead of creating another new, incomplete solution, the chosen approach is to utilize the existing tools to generate the data that GIS back-ends require to drive the respective front-ends.

These pre-rendered tiles consist of various zoom levels that are transferred to the mapping tools as required. Examples of different resolutions of a part of a small counter circuit are shown in Fig. 5.

The approach currently utilizes the slippy map tiles standard (like Google or OpenStreetMap), which is usually used to store and deliver square raster map tiles from 85.0511 degrees north to south. The chosen EDA tools are used to precompute these tiles in the required format. In this way, the raster tiles can be easily handed out to arbitrary clients which – being web-based, desktop application, or any other platform – can be used to browse through the circuits.

For the proof-of-concept the following tools were used:

- *Qflow* (*Qflow*, 2013) for synthesis and layouting,
- *Magic* (*Magic*, 2013) for IC visualisations and generation of the raster tiles,
- *apache* (*Apache*, 1995) webserver for providing raster tiles to clients and

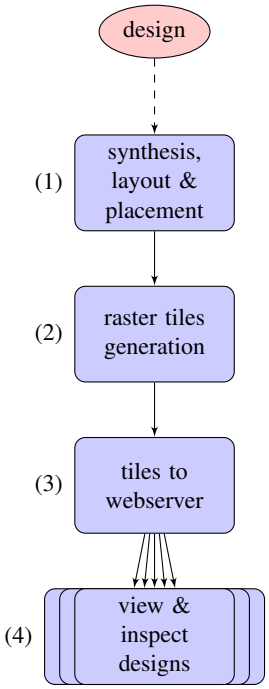

Figure 6. Suggested workflow to explore IC designs with GIS.

– *leaflet* (*Leaflet*, 2011) to access and browse through the circuits' raster tiles.

The whole workflow is illustrated in Fig. 6. It relies solely on the combination and scripting of off-the-shelf open-source tools and should thus be easily reproducible and extensible.

1. The tool *Qflow* (*Qflow*, 2013) is open source, which provides a workflow for digital synthesis. It uses hardware in a high-level behavioral language and turns this into a physical circuits layout. Input data are hardware descriptions in Verilog. This is synthesized into a lower level of abstraction (transistor level). Afterwards, a layout and placement is done for the newly received hardware description. Here it is determined where in the real chip the transistors and lines are placed. The result is used by the next tool.
2. The open source tool *Magic* (*Magic*, 2013) is available for the visualisation of hardware descriptions that have been already placed and laid out. *Magic* can also zoom into the hardwares visualisation and create screenshots. Our approach uses raster tiles, since *Magic* only generates images and does not provide a way to get vector tiles. The designs are usually depicted with a quadratic shape and are therefore easily transferable to the **Mercator** representation of map applications. Furthermore, *Magic* can be executed by scripts and thus performs the generation of the tiles. The script specifies the position and zoom level of the respective tile and helps to easily create a comprehensive set of tiles on multiple levels.

3. Next, we need a server that provides the generated tiles to the user as easily as possible. A web server is well suited for this purpose, which usually works with the slippy map tiles by default. We used *Apache* (*Apache*, 1995) for our proof-of-concept. In order to access the tiled visualisation, the generated tiles only had to be placed in the directory intended for web pages.
4. Finally, for accessing and viewing the hardware visualisation, a web page has to be created that can cope with the map data. For this task the tool *Leaflet* (*Leaflet*, 2011) – widely known in the mapping community – was used.

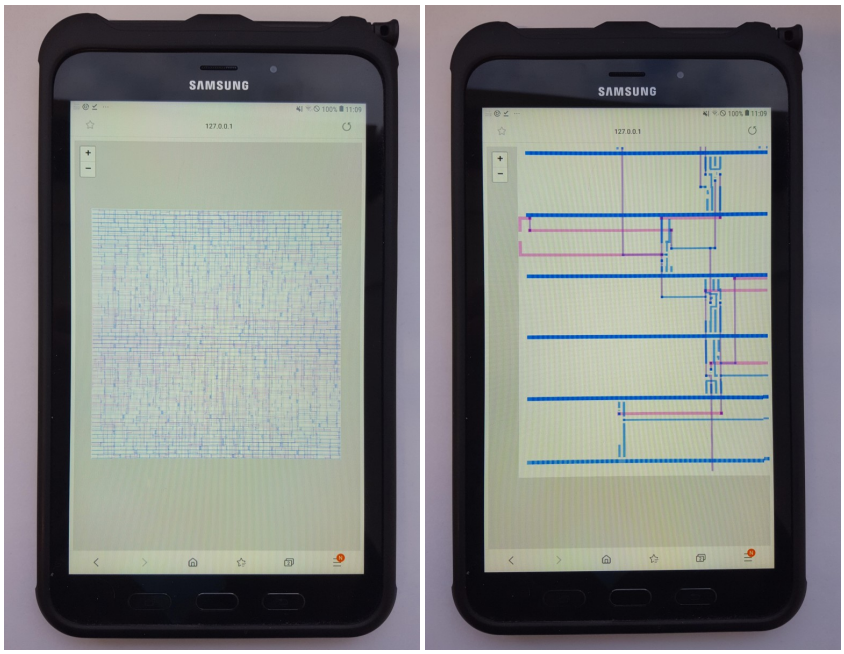

Figure 7. An adder circuit on a mobile device

5 Results

The resulting visualisation can be accessed from any connected device with a web browser that can access the server. It generally results in a smooth experience even on mobile devices with very limited resources. Examples of a visualisation of an adder from the École polytechnique fédérale de Lausanne (EPFL) benchmark set can be seen in Fig. 7.

The major advantage of this technology is that it outsources the computationally intensive parts into the synthesis flow, allowing designers to quickly and easily browse the circuits from anywhere, as long as the device has a good enough connectivity. This

allows designers to check results of synthesis flows from a train, collaborate with colleagues on bug finding from their home or check oddities in the design from a hotel – as long as a connection to the hosting system is available.

The system itself is a proof-of-concept that lacks a lot of functionality. While mapping systems support the selective display of layers, they are not exported yet. Also, the system does not support any functionality besides viewing the circuit yet – no elements can be selected or investigated further in any way. Still, the experience itself may already be considered a step into a new direction of browsing designs ubiquitously.

The most obvious room for improvement would be to generate required tiles on-the-fly instead of precomputing them. This is because the generation of raster tiles, especially in the higher zoom levels, takes a lot of time. The on-the-fly generation strips off the tedious pre-processing step. Only the tiles requested by clients must be generated. This can cause delays, especially with the first calls, but it allows faster access to the service for viewing a new circuit design.

6 Conclusion

This work uses open source software connecting tools from the EDA and GIS communities to visualise IC designs. The EDA tools are used for raster tile generation whereas the GIS tools display the visualisations. Our aim was to make the visualisation of an ICs design easily accessible, especially on mobile devices, so we have proposed the simplest approach with examples to achieve this.

References

Apache (1995). https://httpd.apache.org/. Accessed: 2020-09-01.

Bell, D. G., Kuehnel, F., Maxwell, C., Kim, R., Kasraie, K., Gaskins, T., Hogan, P., Coughlan, J. (2007). Nasa world wind: Opensource gis for mission operations, *2007 IEEE Aerospace Conference*, IEEE, pp. 1–9.

Brorson, S. (2006). Circuit design on your linux box using geda, *Linux Journal* **141**(7).

Cozzi, P., Bagnell, D. (2013). A webgl globe rendering pipeline, *GPU Pro 4: Advanced Rendering Techniques* **4**, 39–48.

Crickard III, P. (2014). *Leaflet. js essentials*, Packt Publishing Ltd.

Fitzke, J., Greve, K., Müller, M., Poth, A. (2004). Building sdis with free software–the deegree project, *Proceedings of GSDI-7, Bangalore, India*.

Fourie, C. J. (2018). Digital superconducting electronics design toolsstatus and roadmap, *IEEE Transactions on Applied Superconductivity* **28**(5), 1–12.

Fourie, C. J., Volkmann, M. H. (2013). Status of superconductor electronic circuit design software, *IEEE Transactions on Applied Superconductivity* **23**(3), 1300205–1300205.

Iacovella, S. (2017). *GeoServer Beginner's Guide: Share Geospatial Data Using Open Source Standards*, Packt Publishing Ltd.

KiCAD (1992). https://kicad-pcb.org/. Accessed: 2020-09-01.

Kolbe, T. H., Gröger, G., Plümer, L. (2005). Citygml: Interoperable access to 3d city models, *Geo-information for disaster management*, Springer, pp. 883 – 899.

Leaflet (2011). https://leafletjs.com/. Accessed: 2020-09-01.

Longley, P. A., Goodchild, M., Maguire, D. J., Rhind, D. W. (2010). *Geographic Information Systems and Science*, 3rd edn, Wiley Publishing.

Magic (2013). http://opencircuitdesign.com/magic/. Accessed: 2020-09-01.

Netek, R., Masopust, J., Pavlicek, F., Pechanec, V. (2020). Performance testing on vector vs. raster map tiles – comparative study on load metrics, *ISPRS International Journal of Geo-Information* **9**(2), 101.

Ousterhout, J. K., Hamachi, G. T., Mayo, R. N., Scott, W. S., Taylor, G. S. (1985). The magic VLSI layout system, *IEEE Design & Test of Computers* **2**(1), 19–30.

Qflow (2013). http://opencircuitdesign.com/qflow/. Accessed: 2020-09-01.

Rabaey, J. M., Chandrakasan, A., Nikolic, B. (2008). *Digital Integrated Circuits*, 3rd edn, Prentice Hall Press, USA.

Rubin, S. M. (2012). Using the electric vlsi design system.

Stefanakis, E. (2017). Web mercator and raster tile maps: two cornerstones of online map service providers, *Geomatica* **71**(2), 100–109.

Vatsavai, R. R., Shekhar, S., Burk, T. E., Lime, S. (2006). Umn-mapserver: A high-performance, interoperable, and open source web mapping and geo-spatial analysis system, *International Conference on Geographic Information Science*, Springer, pp. 400–417.

Wang, L.-T., Chang, Y.-W., Cheng, K.-T. T. (2009). *Electronic Design Automation: Synthesis, Verification, and Test*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Appendix: *Magic* Script to Automate Tile Extraction

```
set viewbox [view bbox]
box values [expr [lindex $viewbox 0]/4] [expr [lindex $viewbox 1]/4] [expr [lindex
      $viewbox 2]/4] [expr [lindex $viewbox 3]/4]

set depth 7
set startDepth 0

proc pow {arg1 arg2} {
        if {$arg2 == 0} {return 1}
        set result $arg1
        for {set i 1} {$i < $arg2} {incr i} {
                set result [expr $result*$arg1]
        }
        return $result
}

proc squareBox {} {
        if {
                [box height] < [box width]
        } then {
                box height [expr [box width]/4]
        } elseif {
                [box height] > [box width]
        } {
                box width [expr [box height]/4]
        }
}

view

for {set i $startDepth} {$i < $depth} {incr i} {
        set boxDivider [pow 2 $i]
        box values [expr [lindex $viewbox 0]/[expr 4*$boxDivider]] [expr [lindex
             $viewbox 1]/[expr 4*$boxDivider]] [expr [lindex $viewbox 2]/[expr 4*
             $boxDivider]] [expr [lindex $viewbox 3]/[expr 4*$boxDivider]]
        box position [expr [lindex $viewbox 0]/4] [expr [lindex $viewbox 1]/4]
        squareBox
        puts "Box size is now [box values]"

        for {set j 0} {$j < [pow 2 $i]} {incr j} {
                for {set k 0} {$k < [pow 2 $i]} {incr k} {
                        puts "Rendering zoom level $i at $j / $k"
                        set stepwidth [box width]
                        box position [expr [lindex $viewbox 0]/4+[expr $stepwidth
                             /4*$j]] [expr [lindex $viewbox 1]/4+[expr $stepwidth
                             /4*$k]]
                        set inverseYIndex [expr [pow 2 $i]-$k-1]
                        plot pnm $i-$j-$inverseYIndex 256
                }
        }
}
```

Received December 18, 2020, revised February 19, 2021, accepted February 24, 2021