PREPRINT COPY – Generated December 17, 2016

PERSHING: An Automatic Place-and-Route Tool for Minecraft Redstone Circuits

Quan Nguyen
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02138
qmn@mit.edu

Abstract

In *Minecraft*, circuits can be assembled from redstone components. We introduce PERSHING, a place and route tool that can transform a gate-level netlist into logic gates built of redstone in Minecraft, without the need for external Minecraft mods. We discuss challenges, such as vertical signal transmission, and the algorithms used to perform placement and routing. We show preliminary results, including completed layouts, note future paths to explore, and conclude.

1. Introduction

With the *Redstone Update* [1], *Minecraft* [2] introduced redstone, a block that allows the transmission of signals not unlike modern electronic circuits. These wires can be driven with redstone torches, which, in some configurations, can act as a diode or as an inverter. Redstone "repeaters" act as the circuit equivalent of a buffer, and "comparators" allow two signals' strengths to be compared. By exploiting the properties of these blocks, players can construct elementary logic gates computing functions such as AND, OR, and XOR. Minecraft players have built fantastically complex circuits, including multi-digit combination locks [3], a 16-bit ALU [4], and even full-blown computers that run the Bresenham line-drawing algorithm [5].

However, these circuits must be painstakingly built by hand. The user must place the gates, and route redstone between these points. This is especially complex because gates may be of any size, and may be anywhere in a three-dimensional space. The difficulty of planning, placing, and routing extremely complex redstone circuits impedes the development of more complex computers in Minecraft.

Individually placing blocks, redstone wires, torches, and repeaters is tantamount to full-custom layout in the VLSI community. Furthermore, players lose the benefit of modularization and hierarchical design, because they may re-engineer, much less repeat the placement of, existing circuits. Effective algorithms for automating the design and placement of these circuits have existed since the 1980s, but have yet to be incorporated in Minecraft.

Finally, circuit design has advanced considerably from hand layout to the use of standard cells and hardware description languages (HDLs). HDLs, like Verilog and VHDL, allow designers to express the functionality of a circuit without having to specify the exact logic gates and their connections.

In this paper, we introduce the *PlacE RedStone Hardware IN Game* (PERSHING)[1] program, which accepts a description of a circuit as its input a BLIF file and a standard cell library, and creates a placed-and-routed Minecraft redstone circuit, ready to be placed in any Minecraft world. This tool completes only with the use of *vanilla* Minecraft blocks – that is, without the use of external Minecraft modifications or blocks.

2. Previous Work

There have been several attempts to build Minecraft redstone circuit synthesis tools. GitHub user Disasm created VLSI Craft, a place and route tool for Minecraft [7]. However, his tool requires the user to install the ComputerCraft and redlogic mods. The latter mod introduces logic gates that fit in a single block. Furthermore, it allows wires to travel in arbitrary directions, which conventional redstone propagation does not permit. Disasm's work, while novel, is not supported by Minecraft out-of-the-box. This paper's solu-

[1] The MGM-31A Pershing ballistic missle system succeeded the United States' Redstone ballistic missle system in the 1960s [6].

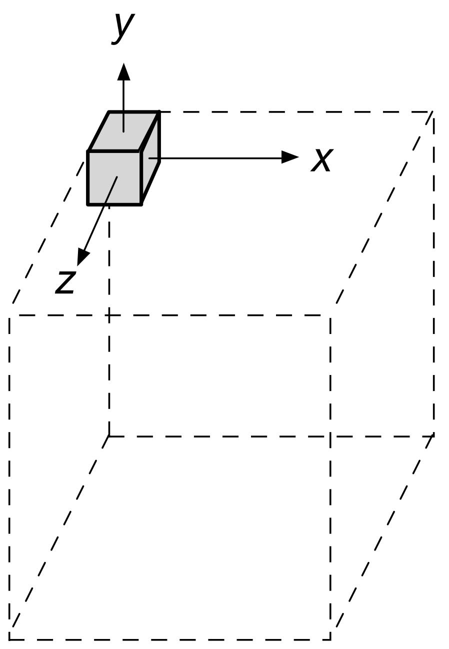

Figure 1. The layout of a chunk in Minecraft, based on the description and drawing from [9].

tion is supported by "vanilla" Minecraft; that is, one without external modifications installed.

In 2014, Karl Kroening created RedGen [8], a "redstone logic generator." However, as of its last code commit in June 2014, the project appears incomplete – featuring just a maze router. This work presents the most complete effort known to its authors to produce an end-to-end place-and-route solution.

3. The Minecraft World

In Minecraft, the entire world is discretized into blocks one meter on a side. As a result, the world can be nearly completely characterized by a three-dimensional grid. A world is limited to a maximum vertical height of 256 blocks (256 meters), but is practically unlimited in the lateral directions. To represent the world, Minecraft divides the world into "chunks" that are 256 blocks tall, and represent a portion of the world 16×16 blocks of the surface.

Each location in the world is uniquely specified by a 3-tuple (y, z, x), where y increases vertically, z increases to the south, and x increases to the east. This has the convenient property that linearly increasing x, then z, then y, moves through the blocks as one reads a book: first across the page, then down the lines, and then through the pages [9]. Figure 1 pictorally demonstrates the layout of a chunk.

Each location in the Minecraft world is occupied by exactly one block, whether stone, wood, redstone dust, pistons, or other mechanisms. The discretization of the world into units assists in block placement, but complicates signal transmission, as discussed in Section 3.1.

3.1 Redstone

Redstone *dust* can transmit signal in each of the compass directions: north, east, south, and west. It may also transmit a signal up or down one meter, as long as it is not obstructed. Redstone dust automatically reconfigures itself to connect to neighboring redstone components, including repeaters, comparators, torches, switches, and other redstone dust.

3.2 Redstone Propagation

Redstone signals have strengths, represented as an integer between 0 and 15, inclusive, and the strongest signal has the value 15. Each additional meter the signal travels reduces the signal strength by one unit until it reaches zero. As a result, redstone has finite propagation (something perhaps akin to RC delays) which requires occasional buffering. A redstone signal may travel as far as 15 units before it completely attenuates. Therefore, a redstone mechanism may be powered by as far as 15 meters away from the source of the signal. However, as long as an unbuffered redstone signal is no more than 15 meters away, it may power as many mechanisms as can be connected to it; redstone does not exhibit problems with fan-out.

A redstone repeater, which acts as a signal buffer, boosts any non-zero redstone signal to its full strength and adds a delay between one to four "redstone ticks." A 'redstone tick' is the unit of time for redstone propagation as used in Minecraft, and, when there is no lag, corresponds to 0.1 second in real time [10].

3.3 Vertical Signal Transmission

In general, no circuits but the most trivial can be routed in two dimensions. As a result, redstone signals must be routed over or under other redstone signals. As previously mentioned, redstone signals can vertically travel one block up or down. To avoid intersecting other redstone nets or logic gates, a redstone signal must remain at least one block away from other blocks. As a result, long ramps would be necessary to raise and lower these signals. Fortunately, torches can be arranged to transmit signals upward, as in Figure 2. For all practical purposes, redstone torches, as vias, can be infinitely tall.

However, torch towers cannot transmit signals *downwards*. Instead, sticky pistons can be used to move a block of redstone up and down in a manner similar to a mechanical relay. As depicted in Figure 3, when a signal reaches the piston, the piston extends, forcing a redstone block down to the redstone dust below, powering the circuit. When the signal is de-asserted, the sticky piston retracts, taking the redstone block with it. Downwards vias may be up to 12 blocks tall; this is the maximum number of blocks a piston can push. Any intervening blocks must be slime blocks.

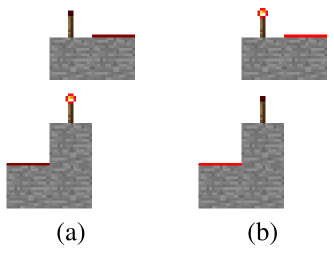

Figure 2. A redstone torch tower can transmit a redstone signal vertically upwards. (a) Unpowered. (b) Powered.

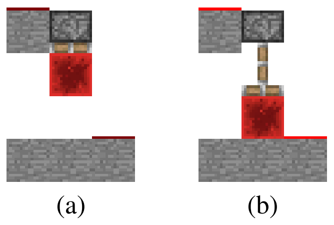

Figure 3. A sticky piston can transmit a redstone signal vertically downwards. (a) Unpowered. (b) Powered.

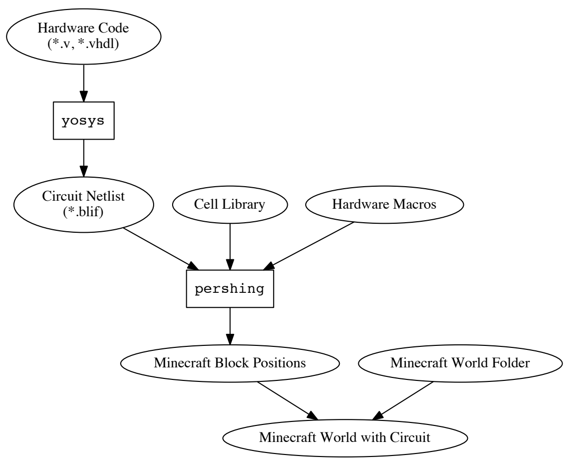

Figure 4. Flow chart depicting automatic design process for Minecraft redstone circuits.

Both of these mechanisms have the inherent drawback that signals may only travel in one direction. However, as long as the nets have only one driver, there is always a directed path from the driver to all of the inputs.

4. Design

As introduced earlier, PERSHING can turn hardware code into a complete Minecraft redstone circuit in the process depicted in Figure 4.

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

4.1 Logic Synthesis

PERSHING uses Yosys [11] to synthesize HDL code into the gates needed to construct the circuit. Yosys generates a Berkeley Logic Interchange File (BLIF) which specifies the logic gates and connections.

4.2 Cell Libraries

A cell library informs PERSHING of the shape and configuration of a Minecraft logic gate, in addition to the blocks required to assemble it. It also specifies the logic function the gate computes. We created an initial cell library, used by default in PERSHING, but it can be expanded to accommodate alternative or improved designs for logic gates[2]. An example design is shown in Figure 6. PERSHING uses the YAML representation directly; a parser converts the YAML into the Liberty format for Yosys.

4.3 Placement

The placement algorithm in PERSHING is based on the placement algorithms in Timberwolf [12]. From an initial placement, PERSHING rearranges the logic gates to minimize wire length and violations. Violations occur when a portion of one logic gate contacts or overlaps another logic gate. Because Minecraft logic gates do not require power rails (*i.e.*, V_{dd} and V_{ss}/GND), it is not necessary to align the cells to a grid, or have them face the same orientation. Logic gates may be altered by displacement, rotation, or exchange with another logic gate.

PERSHING uses the simulated annealing algorithm discussed in the Timberwolf paper to achieve the best gate placement. The wire lengths and the number of violations are combined to form a score to be minimized. The actions that the PERSHING placer takes is based on a "temperature" T. At high temperatures, the placer will displace the cells further and possibly take suboptimal options to escape local minima. As T decreases, the algorithm is less likely to make drastic changes to the placement, but it will always accept a lower-cost placement.

[2] The default logic library contains specifications for a NOT gate, 2-input AND, NAND, and XOR gates, and a D flip-flop.

```
AND:
  pins:
    A:
      direction: input
      facing: west
      coordinates: [0, 0, 0]
    B:
      direction: input
      facing: west
      coordinates: [0, 2, 0]
    Y:
      direction: output
      facing: east
      coordinates: [0, 1, 3]
      function: "(A*B)"

  blocks: [[[55, 1,  0,  0],
            [ 0, 1, 76, 55],
            [55, 1,  0,  0]],
           [[ 0, 76,  0, 0],
            [ 0, 55,  0, 0],
            [ 0, 76,  0, 0]]]
  data:    [[[0, 0, 0, 0],
             [0, 0, 1, 0],
             [0, 0, 0, 0]],
            [[0, 5, 0, 0],
             [0, 0, 0, 0],
             [0, 5, 0, 0]]]
```

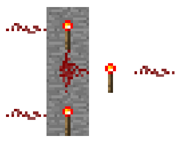

Figure 6. Specification for a 2-input AND gate in the cell library, written in YAML. The corresponding AND gate, in Minecraft blocks, is depicted to the right. The numbers in `blocks` represent Minecraft block IDs, which uniquely identify a block in Minecraft. The numbers in `data` specify additional information about the block, such as rotation for torches.

The placer places cells onto a grid larger than the individual block sizes to reduce the number of configurations, and consequently, running time. The grid is spaced by the largest dimension of any one cell, plus a margin set by the designer (at this time, the margin is 5). This allows ample room for redstone signals to travel between cells, especially in congested regions. (Unlike real-life physical design, logic gates and redstone signals can be placed anywhere.)

A completed placement of a 4-bit counter is shown in Figure 7.

4.4 Routing

PERSHING's routing algorithm is loosely adapted from the SILK router paper [13]. That algorithm scores nets by their lengths, the number of vias and pins, and the number of violations, with worse nets earning higher scores. A net has a violation if it intersects a logic gate or another net. After initially routing all of the nets, PERSHING scores these nets. With preference to nets with high scores, PERSHING randomly selects nets to rip-up and re-route with Lee's algorithm [14]. Each net has between 10 - 90 % of being re-routed, so PERSHING can even rip up seemingly good nets that happen to critically block the way of another net. After re-scoring all of the nets, PERSHING repeats rip-up and re-routing until there are no more violations.

In Lee's algorithm, a breadth-first search determines the lowest-cost feasible routing for the selected pins. This algorithm may introduce vias to cross over other nets. Occasionally, Lee's algorithm may not find a violation-free routing for the net. In this case, the algorithm, at high cost, can introduce a violation to complete routing. Such violating nets will be later ripped up for re-routing.

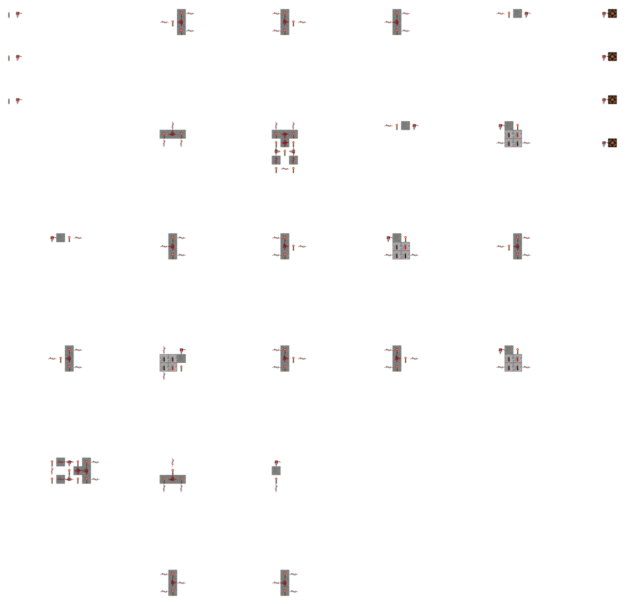

Figure 7. Top-down view of the completed PERSHING placement of a 4-bit counter on a grid, with input and output pins added to the left and right, respectively.

A completed routing of the 4-bit counter is shown in Figure 8.

4.5 Extraction

Extracting the blocks in the circuit involves processing the routing produced by the router (a list of coordinates) to produce the actual blocks present in the final Minecraft design. In most cases, this will be a redstone wire traveling on some surface, but they can also be redstone repeaters or vias.

4.5.1 Signal Repeating

Redstone repeaters are required to boost a signal from any non-zero strength back to full strength. However, they may only accept a signal from directly behind the repeater to directly in front of the repeater. This limits redstone repeater placement, and, in pathological cases, may cause signals to be unbufferable. This would require the wire to be re-routed.

4.6 Timing

While redstone wire transmits with no delay, it has a finite transmission distance. Redstone repeaters, piston circuits, and redstone torches introduce circuit delays. In sequential circuits, the clock period is limited by the longest combinational delay between any input or sequential circuit and any output or any other sequential circuit. As a result, it is cru-

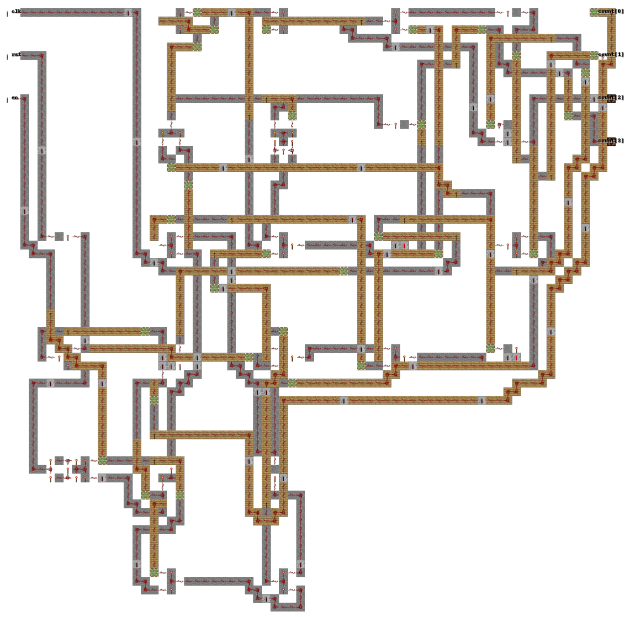

Figure 8. Completed PERSHING routing of a 4-bit counter based on the placement in Figure 7. Wires traveling on the stone layer (gray) are separate and do not intersect wires traveling on the wood layer (light brown). The two layers are connected by vias (not depicted).

Component	Ticks	Redstone Ticks
Redstone Wire	0	0
Redstone Repeater	2 – 8	1 – 4
Up Via	4	2
Down Via	4	2
AND Gate	4	2
XOR Gate	4	2
NAND Gate	6	3
NOT Gate	6	3

Figure 9. Delays for some redstone components.

cial to minimize any delays, especially those introduced by repeaters.

The smallest unit of time is the "tick," which represents 0.05 seconds of real-life time. However, redstone ticks, the minimum delay required to activate a redstone component, are two such ticks. The delay for some components are listed in Figure 9.

PERSHING measures the critical path in the circuit by measuring the longest path delay from any input or sequential element to any other sequential element or output, t_{crit}. The program also reports the clock frequency of the complete circuit, or $1/t_{\text{crit}} = f_{\max}$. Typical maximum delays for these circuits run between 20 to 40 redstone ticks, which means a maximum frequency no higher than 0.5 to 1.0 Hz.

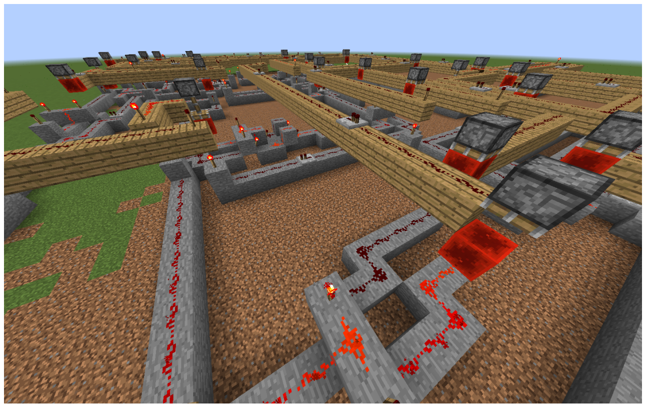

Figure 10. Fully placed-and-routed 4-bit adder in the Minecraft world.

4.7 Insertion

Finally, the extracted layout must be placed into the Minecraft world. Figure 10 shows a placed and routed 4-bit adder in a flat Minecraft world.

5. Evaluation

5.1 Metrics

There are several measures to measure the ability of an algorithm to place and route a circuit, and they have analogues in Minecraft.

- *Area or Volume*. A good design minimizes the volume required to accommodate the logic gates and necessary routing. Because Minecraft circuits are not limited to a single layer, three-dimensional circuits are possible, and may indeed be necessary to achieve the most compact design. This can be measured by the volume occupied by the smallest three-dimensional box or two-dimensional rectangle that can enclose the entire design. (In Minecraft, this measure can also be the number of blocks used in the design.)
- *Delay*. An optimal routing reduces the delays between circuits. Reducing the wire lengths also reduces the number of redstone repeaters required by the circuit; it also increases the clock speed for sequential circuits.
- *Feasibility*. The tools, especially the router, must complete; otherwise, circuit designers must repair violations by hand, which can be intractable.
- *Completion Time*. For them to be useful, the place and route algorithms must be fast.

5.2 PERSHING Performance

Figure 11 shows some statistics regarding PERSHING's performance. This paper evaluates four designs: a simple RS flip-flop, a 2-bit adder with carry, a 4-bit counter, and a 7-segment display decoder (roughly the equivalent of a 7447 integrated circuit). The "Logic Cells" column corresponds

Design	Logic Cells	Volume ($h \times w \times l$)	Nets	Critical Delay (ticks)	Max Frequency (Hz)	Completion Time (s)
2-input RS flip-flop	2	$5 \times 9 \times 34$	4	20	1.00	15
2-bit adder with carry	7	$5 \times 71 \times 60$	7	32	0.63	117
4-bit counter	23	$5 \times 58 \times 86$	26	41	0.49	1084
7-segment display decoder	63	$5 \times 97 \times 123$	67	*	*	(still running)

Figure 11. PERSHING performance statistics for various small circuits. "*" entries indicate placeholders for a routing that did not complete.

with the number of gates and flip-flops used in the circuit. The "Volume" column measures the three-dimensional space needed to fully capture the extracted layout. The "Nets" column indicates the number of connections required by the circuit and provides a relative estimate of difficulty for the routing algorithm. The "Critical Delay" and "Max Frequency" measure the slowest combinational propagation delay and the fastest clock frequency for the resultant circuit, respectively. Finally, the "Completion Time" column shows the time required by the computer to complete the placement and routing. All measurements were performed on a MacBook Pro with a 2.5 GHz quad-core processor and 16 GB RAM.

5.3 Analysis

The placement completes relatively quickly for small designs, although larger designs can take a long time, due to the fixed number of iterations, currently 2,000. The routing algorithm can be particularly sluggish; the incomplete routing for the 7-segment display decoder illustrates the shortcomings of Lee's maze-routing algorithm. When the number of logic cells and nets triples from one design to the next, the completion time increases by nearly an order of magnitude.

6. Conclusions and Future Work

The placer could be optimized by abandoning the grid arrangement, but more effort will be needed to route the wires without violations. Knowing when to stop the placement algorithm, based on finding a locally optimal solution, could help reduce the running time. The router could also be improved by implementing an algorithm to find the minimum rectilinear Steiner tree (RST) to span all of the points in a net. The current router only routes segments between pins and fails to consider points within existing segments. While Lee's maze-routing algorithm guarantees the shortest path possible, if it exists, the algorithm incurs an $O(mn)$ computational cost for routing a single net on an $m \times n$ grid because it must exhaustively check all grid locations in its breadth-first search. Routing Minecraft redstone circuits introduces a third dimension, which exacerbates the running time problem to $O(lmn)$ for an $l \times m \times n$ three-dimensional grid. Future work suggests the implementation of the Mikami-Tabuchi line-routing algorithm [15], which only has a running time in $O(L)$ for searching L lines.

With adequate sophistication and enough computational time, it will be possible to synthesize arbitrarily complex circuits, including complete computer processors. Adding hardware macros, which represent compact black-box hardware functionality, can accelerate the design process, and are theoretically already supported by PERSHING. Supporting "schematic capture," to obtain the configurations and pins of blocks, instead of manually entering in Minecraft block IDs, would greatly improve the library available to circuit designers. PERSHING only supports one module, but allowing encapsulation of pre-built modules would reduce the design time and facilitate sharing of good designs.

Being able to edit existing placements or existing routings before incorporating it into the Minecraft world would greatly assist designers. Although challenging, algorithms could be developed to factor into any existing Minecraft terrain, altering it as little as possible (as the world is typically not flat).

Finally, PERSHING has incredible educational potential; the appeal of Minecraft to younger children means that many kids could be introduced to digital circuit design and logic through redstone circuits.

In this paper, we introduced PERSHING, a tool that places redstone circuits in Minecraft and routes wires between the gates. There are decades' worth of research in placing and routing algorithms to incorporate into Minecraft redstone place and route tools, but PERSHING represents the first comprehensive step in this direction.

7. Acknowledgements

The authors would like to thank the efforts of the reviewers and the program committee. The authors would also like to thank their advisors, who are not aware 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 otherwised approved by Mojang.

References

[1] Redstone Update. [Online]. Available: http://minecraft.gamepedia.com/Redstone_update

[2] [Online]. Available: https://minecraft.net/

[3] Nine Digit Comination Lock [Minecraft Redstone]. [Online]. Available: https://www.youtube.com/watch?v=MmQ4f6ZlUh0

[4] theinternetftw. (2010, Sep) 16-bit ALU in minecraft. [Online]. Available: https://www.youtube.com/watch?v=LGkkyKZVzug

[5] skupitup. (2012) DEMO program of my redstone computer in Minecraft "BlueStone".

[6] Wikipedia, "MGM-31 Pershing — Wikipedia, The Free Encyclopedia," 2016, [Online; accessed 28-February-2016]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MGM-31_Pershing&oldid=703770275

[7] (2015, April) VLSI Craft. [Online]. Available: https://github.com/Disasm/vlsi-craft

[8] (2014) RedGen. [Online]. Available: https://github.com/kkroening/RedGen

[9] Gamepedia, "Chunk format." [Online]. Available: http://minecraft.gamepedia.com/Chunk_format

[10] ——, "Tick." [Online]. Available: http://minecraft.gamepedia.com/Tick

[11] C. Wolf, "Yosys Open SYnthesis Suite," http://www.clifford.at/yosys/.

[12] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *Solid-State Circuits, IEEE Journal of*, vol. 20, no. 2, pp. 510–522, April 1985.

[13] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai, "SILK: a Simulated Evolution Router," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 8, no. 10, pp. 1108–1114, Oct 1989.

[14] C. Lee, "An Algorithm for Path Connections and Its Applications," *Electronic Computers, IRE Transactions on*, vol. EC-10, no. 3, pp. 346–365, Sept 1961.

[15] K. Mikami and K. Tabuchi, "A Computer Program for Optimal Routing of Printed Circuit Conductors," in *IFIP Congress (2)*, 1968, pp. 1475–1478.