Incremental Improvements on the Placement and Routing of Minecraft Redstone Circuits

Quan Nguyen
Massachusetts Institute of Technology
qmn@mit.edu

ABSTRACT

We present DEWEY, the successor to the PERSHING [5] place-and-route tool for Minecraft Redstone circuits. As a performance-oriented rewriting of PERSHING from the ground-up in C, DEWEY implements greatly improved routines for standard cell placement and routing, and leaks a ton of memory. However, we materialize a speedup of up to 34.0× over previous work. DEWEY shows tremendous strides towards the placement and routing of entire computer processors in *Minecraft*.

ACM Reference Format:
Quan Nguyen. 2018. Incremental Improvements on the Placement and Routing of Minecraft Redstone Circuits. In *Proceedings of Special Interest Group on TBD (SIGTBD'18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Demands on computer architects have increased substantially over the last several decades, to squeeze ever-diminishing gains from ever-shrinking feature sizes. Device designers and computer architects alike have demonstrated tremendous creativity in the pursuit of raw computational power and energy efficiency in light of the constrained art that is physical design. However, hardware design comes with some substantial drawbacks: lengthy hardware prototyping, high capital costs, and potentially cumbersome licensing restrictions.

We contend that Minecraft, a game popular with young children and the young-at-heart, can be used to bridge the gap between a diminishing supply of computer architects and the challenges of tomorrow. With the Redstone Update, Minecraft introduced components that behave much like digital circuits. However, all Minecraft circuits are built fully by hand; this is the equivalent of full-custom design in the VLSI world. We seek to alleviate the trouble and complexity of full-custom Minecraft circuit design by building an automatic place-and-route tool for Minecraft circuits. The current state-of-the-art, PERSHING, was written in Python and accomplishes this task, albeit slowly and with some crucial errors that make it difficult to use in practice. In this paper, we improve on PERSHING, namely, in speed.

SIGTBD'18, April 2, Cambridge, MA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The reader is referred to [5] for a more detailed description of redstone logic. We improve on the state of the art with a completely re-written implementation of PERSHING in C, called DEWEY[1]. We take advantage of manual memory allocation (in fact, to the point of abuse) to maximize performance.

2 THE MINECRAFT PHYSICAL DESIGN FLOW

The process by which a Verilog description of a circuit becomes a fully placed-and-routed circuit largely follows that of prior work. The Verilog must be converted to a netlist with a digital synthesis tool, like Yosys [9], which outputs a file in the Berkeley Logic Interchange Format (BLIF). Combined with a standard cell library expressed in terms of Minecraft blocks, the placer arranges the standard cells to minimize wire lengths subject to constraints on the design. The router then routes nets connecting standard cells to each other and the top-level input and output pins. Extraction, among other things, transforms the design into a workable circuit, complete with timing information and the actual block layout. The extracted design then can be placed in a Minecraft world for *in-situ* testing or can be used as a "black-box" module as part of a larger design.

2.1 Synthesis

Once designers produce the Verilog description of a synthesizable hardware circuit– perhaps generated automatically with Chisel [2] or Bluespec [1] – they can feed it to Yosys [9], an open-source hardware synthesis tool. For an example, we will synthesize, place, and route a four-bit counter. Figure 1 displays the corresponding Verilog.

```
module counter (clk, rst, en, count);

   input clk, rst, en;
   output reg [3:0] count;

   always @(posedge clk)
      if (rst)
         count <= 4'd0;
      else if (en)
         count <= count + 4'd1;

endmodule
```

Figure 1: Verilog source code for a 4-bit binary counter, from http://www.clifford.at/yosys/screenshots.html.

In conjunction with a supplied file in the Liberty format, which expresses not only the functionality of the available logic gates

[1]George Dewey is the only person to attain the rank of Admiral of the Navy in the United States [7]. A contemporary of John J. Pershing [8], General of the Armies, and the namesake of previous work, Dewey can be seen as the "sea" version of Pershing.

but also the costs (*e.g.* area, delay) of using them, Yosys produces a netlist in the form of a BLIF. This becomes one of the inputs to DEWEY.

Although it is not a part of our contribution, Yosys is an essential tool for future work: because DEWEY directly uses its output with no modification, all optimizations to the circuit at the logic level must be performed here. Yosys can help reduce the size of a design by inferring complex yet compact cells rather than logically-equivalent standard cells of more elementary functionality[2]. Thus, it is the prerogative of the standard cell library writer to create compact and efficient cells that represent common functions desired by hardware designers.

2.2 Placement

DEWEY accepts two main inputs: a BLIF file, which specifies the standard cells used and the nets used to connect them, as well as a library file, supplied as a YAML file. The library file contains descriptions of the standard cells made available to Yosys, with the addition of the arrangement of the Minecraft blocks comprising these cells.

We use the TimberWolf placement algorithm [6] to produce the best placements through simulated annealing: a locally-optimal placement, in the early stages, can be abandoned for a non-optimal placement in search for a global maximum.

In determining the score, we consider several factors: the estimate of the wire lengths used to connect all nets (based on the minimum spanning tree), the design size, the spread of the design from its center of mass, the squareness of the design, and the overlaps between individual cells. Of these, only cell overlaps will cause violations and force the placer to continue. Placement typically completes in fewer than 1,000 iterations, each of which consisting of 100 generations of modifications, which include displacing cells, rotating them, or interchanging the position of like cells.

A major performance loss occurs in the prior work's version of the placer because it produces an extracted version of the placement before determining its score. Repeated allocations of memory are time-consuming, and also completely unnecessary: thanks to Euclidean geometry, we can directly compute the overlaps knowing only the dimensions of the cells and their placements. Space-saving optimizations like these greatly increase the placer's speed. Figure 2 shows the placement of a four-bit counter.

2.3 Routing

Once placements are determined, the router uses a modified form of Lee's algorithm [3] to route nets between the pins of the netlist. In particular, we adapt the modified Lee's algorithm devised by Silk [4] to our purposes.

We initialize a design based routing all pins together based on the minimum spanning tree produced by Kruskal's algorithm, and connect them using Manhattan routing. Then, we rip-up nets with violations or poor scores and re-route them. Each net is composed of several segments, which may connect standard cell pins or segments together. The routing process is as follows:

[2]Technically, we need only a NAND gate, as it is a universal logic gate. But we are in the business of *good* logic.

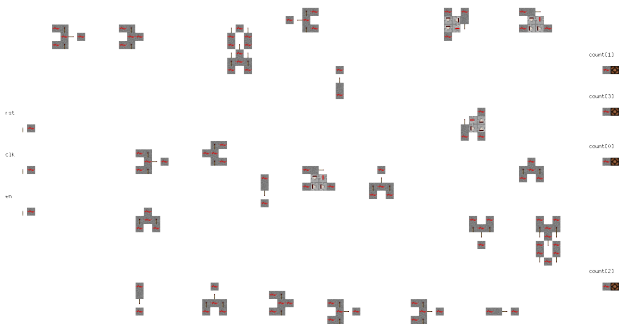

Figure 2: Completed placement of a four-bit counter.

(1) For each pin, or already-routed segment, initialize a matrix marking these locations as START – the origin. Create a priority queue containing locations to explore next paired with the cost of exploring that new location.
(2) Propagate, in a style similar to breadth-first search, from the origin until we reach the wavefront formed by a different pin or segment. In a separate backtrace matrix, record the path from every location explored back to the origin.
(3) Using the backtrace matrices of the two different groups, create a new segment connecting the two groups' origins together. Note that a segment may connect to any other part of any other segment.

It is possible to introduce violations, perhaps by crossing over another already-placed net, in order to complete routing. This is acceptable as long as the net will eventually be routed to remove all violations. Special considerations are made for vertical signal transmission, which impose special requirements on routed nets. Prior work failed to address this, and we are happy to report that DEWEY's router correctly handles these cases by introducing violations in the relevant cases.

We also speed up routing here: we avoid the allocation of large matrices wherever possible. The actual path of a net is expressed as an array of backtraces instead of a matrix with the block locations filled in. This also has the added benefit of making displacement extremely easy: simply modify the start and end points of a given segment.

2.4 Extraction

Finally, with routing completed, the extraction process produces the actual blocks necessary to build the circuit in Minecraft. By performing a depth-first search from the driving pin (of which each net has exactly one), we can determine not only the delay of a net, from source to sink, but also any needed buffering. Buffering is achieved by placing redstone "repeaters", which buffer the signal but add additional delay. With signal buffering complete, we can compute the maximum frequency $f_{\max}$ at which this circuit can operate. Figure 3 shows a completed routing and extraction of a four-bit counter.

As the final output, the extraction process produces a matrix of the actual block placements needed to build this circuit in the

Minecraft world. PERSHING provides tools to actually place these blocks in a game save format.

Figure 3: Extracted design of a four-bit counter.

3 PERFORMANCE RESULTS

DEWEY's performance was measured using the `time` utility, using the same environment as PERSHING: a 2013 MacBook Pro 2.5 GHz quad-core with 16 GB RAM. DEWEY was compiled with the `-O3` option using `clang` version 8.0.0.

DEWEY is much faster than prior work in this field; see Table 1 for details. The author in [5] uses markedly inferior programming to perform placement and routing.

As DEWEY is still in development, we are currently unable to compute critical delays, and therefore, the $f_{\max}$ of a given circuit. However, we expect them to be vastly faster than results produced by PERSHING by the simple fact that our designs tend to be smaller.

				Completion Time (s)		
Design	Cells	Nets	Volume	DEWEY	Prior Work	Speedup
2-input RS flip-flop	2	4	$5 \times 20 \times 21$	2.2	15	6.8×
2-bit adder with carry	7	11	$7 \times 31 \times 37$	8.8	117	13.3×
4-bit counter	23	26	$7 \times 52 \times 77$	31.9	1084	34.0×

Table 1: DEWEY performance, including speedup over prior work.

4 CONCLUSION

We made PERSHING a lot faster. This brings a considerably larger variety to the kinds of circuits we can synthesize in Minecraft. With parallelization – a feature kept in mind while DEWEY was being written – we can route even larger circuits with ease.

5 ACKNOWLEDGEMENTS

We would like to again thank the efforts of the reviewers and the program committee. The authors would also like to thank their advisers, who are still unaware this work was happening despite the amount of *real* research work to be done. The work of the authors is unofficial, not from Minecraft or its creator, Mojang, and not endorsed by or otherwise approved by Mojang.

REFERENCES

[1] [n. d.]. Tutorial: Bluespec SystemVerilog: Efficient, Correct RTL from High-Level Specifications.
[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, 1216–1225.
[3] C.Y. Lee. 1961. An Algorithm for Path Connections and Its Applications. *Electronic Computers, IRE Transactions on* EC-10, 3 (Sept 1961), 346–365. https://doi.org/10.1109/TEC.1961.5219222
[4] Youn-Long Lin, Yu-Chin Hsu, and Fur-Shing Tsai. 1989. SILK: a Simulated Evolution Router. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 8, 10 (Oct 1989), 1108–1114. https://doi.org/10.1109/43.39072
[5] Quan Nguyen. 2016. *PERSHING: An Automatic Place-and-Route Tool for Minecraft Redstone Circuits.*
[6] C. Sechen and A. Sangiovanni-Vincentelli. 1985. The TimberWolf Placement and Routing Package. *Solid-State Circuits, IEEE Journal of* 20, 2 (April 1985), 510–522. https://doi.org/10.1109/JSSC.1985.1052337
[7] Wikipedia. 2018. George Dewey — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/George_Dewey
[8] Wikipedia. 2018. John J. Pershing — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/John_J._Pershing
[9] Clifford Wolf. [n. d.]. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/. ([n. d.]).