A High-Speed & High-Capacity Single-Chip Copper Crossbar

-A submission to the SRC Copper Design Challenge-

Design Team members:	**John Damiano** **Bruce Duewer** **Alan Glaser** **Toby Schaffer** **John Wilson**
Lead Faculty:	**Dr. Paul Franzon**
Affiliation:	**North Carolina State University**
Contact:	**John Damiano** **Rm. 419, EGRC, 1010 Main Campus Dr.** **North Carolina State University** **Raleigh, NC 27695-7914** **(919) 513-2016**

Abstract

In this report we discuss how the use of copper interconnects as part of a state-of-the-art IC fabrication technology substantially enhances the performance of a crossbar circuit compared to conventional aluminum interconnect technology. The crossbar offers 30% greater throughput and 15% lower latency with copper interconnect. Moreover, using our crossbar design as an example, we show that the use of copper interconnect provides additional design advantages. Copper technology enables use of a smaller crossbar cell, offering higher performance with a substantially smaller die size (more than 50% smaller compared to cell with aluminum interconnect). Reducing die size can improve process yield and perhaps improve integration of other “system-on a-chip” (SOC) components - these features make copper technology particularly attractive for future SOC solutions.

Note: This report is available on the Web at http://www4.ncsu.edu/~jdamian/copper.html

I. Mission

The objective of this project is to demonstrate the advantages of copper interconnect technology for state-of-the art IC design. To accomplish this goal, we have developed a circuit where performance is chiefly constrained by the interconnect; that is, a circuit where improvements in interconnect performance directly and significantly impact the overall performance. Such an application will most clearly demonstrate the advantages of copper technology for IC design compared to the traditional aluminum interconnect process.

II. The Copper Crossbar - an introduction

Our entry in the Copper Challenge is a crossbar (or crosspoint) switch. This type of circuit consists of numerous input and output lines and, upon programming, can provide for the arbitrary and simultaneous connection of any input to any output, as shown in Figure 1. The crossbar is an essential part of many circuits requiring multi-channel signal switching, such as ATM (Asynchronous-transfer-mode) switches[1], specialized VLIW (Very Long Instruction Word) video signal processors[2], and many DSPs (digital signal processors)[3].

Tomorrow's designs will also include crossbar switches as key components. "System-on-a-chip" [4] (SOC) solutions involve integrating numerous IC components onto one die, reducing the number of chips required to implement a given functionality. For example, microprocessor, DRAM, and DSP functions can be integrated onto a single chip, improving circuit performance and enhancing utility to the consumer. This integration increases the complexity of the interconnect and complicates the switching between components. Since signals between functional units are no longer routed off chip, a high-performance interconnect scheme is required to handle the extensive routing and added design complexity. High-density crossbars will be a key element in these embedded applications where they will be used to efficiently route signals between processing units within a chip. We anticipate that a demand for high-density, high performance crossbars will be driven by the industry's need for

advanced design solutions, and new materials such as copper will enable process technology to keep pace with tomorrow's designs.

We chose a crossbar design for this project for several reasons. First, as mentioned above, *the need for high-speed switching technology is growing as designs grow faster*, especially with the advent of SOC technology. Second, *crossbar circuits inherently contain long, heavily-loaded interconnect lines* - a characteristic they share with other common designs such as SRAM, DRAM, and logic cache memory. A crossbar is therefore representative of a family of designs which may benefit from advances in interconnect technology. Finally, *the crossbar is simple and efficient enough such that it can directly demonstrate the advantages offered by advanced interconnect*, including lower resistivity and reduced capacitive load.

In this report we discuss how the use of copper interconnects substantially enhances the performance of a crossbar circuit compared to conventional aluminum interconnect technology. The crossbar offers 30% greater throughput and 15% lower latency with copper interconnect. Moreover, using our crossbar design as an example, we show that the use of copper interconnect provides additional design advantages. Copper technology enables use of a smaller crossbar cell, offering higher performance with a substantially smaller die size (more than 50% smaller compared to cell with aluminum interconnect). Reducing die size can improve process yield and perhaps improve integration of other "system-on a-chip" components - these features make copper technology particularly attractive for future SOC solutions.

III. Features of the copper crossbar circuit

Below, the most significant features of the Copper Crossbar design are described. Details regarding how these features are implemented are discussed in the next section, "Circuit Design and Layout Features".

Efficient programming

This circuit is fully programmable using the input lines and write enable lines to

address each memory bit of a given output column. Additional decoder lines for each column are not required since programming is done through input lines. Programming is performed column-by-column as the chosen cell from each column is selected through the input. It should be noted that this crossbar design is non-blocking, i.e. any input can be sent to any output, and the crossbar can operate in broadcast mode.

Reset feature

The reset feature writes a “0” to all cells, effectively disconnecting all switches and clearing all outputs. A reset is always performed prior to programming. After reset, the output lines remain low until each input is programmed to a chosen output using the write-enable lines. The reset feature also works with the pre-configure feature, allowing for a fast write or re-write.

Pre-configure feature

The pre-configure feature allows the user to instantly program any of several common input/output configurations to the crossbar array within a single write cycle (<3ns). The Copper Crossbar circuit features two built-in configurations: corner turn[5] (used to perform a bit-reverse for DSP butterfly algorithms) and broadcast mode (required for ATM switches), which are illustrated in Figure 2. Pre-configured input/output mappings improve testability and can be modified to fit the needs of a specific application.

High density

Our design fits a fully functional crossbar circuit with 128 input lines and 128 output lines (128x128) into an area of approximately 0.72mm x 2.43mm based on our preliminary layout. The use of UMC’s 0.18μm 6LM copper process allows for a very compact crossbar cell. Our 128 x 128 crossbar network is of higher density than any fabricated single-chip crossbar switch reported in literature to our knowledge[1,3,6,7]. The package requirements of the Copper Challenge limit the number of available

pads to 28. Given the limited number of pads available due to the die size constraint, we have chosen to route only the 16 cells (4x4) on each corner of the array to the input/output pads, as illustrated in Figure 3. This creates a fully testable 8x8 crossbar. Although the testable portion is a subset of the entire design, selecting the corner cells allows us to effectively characterize the 128x128 crossbar and measure parametrics such as delay and top speed since all input and output lines are fully loaded. We believe that this strategy is an acceptable compromise given the nature of our circuit and the contest's packaging constraints.

IV. Circuit Design

Details regarding the implementation of the features listed above are discussed in this and the next section. Specific simulation results are presented in Sections VI and VII.

Crossbar cell

The crossbar cell is designed around a latch used to store a memory bit, as shown in the schematic diagram in Figure 4. An input-output connection is created by writing a '1' to a single memory bit within each output column. Each memory bit is written by holding the selected input 'high' while strobing the chosen output line's write_enable line, as shown in Figure 5. The stored memory bit is used as one input to a 2-input AND gate and therefore determines which input is passed to the output line. A stored value of '0' holds the cell output 'low', while a stored value of '1' passes the input to the cell output. All crossbar cell AND outputs are combined for a given output column through an OR tree, and the output of each tree constitutes a single output line as illustrated in Figure 6. For a crossbar with 128 (2^7) input lines, 7 OR gate levels are required in the tree. The total delay through the OR gates is roughly 7 times a single gate delay, plus any RC delay resulting from long interconnects between the final OR stages. Delay results are discussed in Sections VI and VII.

Several alternatives to the OR tree output stage were considered, such as a transmission gate-based cell with a common output line, and a tree of 2:1 multiplexors.

Each of these designs display similar results for small crossbars (in fact, for small crossbars, the transmission gate cell is faster). However, the alternate designs considered did not scale as well as the OR-tree based design for very large crossbars. The common output line for transmission gate-based cell design would be very heavily loaded. Programming such an array would require one select line per input, and the number of select lines grows prohibitively large for large crossbars. Even a mux-based design requires numerous select lines for each output column. The chosen design uses interconnect more efficiently.

Within the crossbar cell, all transistors are minimum length. Transistor width is slightly larger than minimum for most devices at 0.44μm / 1.04μm (NMOS / PMOS) to maintain beta=1. Use of minimum-size or near-minimum-size transistors serves to keep cell size small, which is critical for a large array. Moreover, the selective use of minimum-width devices reduces the capacitive load on the long input and output lines. For example, the NAND gate in the crossbar cell driven by the 2.43mm-long input line uses minimum-size 0.24μm/0.18μm NMOS and 0.36μm/0.18μm PMOS transistors to reduce the capacitive load on this line. Alternatively, the transistors within the final inverter stage of each OR gate are wider to facilitate driving the long interconnects between the final OR stages. Thus, transistor sizes were determined based on the tradeoff between increased drive strength and increased capacitive load.

The preliminary layout of the crossbar cell is shown in Figure 7. Figure 8 is the same cell shown without several layers for the sake of clarity. The figures actually show two crossbar cells, one flipped above the other to facilitate arraying of the cells. The output of the two cells is OR'd by the top OR gate, while the bottom OR gate is made available for the OR tree configuration. The individual components of the cell are noted in Figure 8 and can be compared to the schematic in Figure 4.

Reset function

The reset feature used to instantly set all memory bits to 'low' and thus clear all out-

puts is implemented using a pull-down NMOS transistor within the crossbar cell (Figure 4). This global reset is asserted prior to any programming / re-programming or preset function to clear any former programmed configurations.

Pre-configure function

The pre-configure feature enables a fast global write for a number of pre-selected input-output configurations. A 'high' preset input to the cell is inverted and turns on a pull-up PMOS transistor to write a '1' to the designated memory bits, as shown in Figure 4. Different bits are chosen for each output, and arbitrary input-output mappings can be obtained, as shown in Figure 1. The reset function should be performed to clear the memory before pre-configuring.

Write-enable (WE) circuit

These subcells consist of decoder input from 7 select bits and a write enable bit, as shown in Figure 9. The decoder inputs and the WE bit are AND'ed (using a NAND/NOR structure) and the output is sent to the write_enable line of all cells in the selected output column. Within the crossbar cell, the write_enable signal writes a value to a cell (WE high) or stores the memory bit (WE low). The layout of the write-enable circuit is shown in Figure 10.

Output line drivers

Each output line comes from a single OR gate within the crossbar cell (as shown in Figure 6) and therefore need to be buffered up to drive the output pads. This is accomplished simply by using an inverter chain with progressively larger transistors. The inverters also serve to restore the input signal.

Connecting only edge cells to form an 8x8 crossbar

We chose a crossbar circuit, in part, for its inherently long interconnects to clearly demonstrate the advantages of low-resistance copper interconnect. While a high-density circuit with numerous I/O's is desirable in an embedded system, a stand-alone crossbar of this size is dominated by input / output lines. The packaging

requirements for this contest allow only a 28-pin DIP package for high-frequency testing. Given this constraint, we elected to route only the corner cells to pads, providing an 8x8 crossbar for testing. Reducing the number of I/O's routed to pads will improve testability. The corner cells were chosen to best demonstrate minimum and maximum delay through the circuit and thereby determine the precise impact of copper technology on a circuit of this nature.

V. Comments on Preliminary Layout - Interconnect Strategy

Although the full layout is not finished at this point, the layout of many subcells is complete. Our layout strategy is based on enabling the efficient design and testing of the circuit as described below:

Interconnect layout

The interconnect strategy for the crossbar is illustrated in Figures 11 and 12. All metals layers are used to optimize circuit performance. Metal1 is used for local routing and local V_{DD}/GND distribution, and metal2 is used for local routing. Metal3 is used for the pre-configure and reset lines as well as output signal routing in the OR tree, while metal4 is used as a GND plane to minimize signal line self-inductance and reduce crosstalk between the metal3 outputs and the inputs, which run on metal5. Metal6 is used for global V_{DD}/GND routing as well as providing an additional GND plane over the crossbar cell array.

The metal3 and metal5 layers are of particular interest. On metal3, the metal pitch is as large as possible given the number of output interconnects required vs. the cell size to allow for maximum space between the metal3 lines, reducing coupling capacitance. Interconnects for the final OR gates before the output buffer are the longest, spanning numerous cells, and therefore present heavy loads. Metal3 was chosen for output lines because it allows for more wires with narrower pitch compared to higher-level metal layers, which is necessary to minimize cell size. In contrast, the single input line per cell was placed on metal5. Using metal5 resulted in low resistance

input lines in spite of their long length. Here, interdigitated ground lines[8] are used to shield the signal lines, reducing the likelihood of crosstalk and delay problems introduced by self-inductance.

The capacitive load on metal3 and metal5 interconnect, namely the input and output stages respectively, limits the maximum input signal frequency and has the most impact on its performance. This is especially true for the metal3 lines where line capacitance dominates load capacitance. This is discussed in more detail below and also in the simulations in Section VII.

Interconnect trade-offs

The use of copper facilitated our strategy of obtaining very small cell size while maintaining high performance in an interconnect-intensive circuit. First, given a cell area and a fixed number of interconnects, an interconnect pitch can be calculated. Using this pitch, a linewidth/space combination must be chosen. Increasing the width of the interconnect decreases line resistance but increases capacitive coupling. Narrower lines reduce capacitance but increase resistance and could potentially impact reliability. The expected outcome is improved performance relative to aluminum. The *unexpected* outcome is particularly relevant in a circuit such as ours - reduced cell size to achieve the required performance. In arrayed applications such as memories, crossbars, gate arrays, etc., cell size is the key figure of merit. Cell size reductions reduce die area, enhance performance, and, perhaps more importantly, increase the device's suitability for embedded applications. Since our industry is moving toward SOC circuits, copper's advantage in this area is clear, as shown by our crossbar circuit.

VI. Simulation Results - Function and Features

The HSPICE netlist used to demonstrate the functionality of the crossbar circuit and illustrate the features of the design is given in Appendix A. The circuit used in the simulations in this section is an 8x8 crossbar with passive loads inserted to simulate

the full 128x128 design. Input and output lines are loaded based on the schematic design and preliminary layout, which we describe in more detail in Section VII.

In the netlist shown, 8 different input signals with varying frequency are applied to the inputs so that each input can be traced to the outputs. First, the configuration shown in Figure 13 is written to the crossbar and verified. Then, the corner turn pre-configure is tested after a reset is performed. After this is verified, the broadcast pre-configuration is applied following another reset. Pre-configure patterns are shown in Figure 2. Results of this basic simulation are described below. Simulation results describing a more extensive analysis of the crossbar circuit, as well as a comparison of the circuit performance for copper vs. aluminum interconnect, are discussed in Section VII of this report.

Simulation results - programming

The programming sequence for the crossbar is shown in Figures 5, 14, and 15. Figures 14a and 14b demonstrate the generation of the write_enable signal from select line inputs. The input signals are set high to overlap the designated output's write_enable signal as shown in Figure 15a, and Figure 15b shows in3, write_enable4 and out4 during programming. The input line value for each column is locked in at the falling edge of the write_enable signal. After programming, the input signals are applied. Figures 16-20 show that the proper functionality was achieved, i.e. we observe in0 -> out7, in1 -> out6, and so forth. An overview of crossbar programmed output signals can be seen in Figures 16 and 17, and both the input and output signals are shown for several cases in Figures 18-20. These results indicate that the crossbar circuit can be programmed accurately and efficiently. Based on these results, each output line can be programmed within 3ns.

Simulation results - reset and pre-configure

The 'reset' function clears (and effectively disables) all outputs. Figures 21-22 show in2 and out5 over the entire simulation as well as the reset/pre-configure sequences. Initially, this I/O combination was connected using the programming method

described above, and out5 passed a signal with f=1.67GHz (Figure 20)

It can be noted from the out5 plot that the reset function does indeed work. Figure 23 shows out5 from 55-95ns, and Figure 24 shows reset & pre-configure signals during the same period. The netlist calls for a reset at 65ns and 80ns. It can be seen in Figure 23 that the output clears at this time and passes no input signal.

The two pre-configure functions, corner turn and broadcast, call for out5 to be connected to in5 and in0, respectively. Pre-configure #1 is takes place at 70ns while pre-configure #2 occurs are 85ns. Examining the out5 signal between 70-80ns (pre-configure #1), a signal with f=2.0GHz is observed - this is in5, as shown in Figure 25. Examining out5 between 85-95ns (pre-configure #2), a signal with f=2.5GHz is observed - this is in0, as shown in Figure 26.

Since the programming, reset, and pre-configure all work properly, the functionality of the circuit and its features has been tested and verified.

VII. Copper vs. Aluminum Interconnect

Impact of interconnect on circuit performance

The resistance and capacitance of interconnect, especially long interconnect spanning the entire chip, can impact circuit performance. The goal of the Copper Challenge contest is to demonstrate the advantages offered by copper technology, and a performance advantage can be observed in our crossbar for copper interconnects vs. traditional aluminum technology.

The electrical properties of copper can be exploited in two ways. First, its reduced resistivity compared to aluminum means that, for a direct substitution of copper into a aluminum design, line resistance is lower by approximately 40%. Another advantage of copper is found when interconnect lines are scaled. Because the electromigration properties of copper are such that linewidths can be narrowed reliably, smaller copper

lines can be used in place of aluminum lines to reduce coupling capacitance while maintaining constant resistance values. Reducing interconnect capacitance improves speed in heavily loaded circuits, and the electromigration properties of copper interconnect prevent reliability failures.

To compare copper and aluminum interconnect characteristics, accurate resistance and capacitance values must be determined given the size and layout of our design. UMC has provided tables for metal resistance and capacitance, and all RC values in this report were calculated using this document [9]. Using the layout in Figure 7, resistance and capacitance values for long interconnects can be calculated and inserted into the crossbar netlist as passive components. In comparing the two materials, the design strategy used in most cases was to keep line resistance identical for the two technologies while shrinking the linewidth of copper, thus reducing capacitance. An example of the calculated interconnect RC values is shown in Table 1.

Table 1: RC values for Crossbar Interconnect

Interconnect	R, Al (Ω)	R, Cu (Ω)	C, Al (fF)	C, Cu (fF)	% Reduction in RC for Cu
input lines	38	38	1478	705	52%
reset lines	82	82	112	77	34%
preset lines	82	82	56	38	32%
write-enable lines	82	82	56	38	32%
OR tree, final stage	41	41	112	77	34%
OR tree, intermediate stage	20	20	56	38	32%

The most important values in Table I are those that lie in the signal path, i.e. the input lines and OR tree. Other interconnects are either non-critical (do not impact throughput) or are dominated by device capacitance and therefore are not impacted by the interconnect characteristics. The input and OR tree interconnects are global (span-

ning a large percentage of the chip length) and are chiefly loaded by the wires themselves. These are the interconnects that can benefit most from copper technology.

Comparing the two technologies using the methodology described above, we find that the use of copper interconnect results in a clear improvement in circuit performance, specifically in RC delay. The term “worst-case” is used below to indicate the longest path through the crossbar, which incurs the worst-case RC delays, and this is a fair metric for comparing the Cu and Al technology.

Figures 27a and 27b trace an input signal with T=0.5ns through the copper crossbar. It can be seen in Figure 27b that the signal degrades through the final stages of the OR-tree. Full rail-to-rail operation is maintained until the final OR stage, where the signal swing is 0.2 to 1.4V. The output drivers restore the full voltage swing. It is clear that the “weakest-link” in the signal path is the OR-tree. Figures 28a and 28b show input signals with varying frequency through worst-case Cu-model crossbar paths. The copper crossbar functions for a square-wave input signals with f=2.67GHz but fails at f=2.86GHz. The maximum input signal frequency for the copper crossbar is therefore 2.67GHz which corresponds to a data rate of 5.33Gb/s. As a point of reference, the “best-case” or shortest signal path is shown functioning at 2.86GHz in Figure 29. The delay (latency) through the copper crossbar is measured in Figure 30. The delay has two components: (1) delay from the input through the crossbar cell, and (2) delay through the OR gates. Delay through the OR tree increases in its final stages, as capacitance grows and the time to charge and discharge capacitance sets a limit on delay (and consequently top speed). The two delay components are measured as 80ps and 290ps, respectively, for the copper crossbar.

Modifying the netlist to use the parameters listed in Table I enables characterization of the crossbar using aluminum interconnect. Figures 31a and 31b trace an input signal with f=2.0GHz through the worst-case aluminum crossbar. Comparing Figure 31 to Figure 27, it can be seen that the more-heavily-loaded aluminum lines show infe-

rior performance on both the input lines and the OR-tree network. The aluminum interconnect prevents full rail-to-rail operation at a lower input frequency. It can be seen that the next to last stage runs from 0.1 to 1.4V, and the final OR stage operates from 0.5 to 1.3V. Figures 32a and 32b show input signals with varying frequency through worst-case aluminum-model crossbar paths. The aluminum crossbar functions for a square-wave input signals with f=2.0GHz but fails at f=2.2GHz. The maximum input signal frequency for the aluminum crossbar is therefore 2.0GHz with a data rate of 4.0Gb/s. The Copper crossbar is more than 30% faster. The delay (latency) through the aluminum crossbar is measured in Figure 33. The delay from the input through the crossbar cell is100ps, while the delay through the OR gates is 325ps - a 15% increase in latency for the aluminum crossbar. These results are summarized in Table 2.

Table 2: Data Rate and Latency vs. Interconnect Technology

Interconnect Technology	Max. Data Rate	Latency
Copper	5.3 GHz	370 ps
Aluminum	4.0 GHz	425 ps

Impact on cell size

Unlike many other process technology enhancements, the enhancements resulting from the use of copper interconnects in this circuit extends beyond a basic performance improvement. The electrical properties of copper allow for interconnect scaling (and improved performance) while maintaining a high current density in the narrow lines. This is possible due to copper's lower resistivity and enhanced electromigration properties. An added benefit exists for embedded applications. Achieving the performance benefits of copper using aluminum interconnect for this type of arrayed circuit would require some or all of the following: (1) use of larger drivers within the crossbar cell to compensate for the more heavily-loaded lines; (2) use of wider interconnect to guarantee that reliability specs are met; (3) increasing cell size to reduce coupling capacitance between input/output lines. The modifications

required to achieve equivalent performance, including changes to drivers as well as to interconnect linewidth and pitch, would increase cell size substantially. Modifying the aluminum crossbar cell to make RC equivalent to the copper cell (achieved by increasing spacing between lines) would require increasing cell width from19.5μm to 24.7μm and increasing cell height from 5.68μm to 7.34μm. The overall area penalty for these modifications is approximately 70μm^2 - the aluminum is 64% larger than the comparable copper cell. moreover, these figures do not include further changes to aluminum linewidth required to compensate for the larger cell size and subsequently longer interconnects. Although the total area increase for a single cell using aluminum interconnect may be small (70μm^2), the overall impact is huge because the crossbar is an arrayed structure, and incremental changes in cell size add up to a significant change in die area.

The impact of this advantage alone - achieving significant die size reduction while improving performance, as demonstrated by the copper crossbar - cannot be overestimated for SOC or embedded applications. This factor aligns copper process technology with the design technology of the future.

VIII. Summary

We have shown that the use of copper interconnect not only provides performance enhancements in a state-of-the-art circuit, but that design trade-offs make copper technology attractive for embedded applications. Compared to conventional aluminum interconnect technology, our crossbar with copper interconnect is 30% faster and displays 15% less latency. We have also demonstrated that the use of copper interconnect provides additional design advantages. Copper interconnect allows use of a smaller crossbar cell, offering higher performance with a substantially smaller die size. Features such as high performance and smaller die size make copper technology particularly attractive for future design solutions.

IX. References

[1] "250 Mb/s 32*32 CMOS Crosspoint LSI for ATM Switching Systems", M. Akata, et al., Digest of Technical Papers, 37th IEEE Int'l ISSCC, 1990, pp. 30 -31

[2] "High-performance Crossbar Interconnect for a VLIW Video Signal Processor", S. Dutta et al., Proceedings of Ninth Annual IEE Int'l ASIC Conference and Exhibit, 1996, pp. 45 -49.

[3] "A High Speed and High Precision 64x33 Crosspoint Switch IC", R. Savara, Gallium Arsenide Integrated Circuit Symposium Tech. Dig., 1997, pp. 105 -108.

[4] "Innovative Processes Bring Systems on Chips to Life", J. Lipman, EDN, September 1998, p. 46-54.

[5] "Crossbar Tree Networks for Embedded Signal Processing Applications", K. Teitelbaum, Proceedings of Fifth Int'l Conf. on Massively Parallel Computing, 1998, pp. 200-207.

[6] "A 2.5Gb/s 16x16bit Crosspoint Switch with Fast Programming", R. Savara and A. Turudic, Gallium Arsenide Integrated Circuit Symposium Tech. Dig., 1995, pp. 47-48.

[7] "A 250-Mbit/s CMOS Crosspoint Switch", H. J. Shin and D. A. Hodges, IEEE J. Solid-State Circuits, April 1989, pp. 478-486.

[8] "Layout Techniques for Minimizing On-Chip Interconnect Self-Inductance", Y. Massoud, et al, Proceedings of Design Automation Conference, 1998, p. 566-571.

[9] "SPEC No:G-04-LOGIC18-1P6M-COPPER/LK-INTERCAP", UMC Group, Rev. 0.1, 1999.

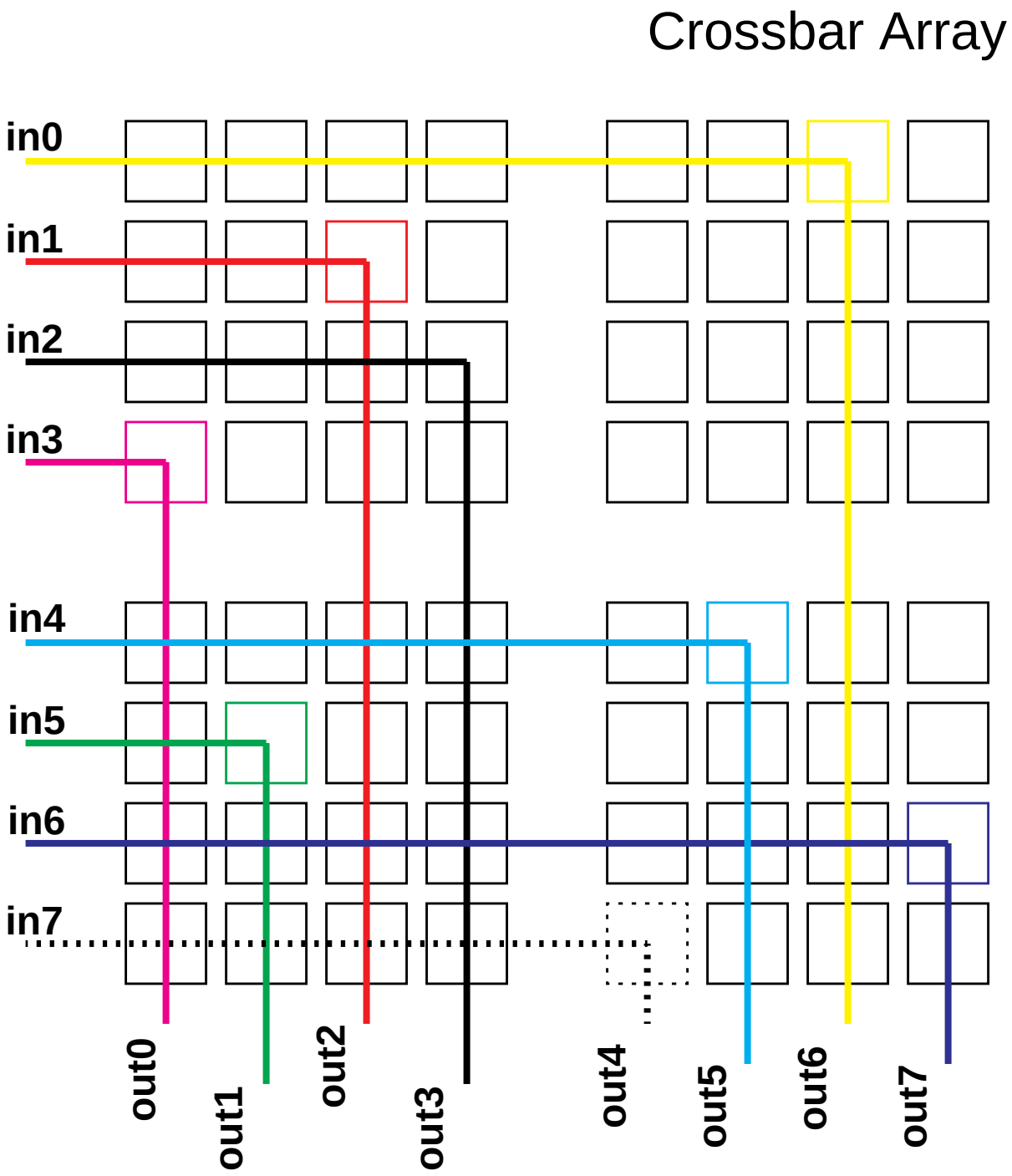

Figure 1. Schematic illustrating crossbar with arbitrary input/ output connections programmed.

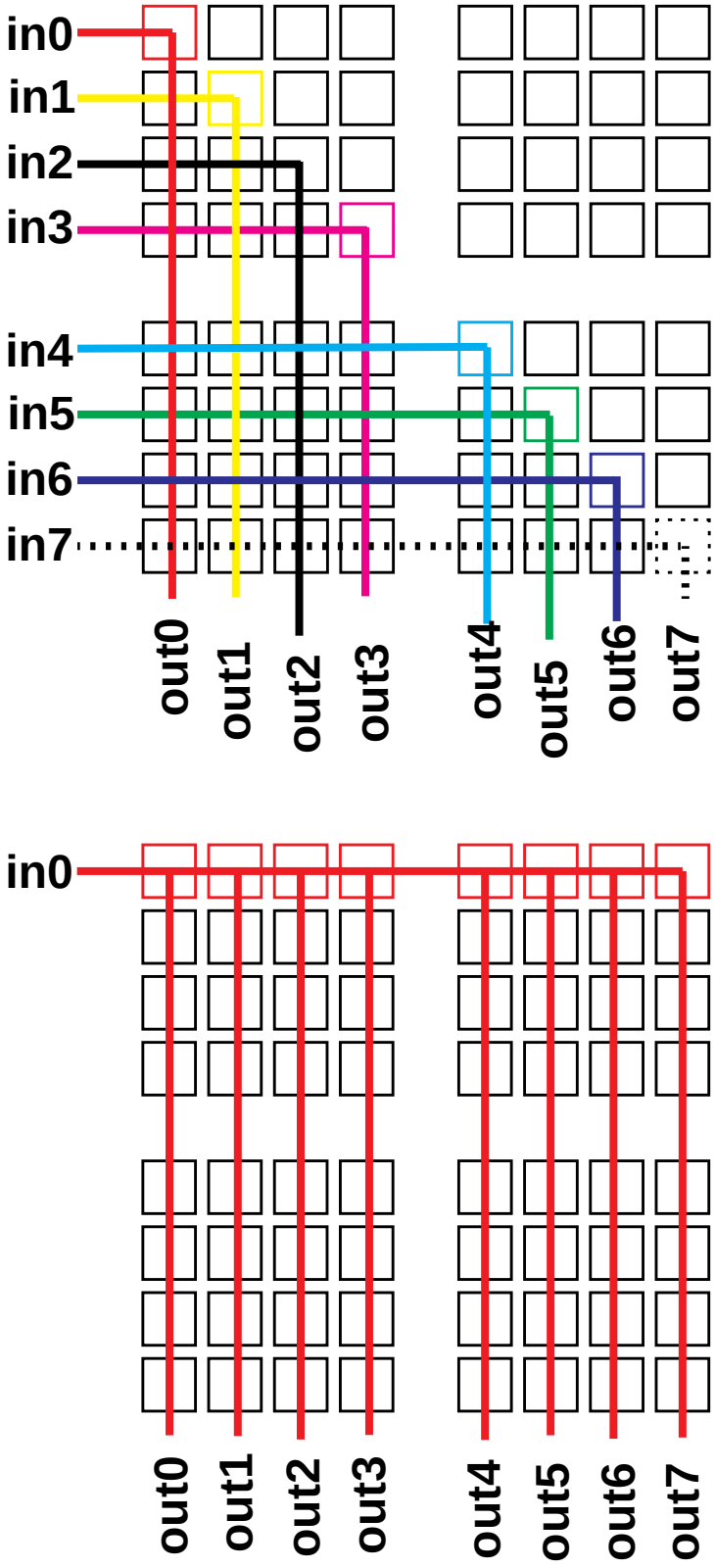

Figure 2. Schematic illustrating crossbar preset I/O configurations. Top: Preset #1, corner turn. Bottom: Preset #2, in0 broadcast mode

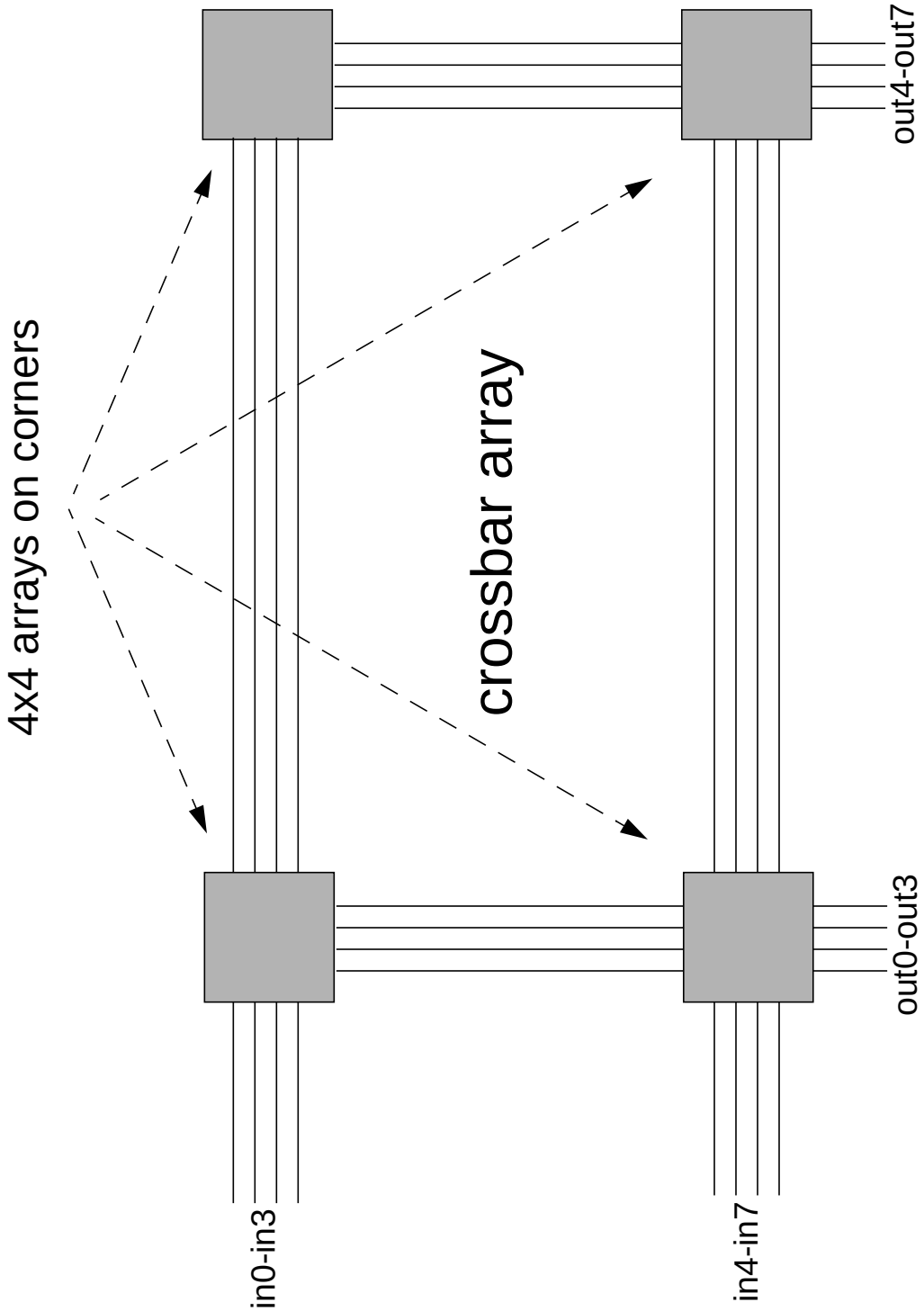

Figure 3. Routing corner cells of 128 x 128 crossbar to form 8x8 crossbar.

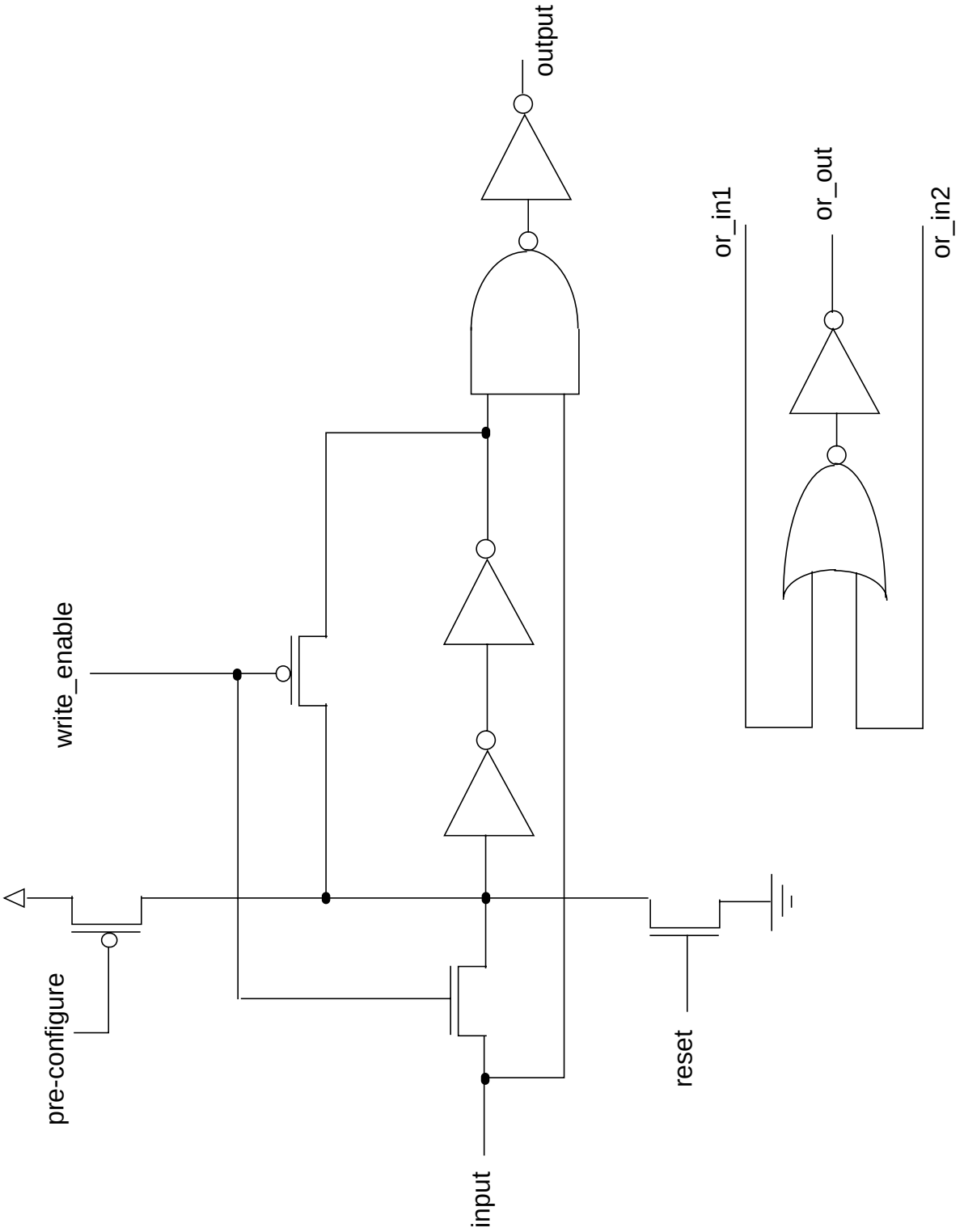

Figure 4. Schematic of crossbar cell.

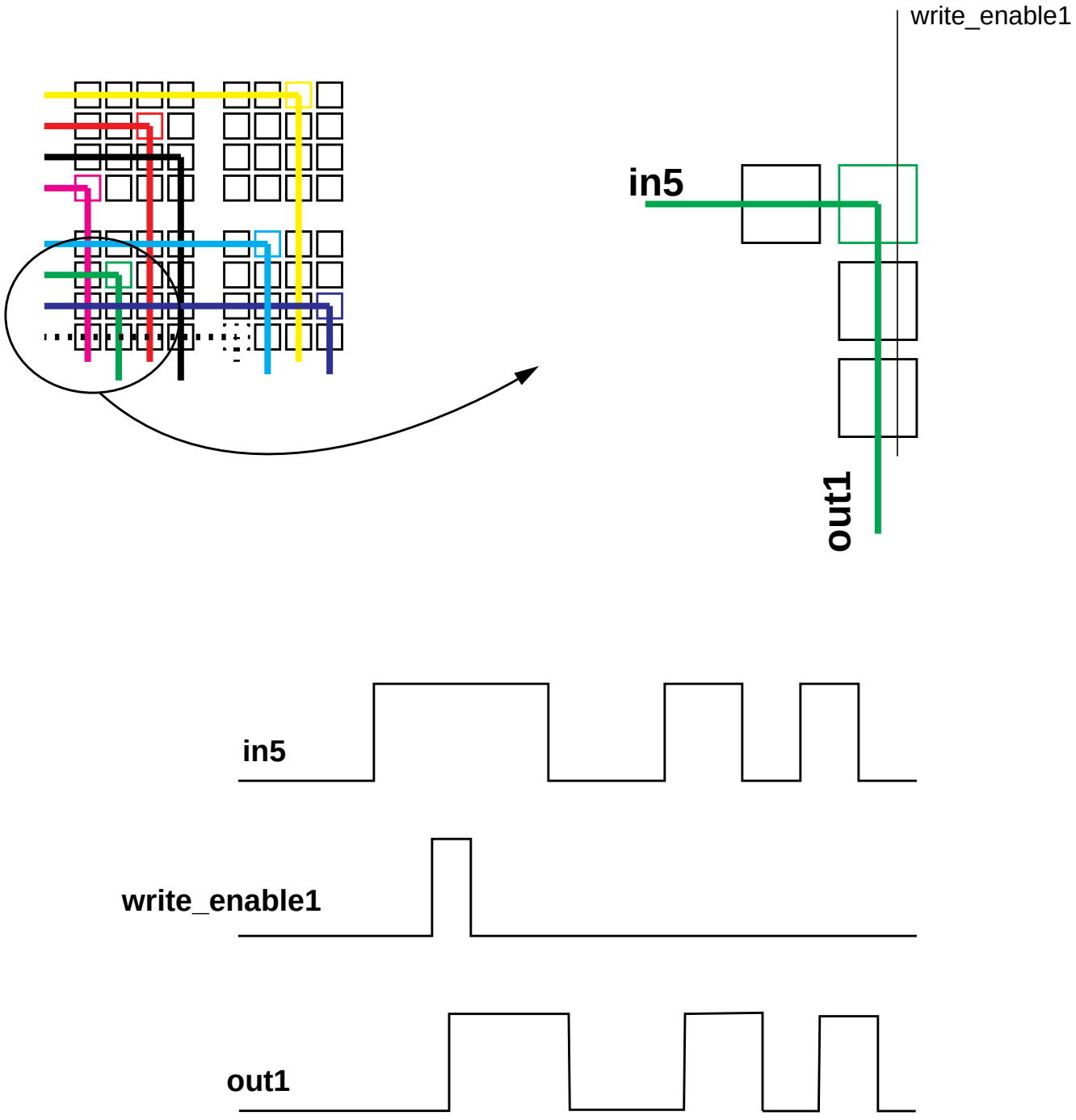

Figure 5. Schematic illustrating crossbar programming sequence Here, out1 is programmed to in5 by strobing write_enable1 while in5 is held high.

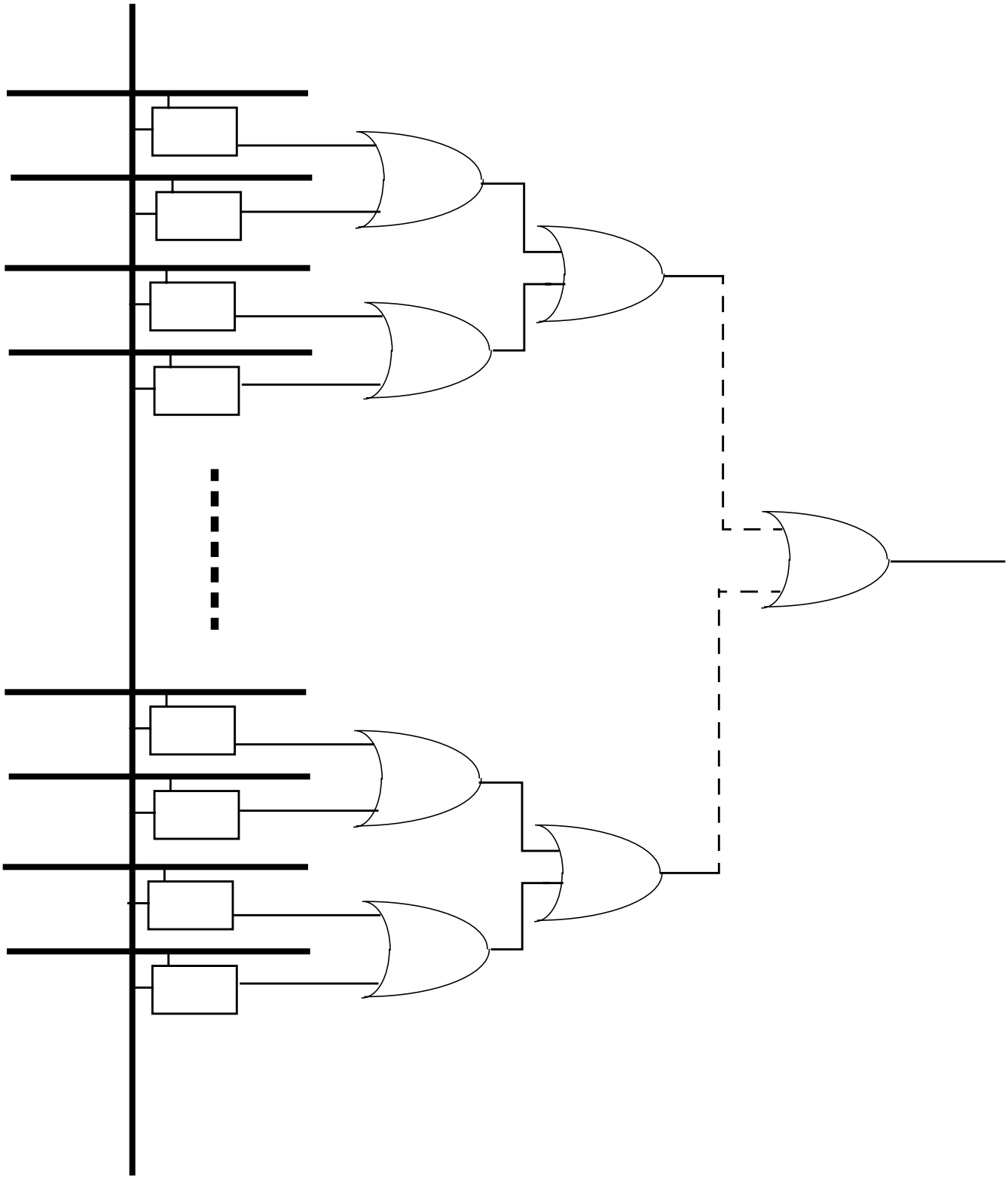

Figure 6. OR-tree network used to generate output signal for any output column.

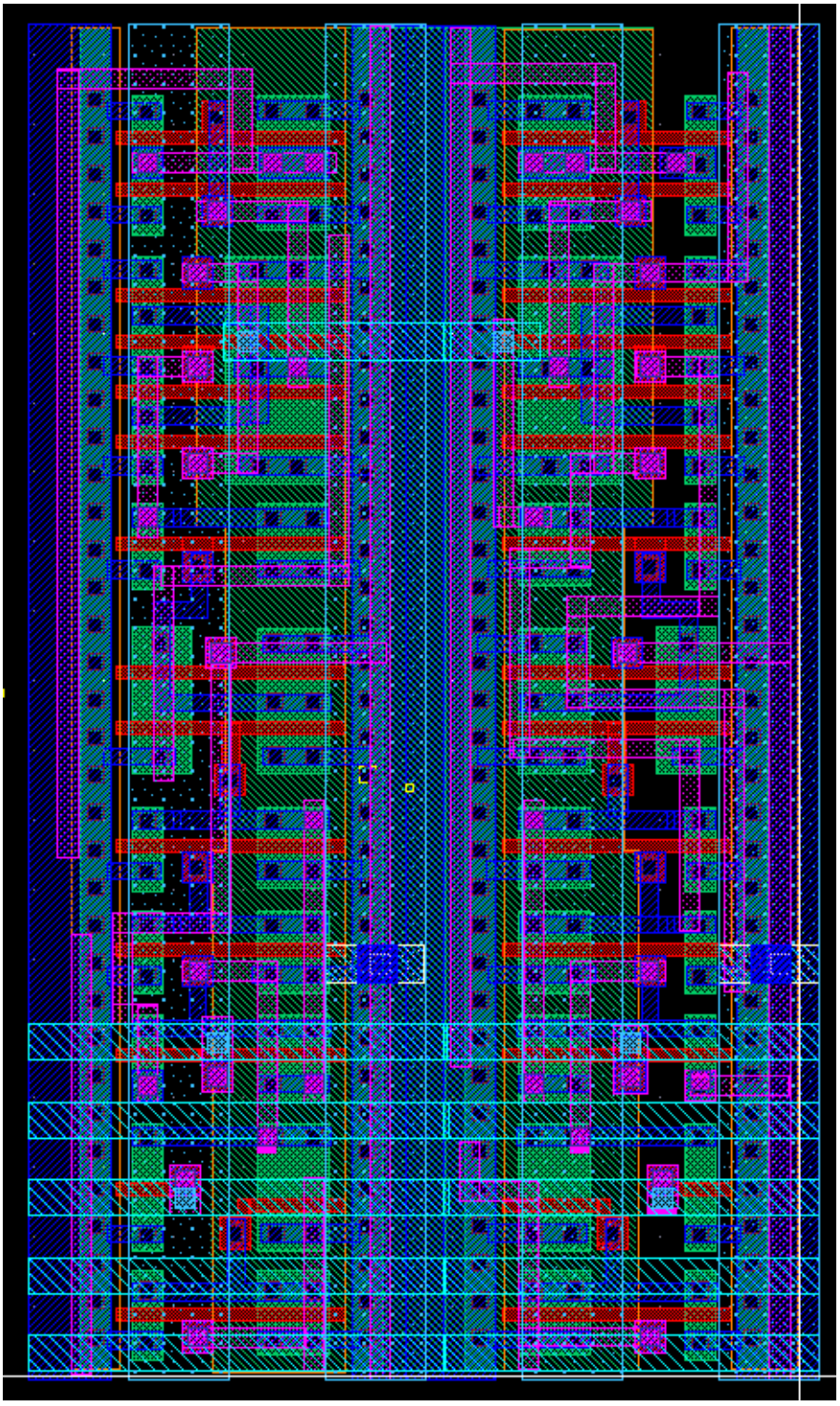

Figure 7. Crossbar Cell Layout

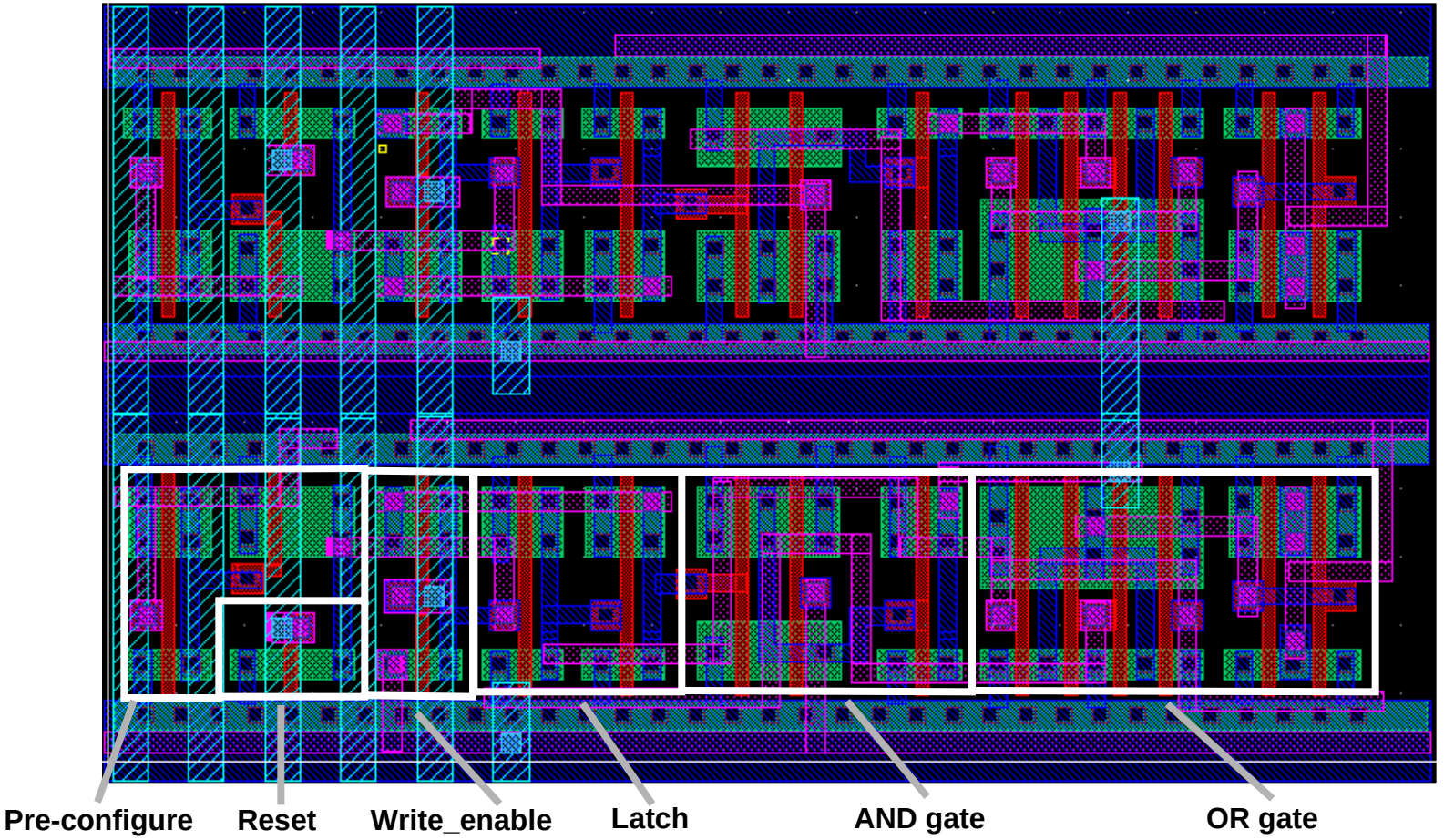

Figure 8. Crossbar Cell Layout. Various features are noted, see report for details.

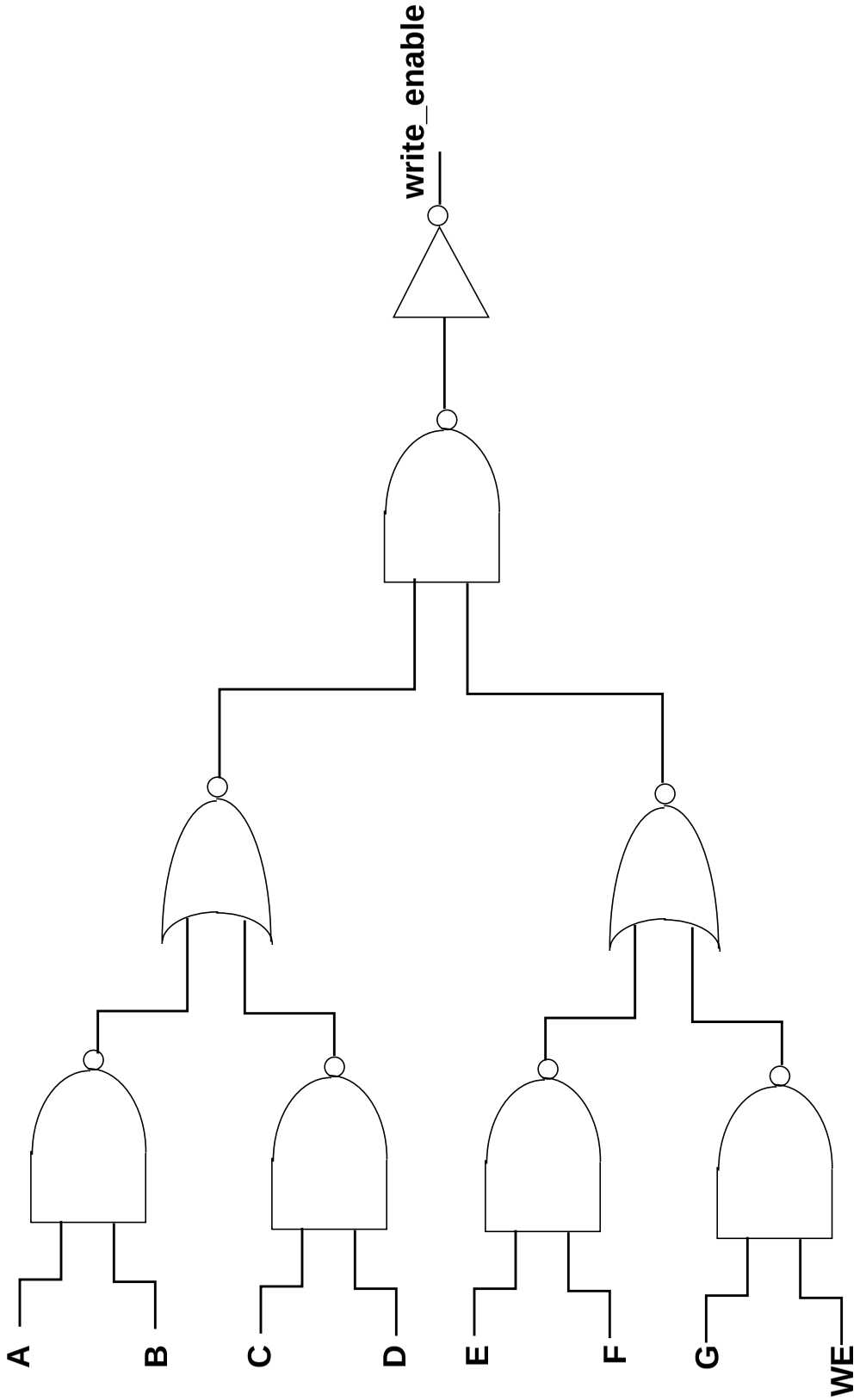

Figure 9. 8-input AND function to form write_enable signal. 2^7 combinations are possible using select lines A-G.

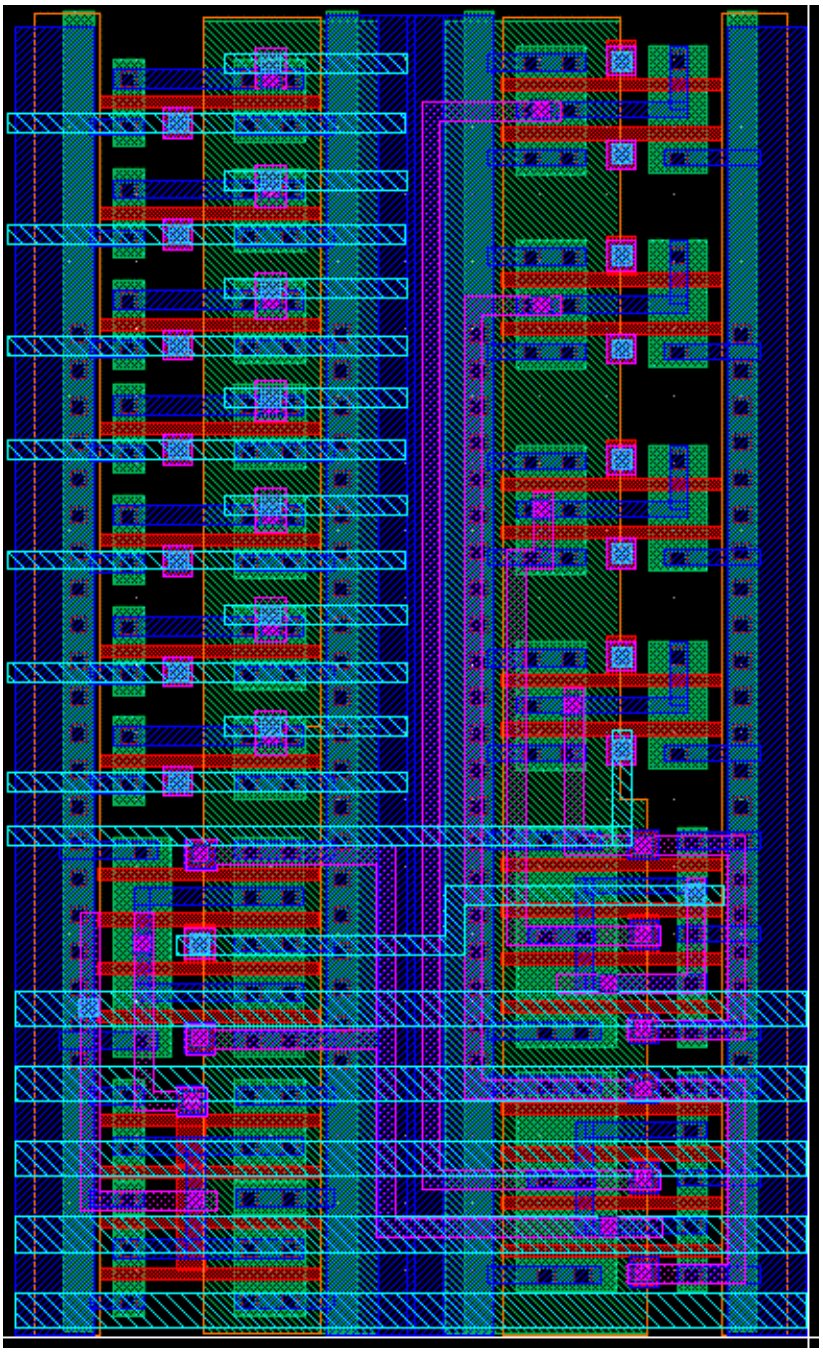

Figure 10. Write_enable Circuit Layout

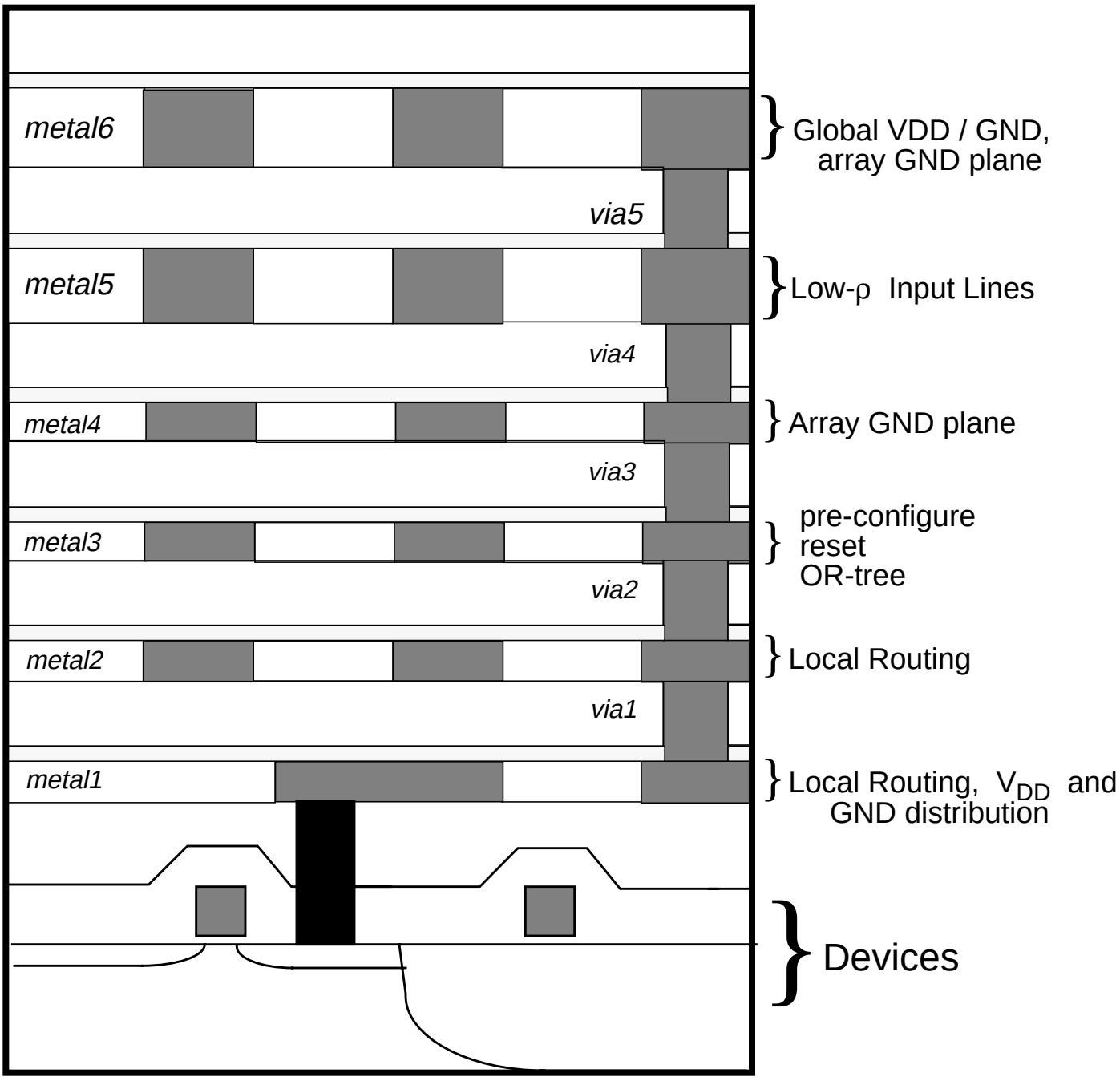

Figure 11. Crossbar circuit global interconnect strategy. See Figure 12 for details regarding how metals are used in the crossbar cell.

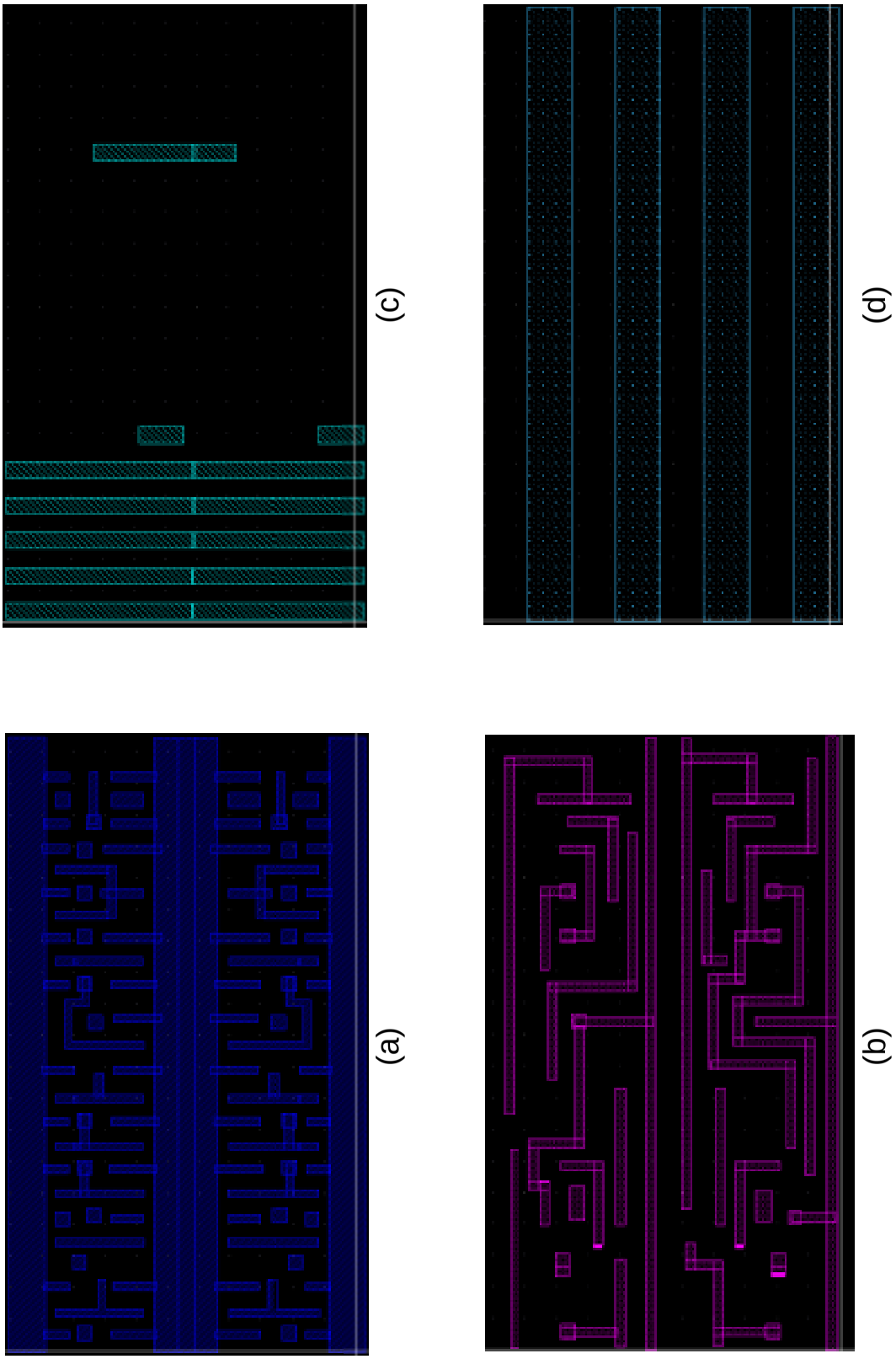

Figure 12. Interconnect Layout in crossbar cell. (a) Metal1 (b) Metal2 (c) Metal3 (d) Metal5

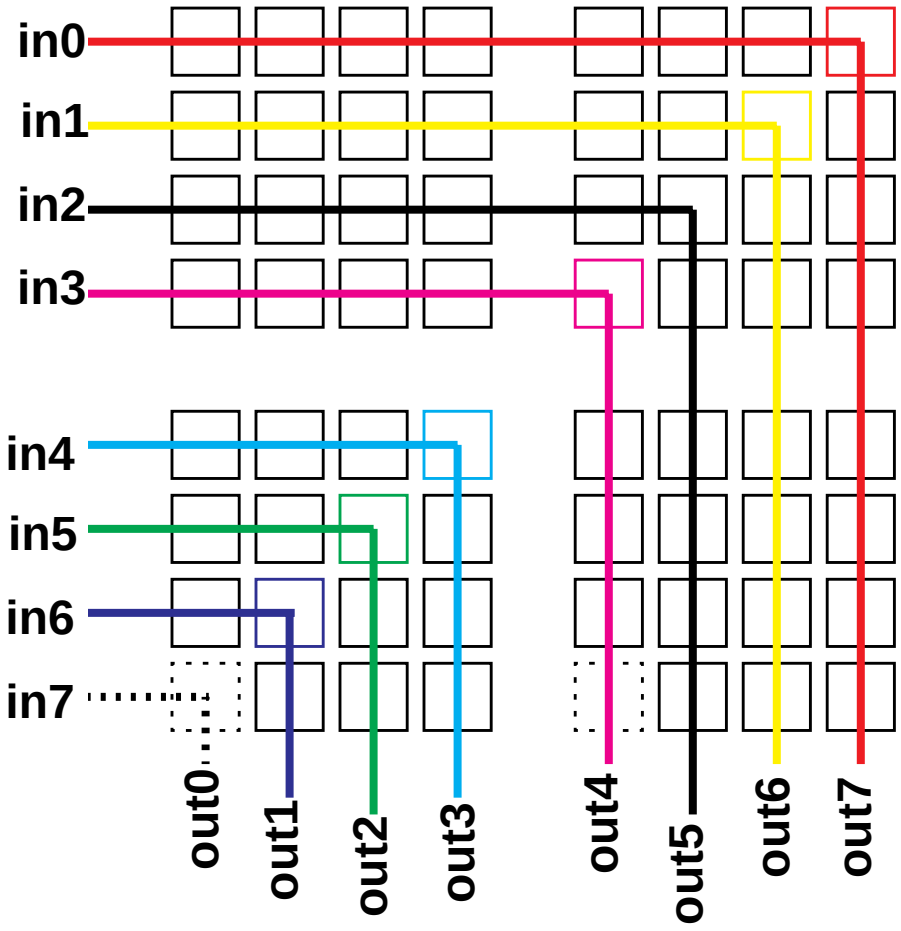

Figure 13. Schematic illustrating crossbar with programmed with I/O configuration discussed in Section VI of this report.

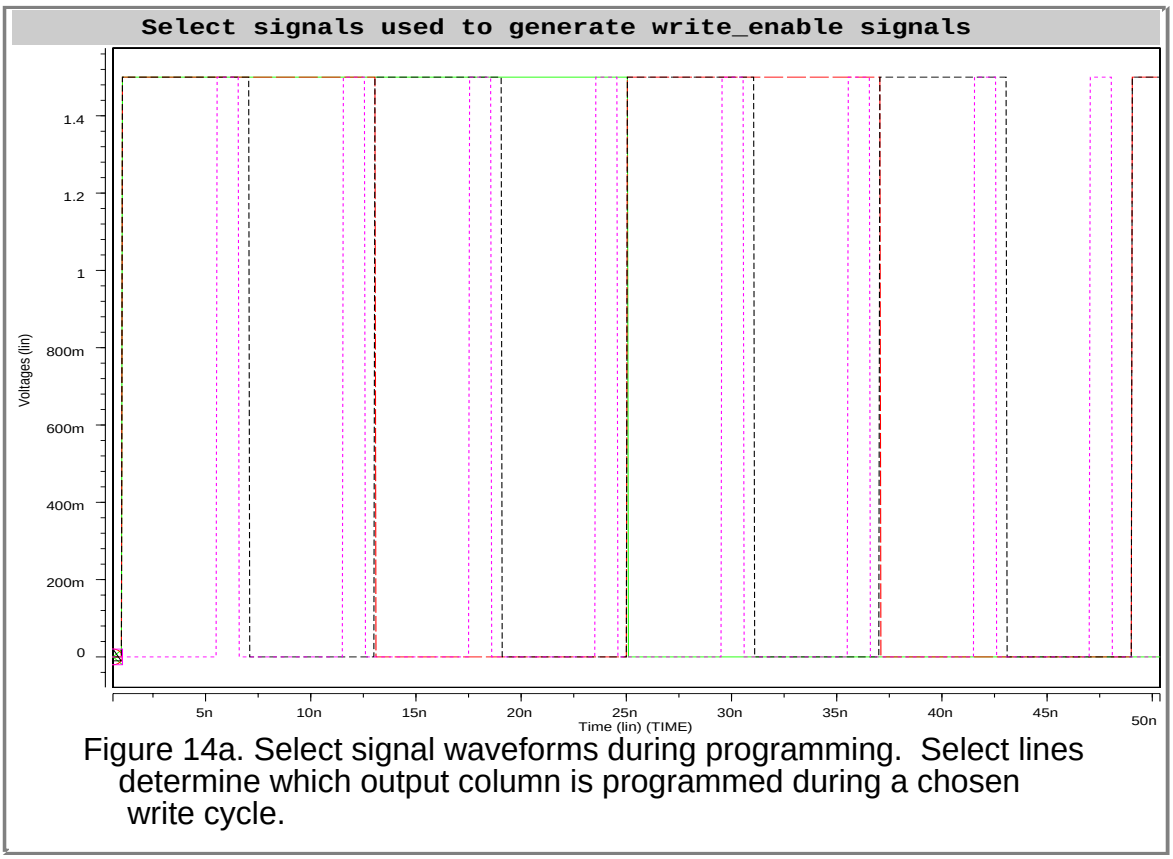

Figure 14a. Select signal waveforms during programming. Select lines determine which output column is programmed during a chosen write cycle.

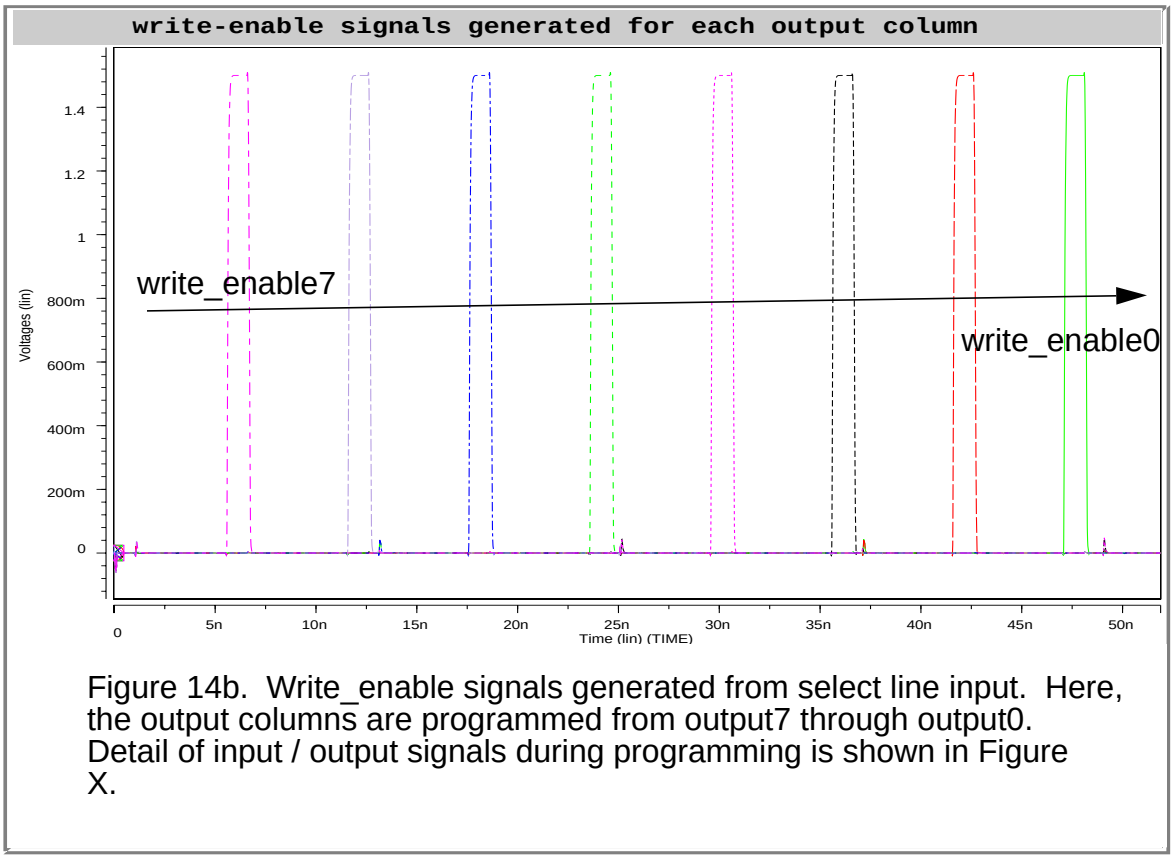

Figure 14b. Write_enable signals generated from select line input. Here, the output columns are programmed from output7 through output0. Detail of input / output signals during programming is shown in Figure X.

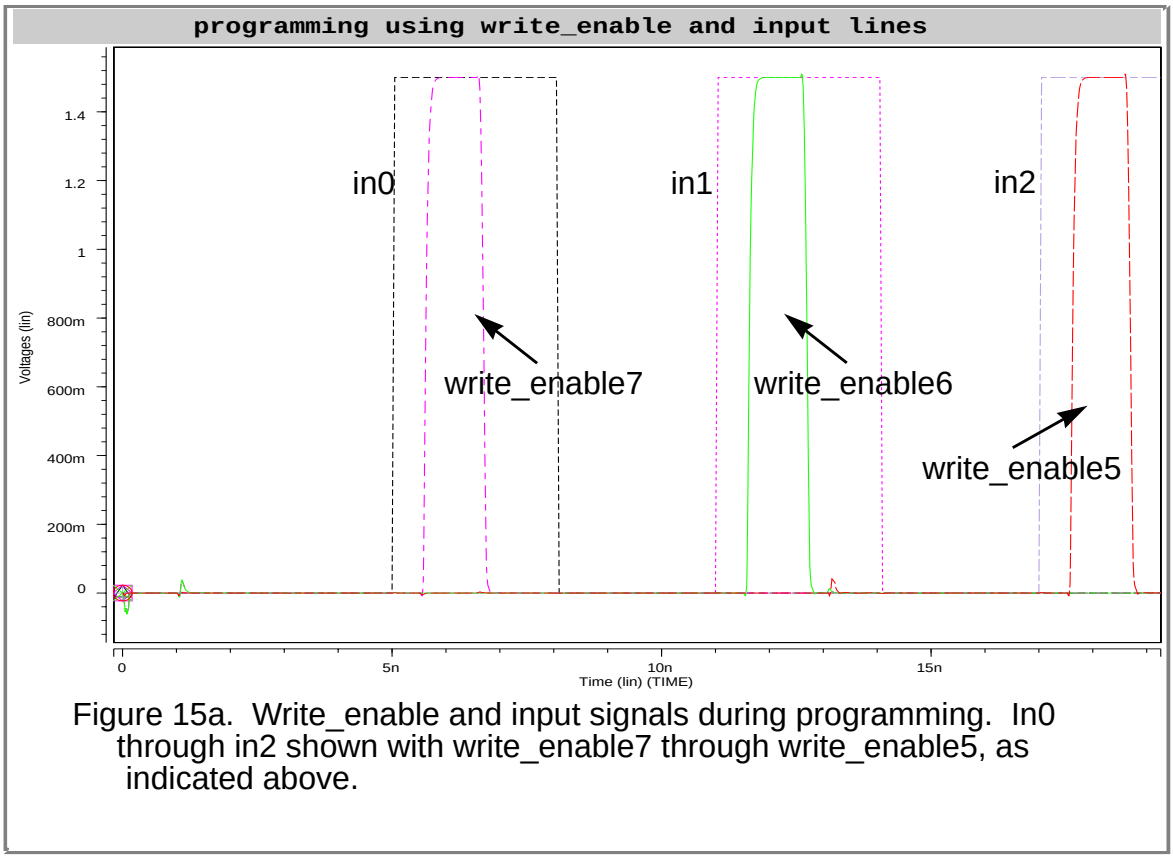

Figure 15a. Write_enable and input signals during programming. In0 through in2 shown with write_enable7 through write_enable5, as indicated above.

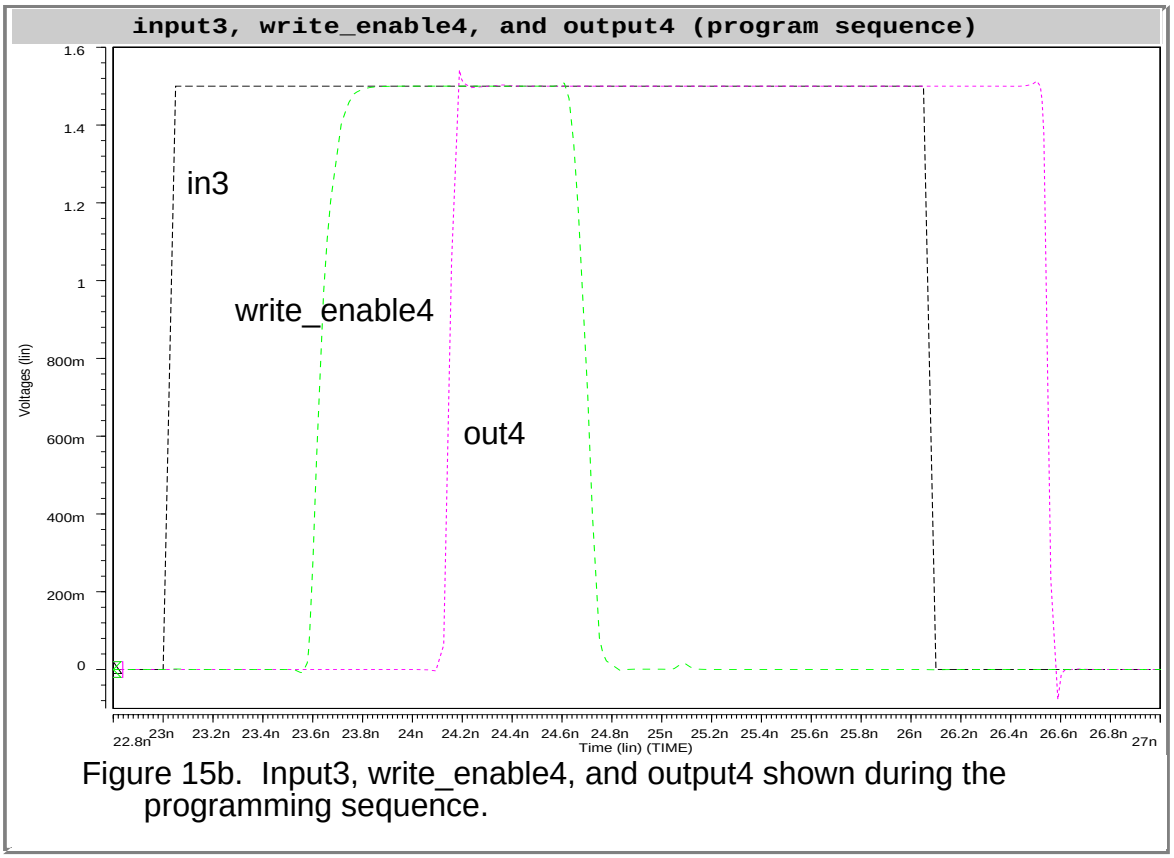

Figure 15b. Input3, write_enable4, and output4 shown during the programming sequence.

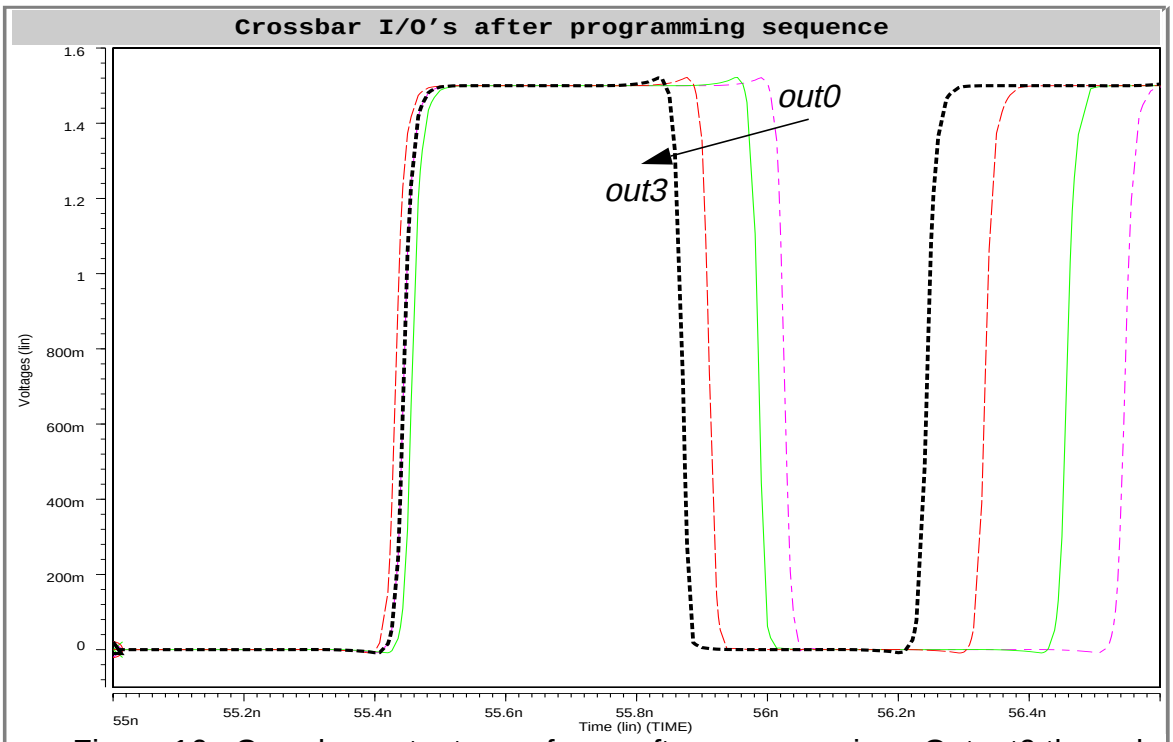

Figure 16. Crossbar output waveforms after programming. Output0 through output4 are shown above - a programmed connection has been made to input 7 through input4, resp.

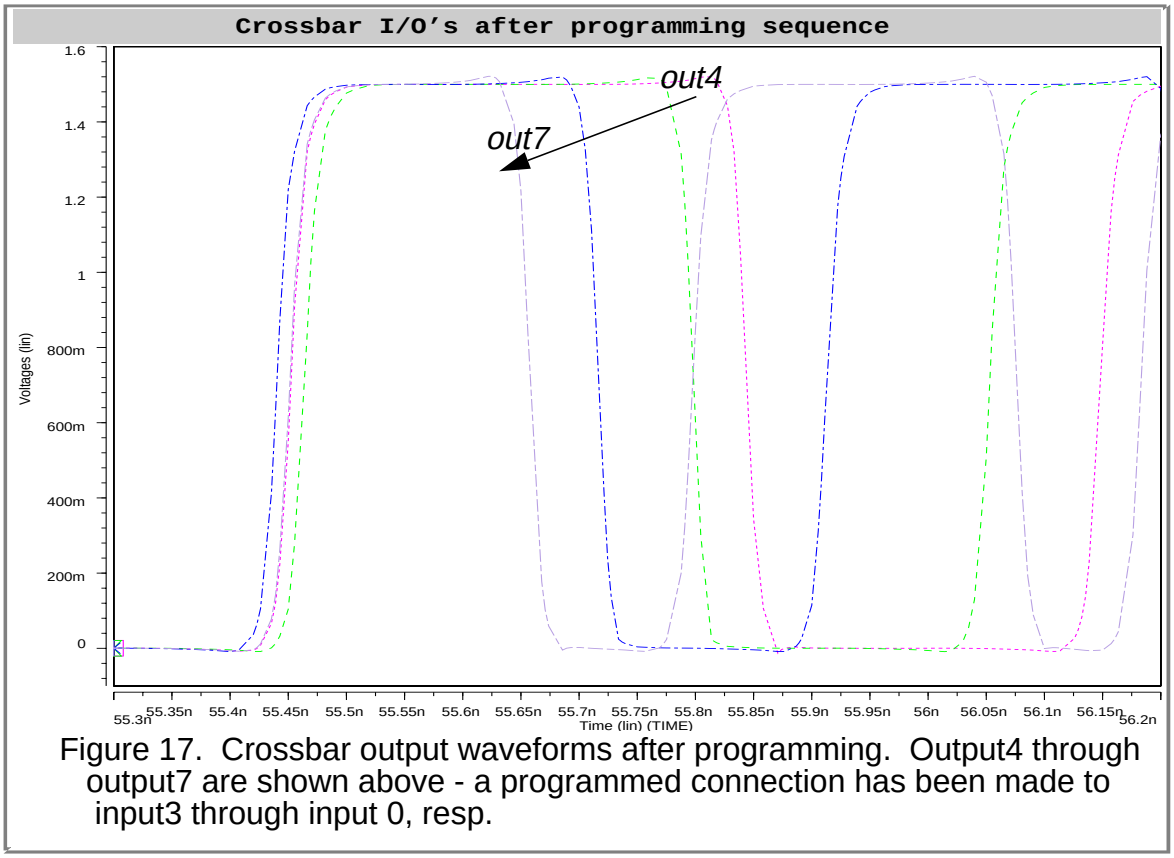

Figure 17. Crossbar output waveforms after programming. Output4 through output7 are shown above - a programmed connection has been made to input3 through input 0, resp.

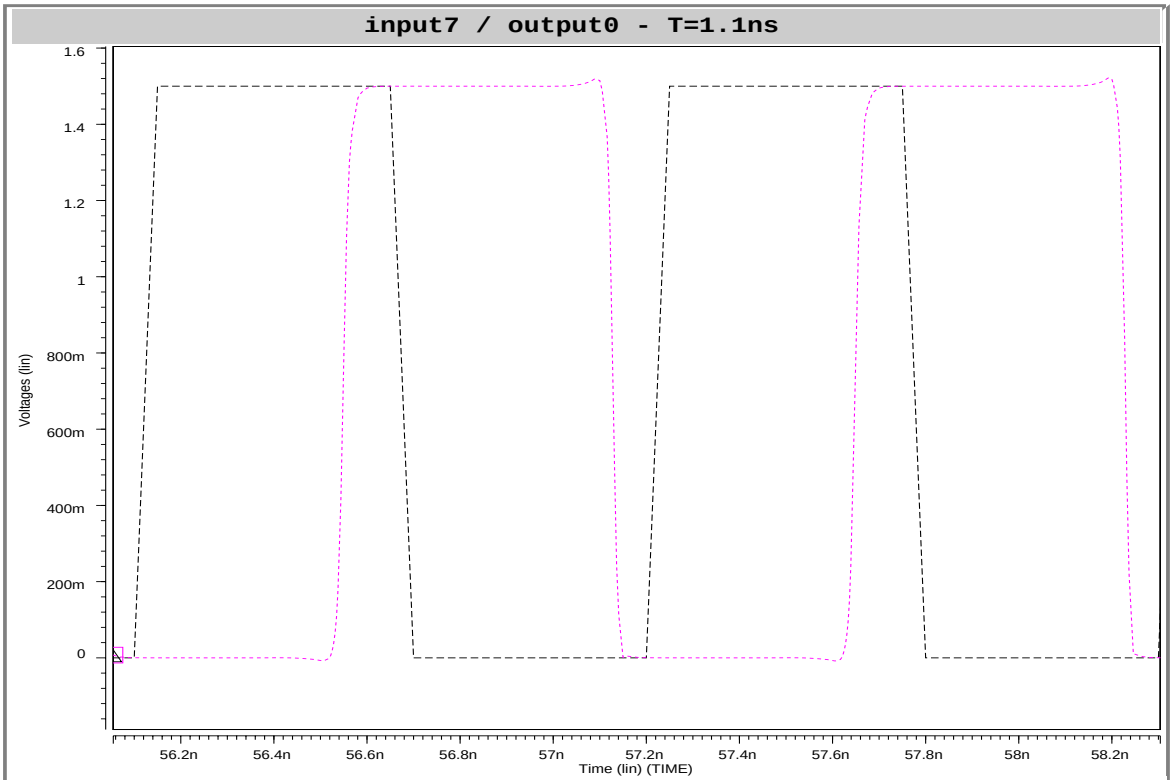

Figure 18. Input7 and output0 for programmed crossbar. Proper crossbar connection is made for 0.91GHz input signal.

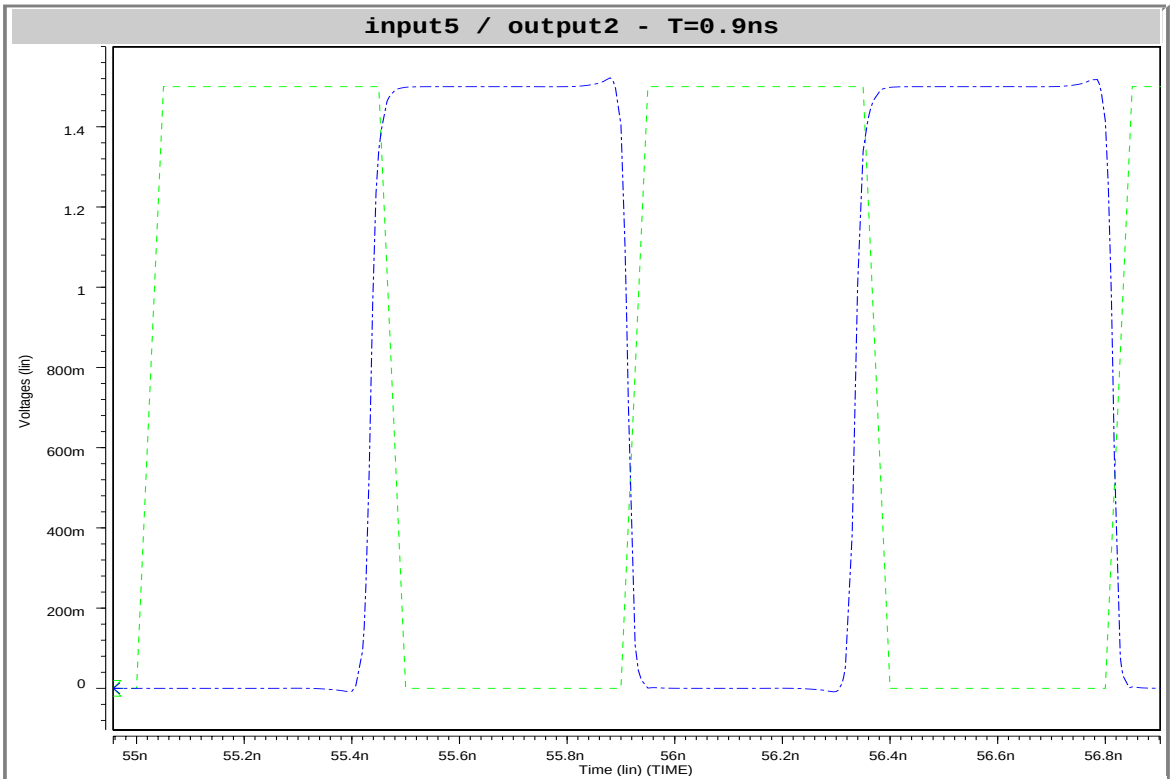

Figure 19. Input5 and output2 for programmed crossbar. Proper crossbar connection is made for1.1GHz input signal.

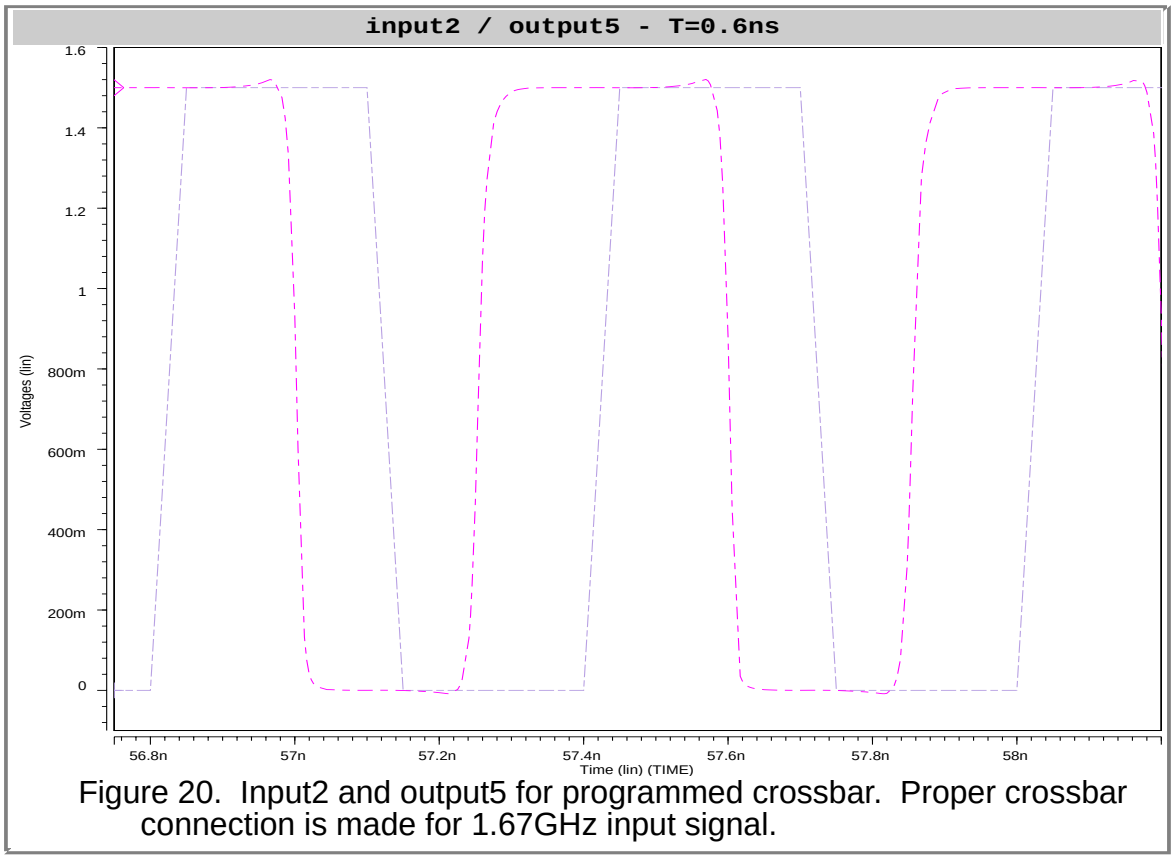

Figure 20. Input2 and output5 for programmed crossbar. Proper crossbar connection is made for 1.67GHz input signal.

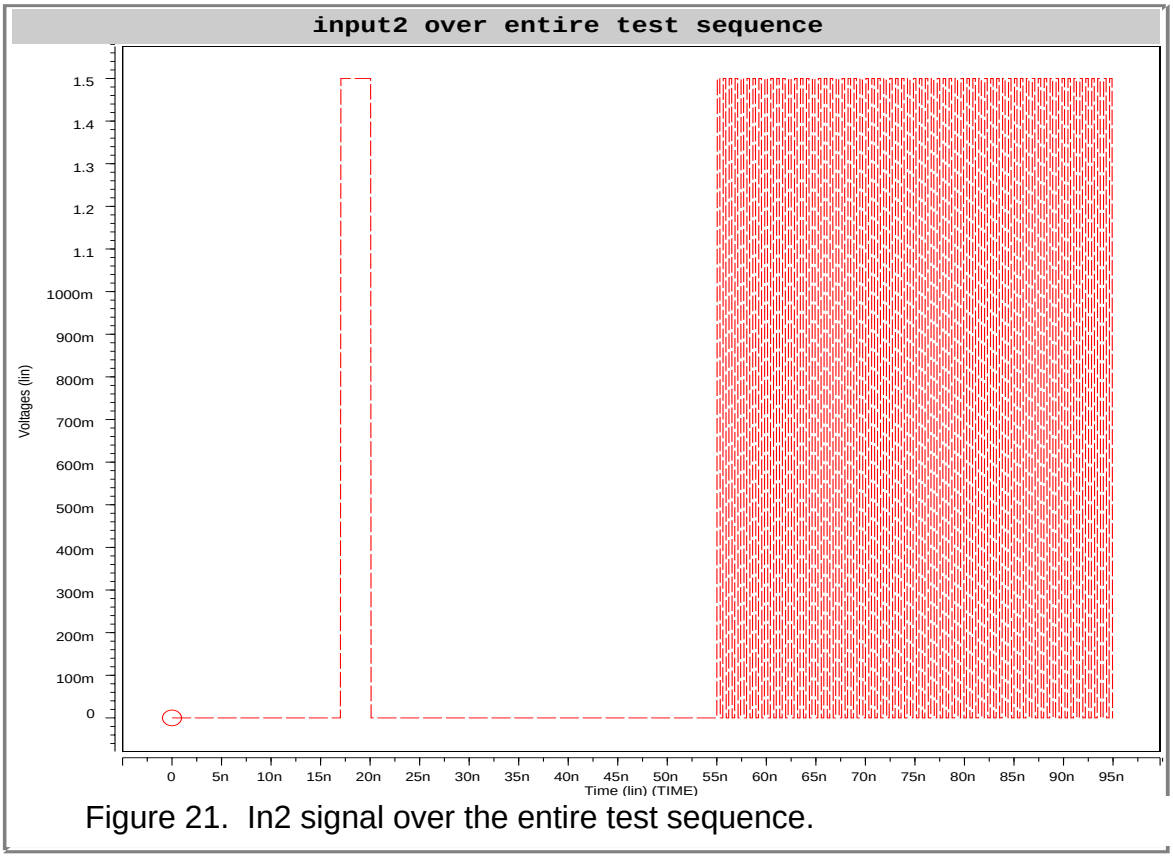

Figure 21. In2 signal over the entire test sequence.

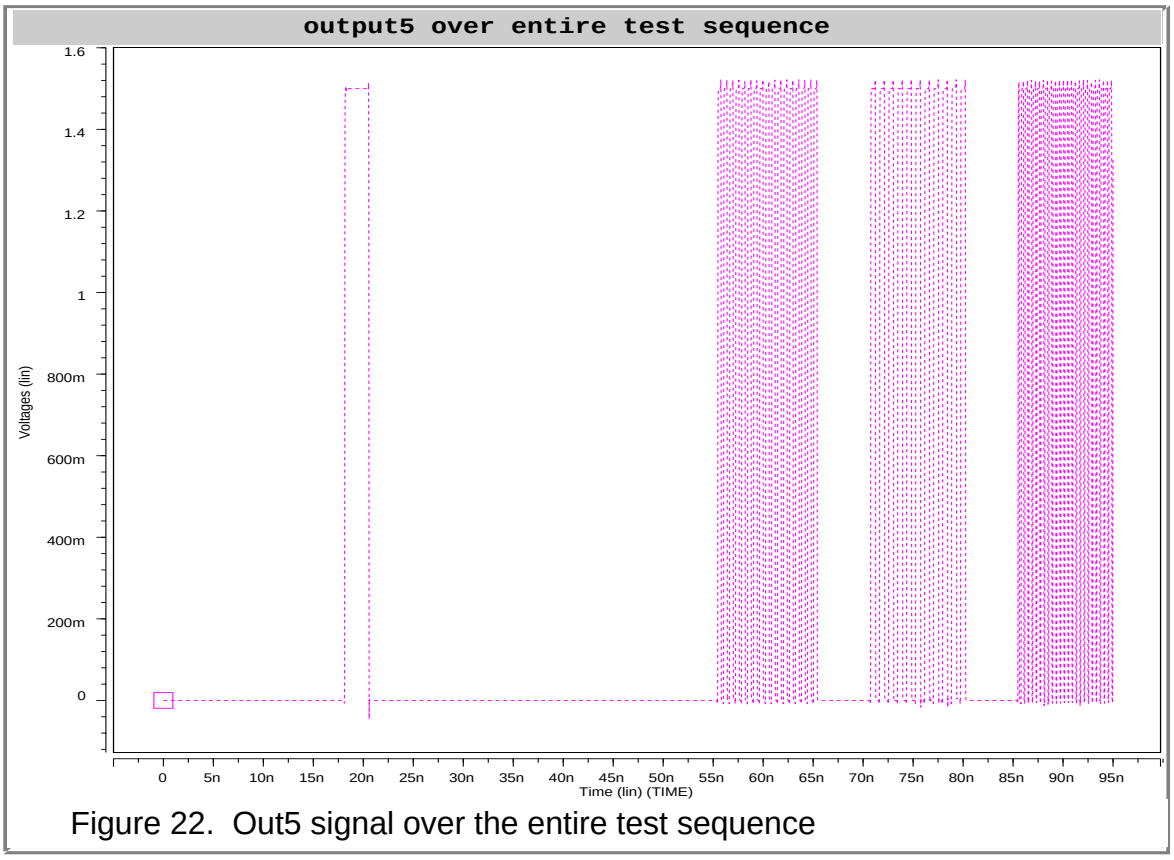

Figure 22. Out5 signal over the entire test sequence

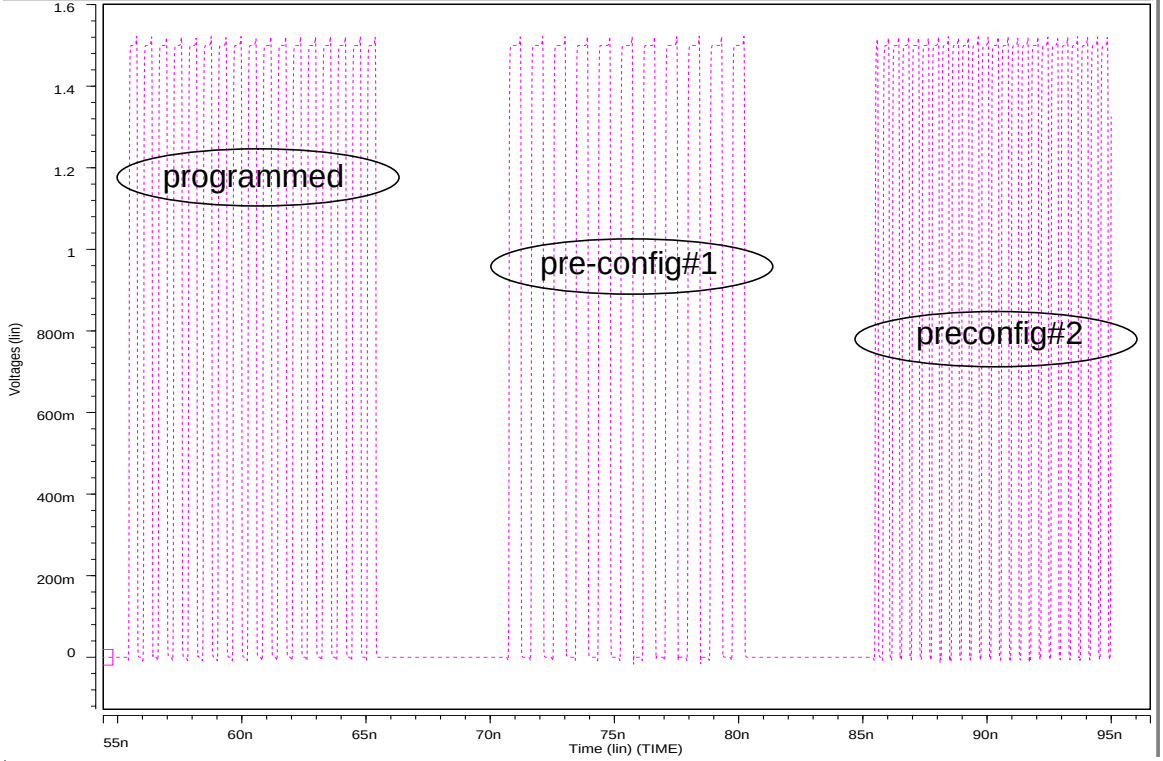

Figure 23. Output5 with programmed, pre-configure #1, & pre-configure #2 conditions

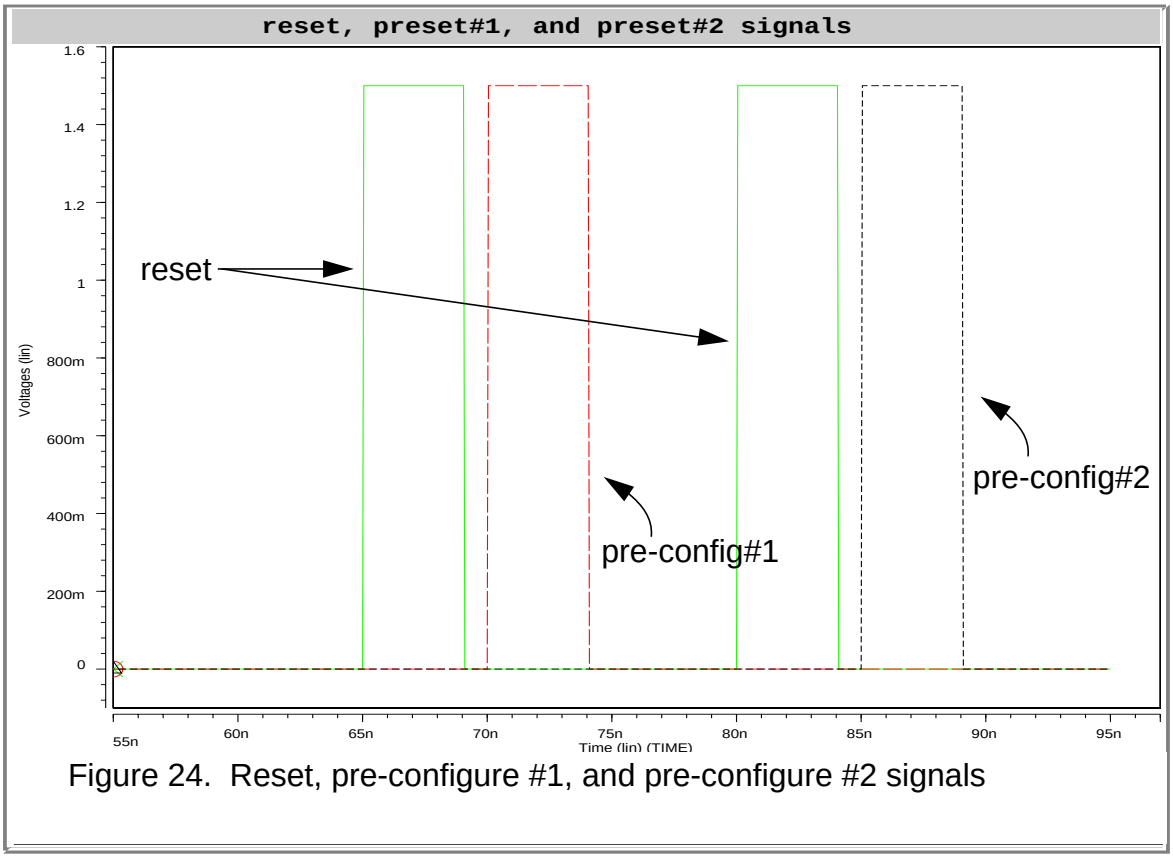

Figure 24. Reset, pre-configure #1, and pre-configure #2 signals

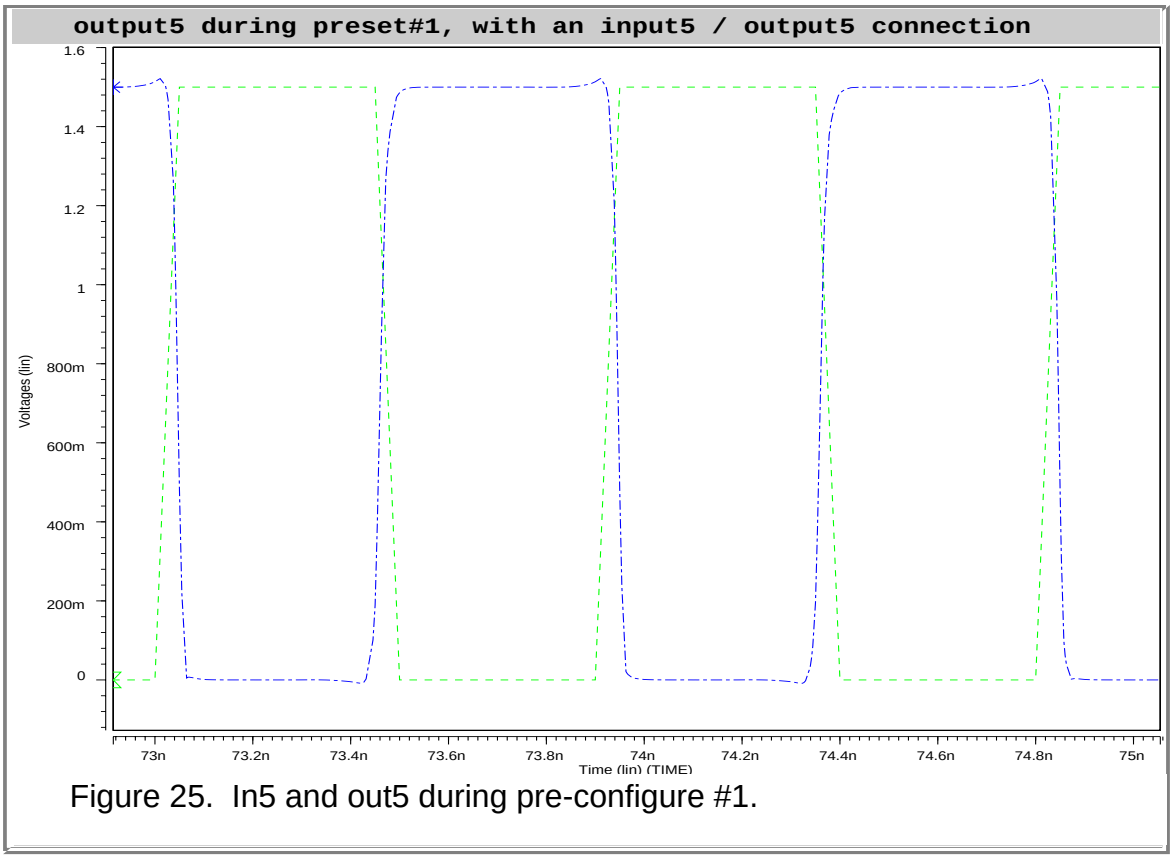

Figure 25. In5 and out5 during pre-configure #1.

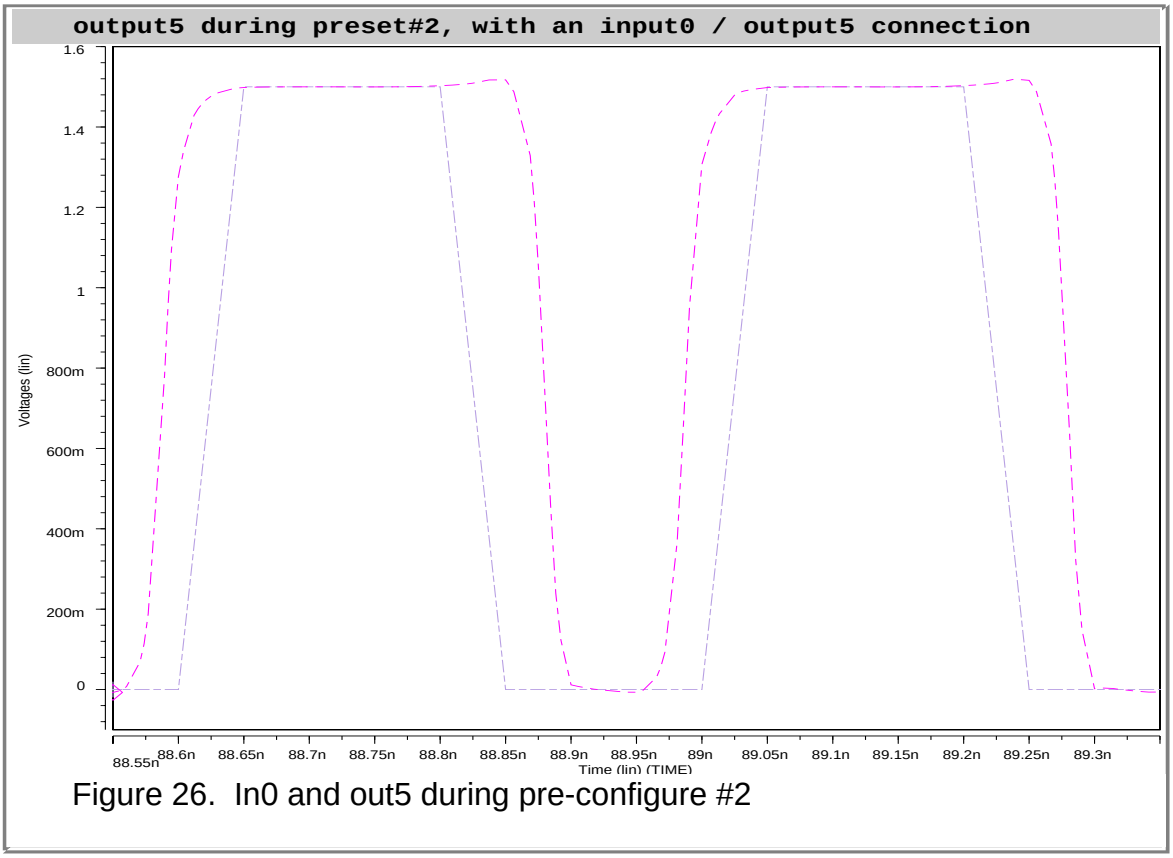

Figure 26. In0 and out5 during pre-configure #2

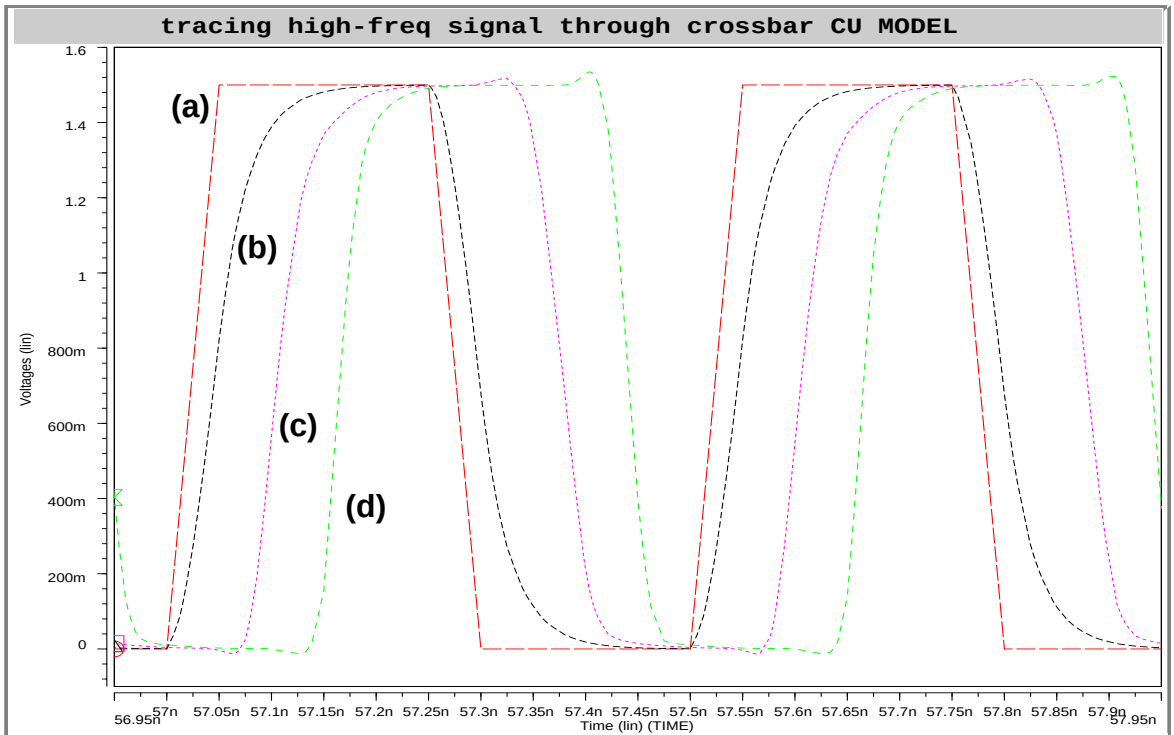

Figure 27a. Tracing a signal through the crossbar, Copper model. Input T=0.5ns. Nodes above are (a) input, (b) input to crossbar cell, (c) output of crossbar cell, and (d) output of initial OR gate.

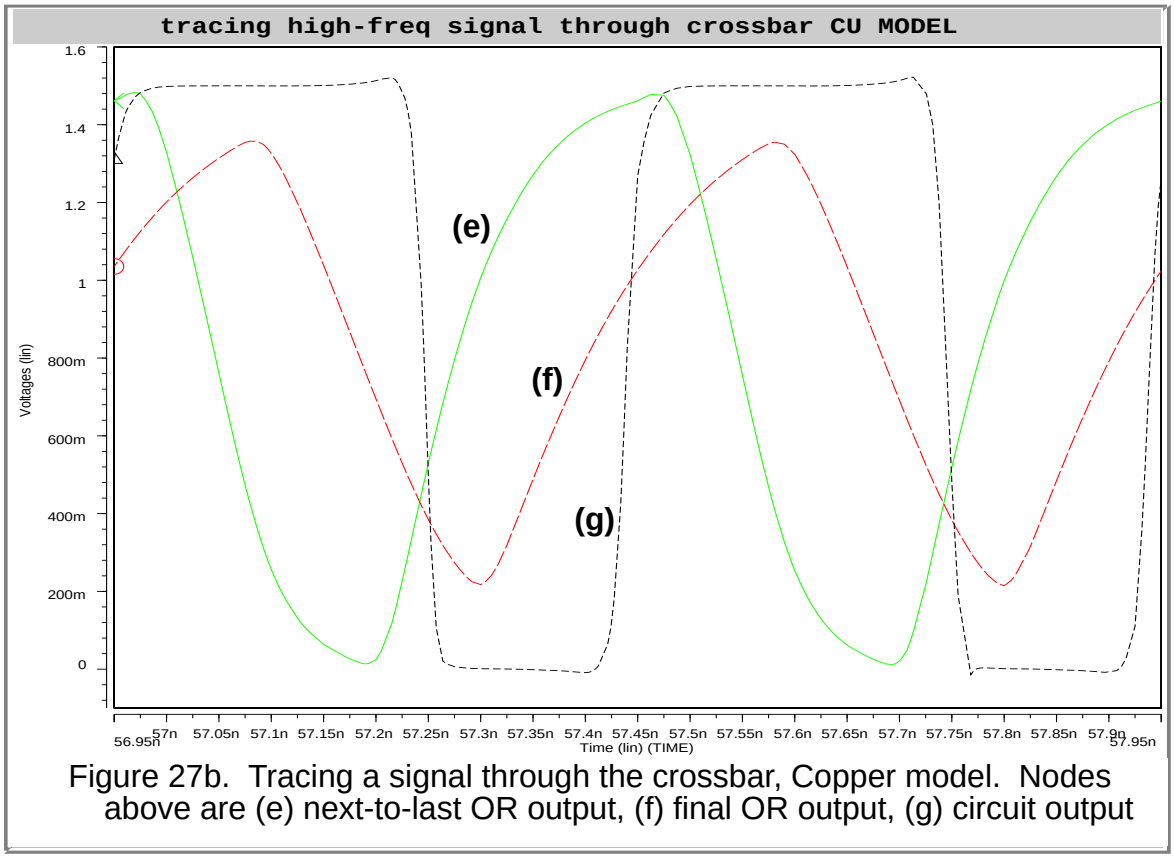

Figure 27b. Tracing a signal through the crossbar, Copper model. Nodes above are (e) next-to-last OR output, (f) final OR output, (g) circuit output

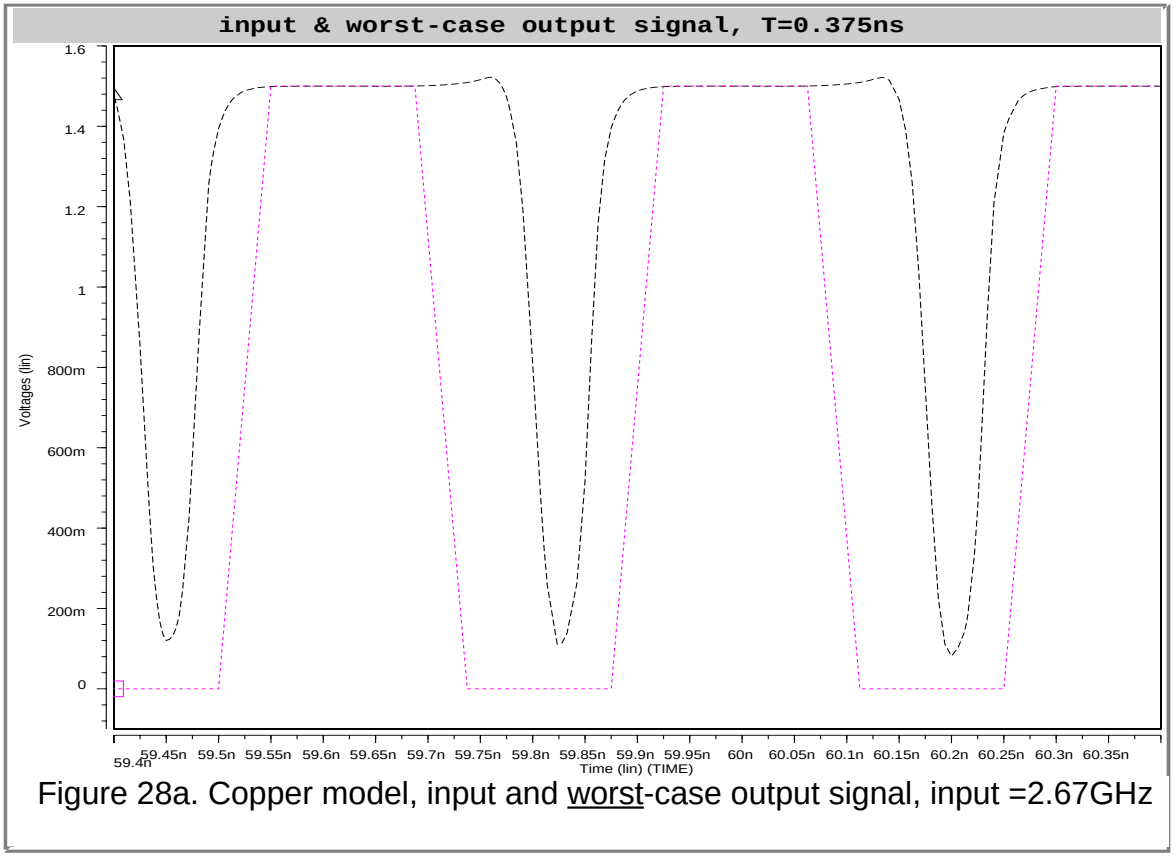

Figure 28a. Copper model, input and worst-case output signal, input =2.67GHz

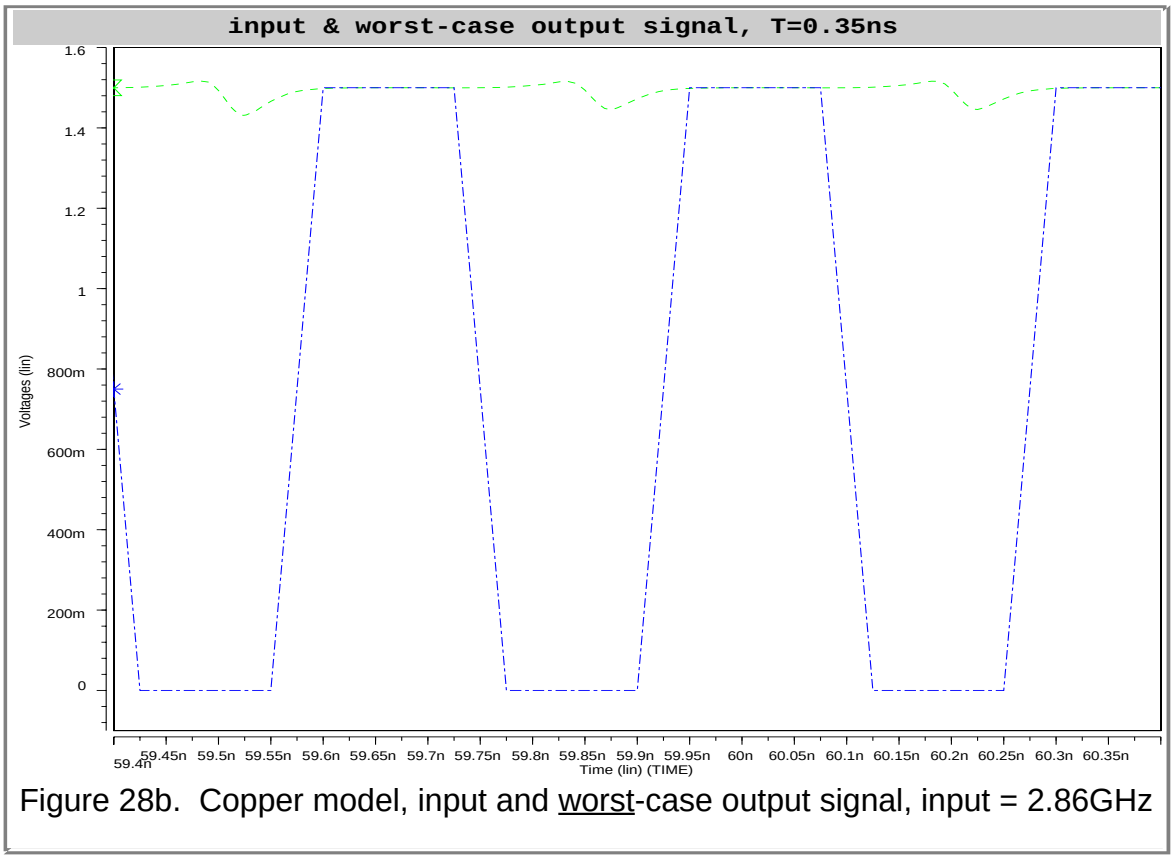

Figure 28b. Copper model, input and worst-case output signal, input = 2.86GHz

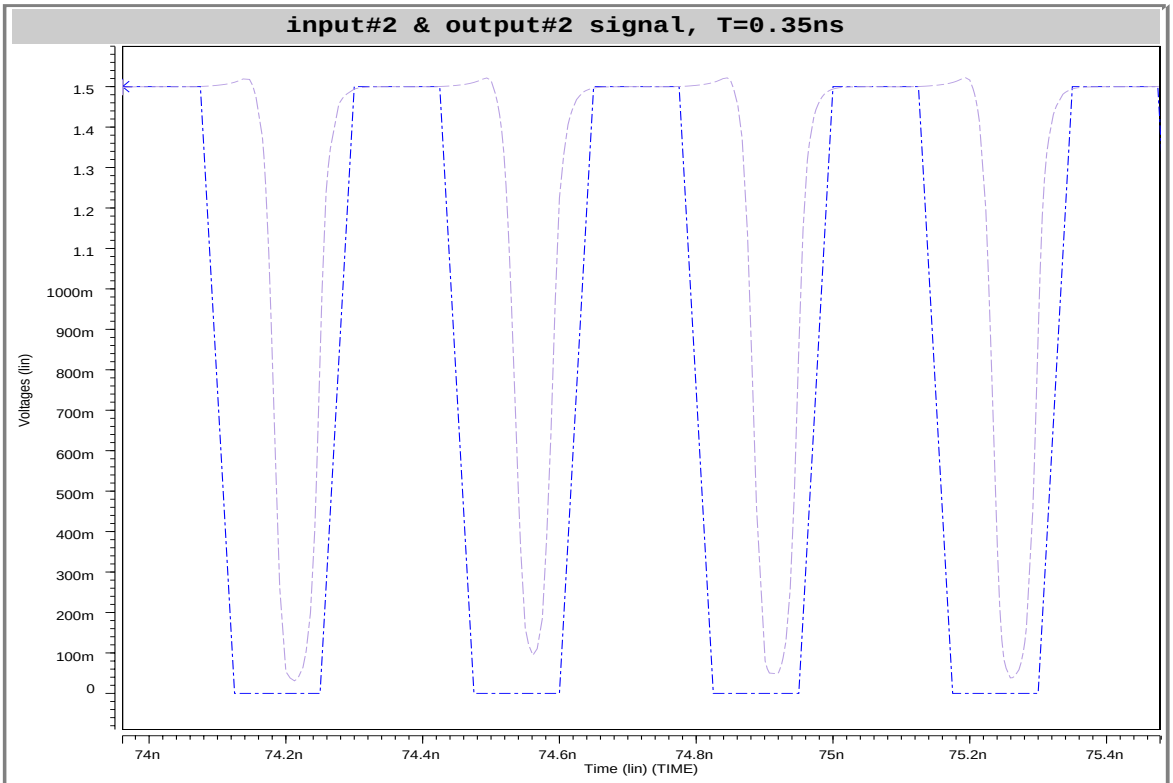

Figure 29. Copper model, input and best-case output signal, input =2.86GHz

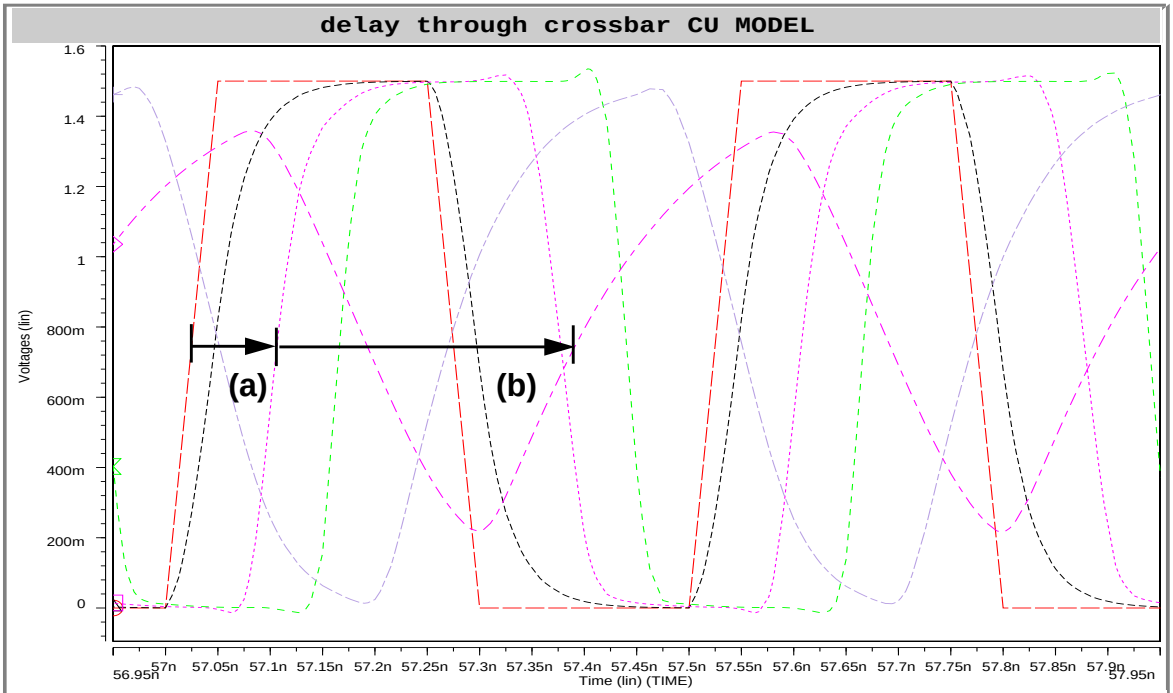

Figure 30. Measuring delay of Copper crossbar from input through OR tree. Delay includes (a) delay from input to crossbar cell and (b) delay through OR tree. OR-tree delay increases as waveforms degrade due to enhanced capacitive load on later stages.

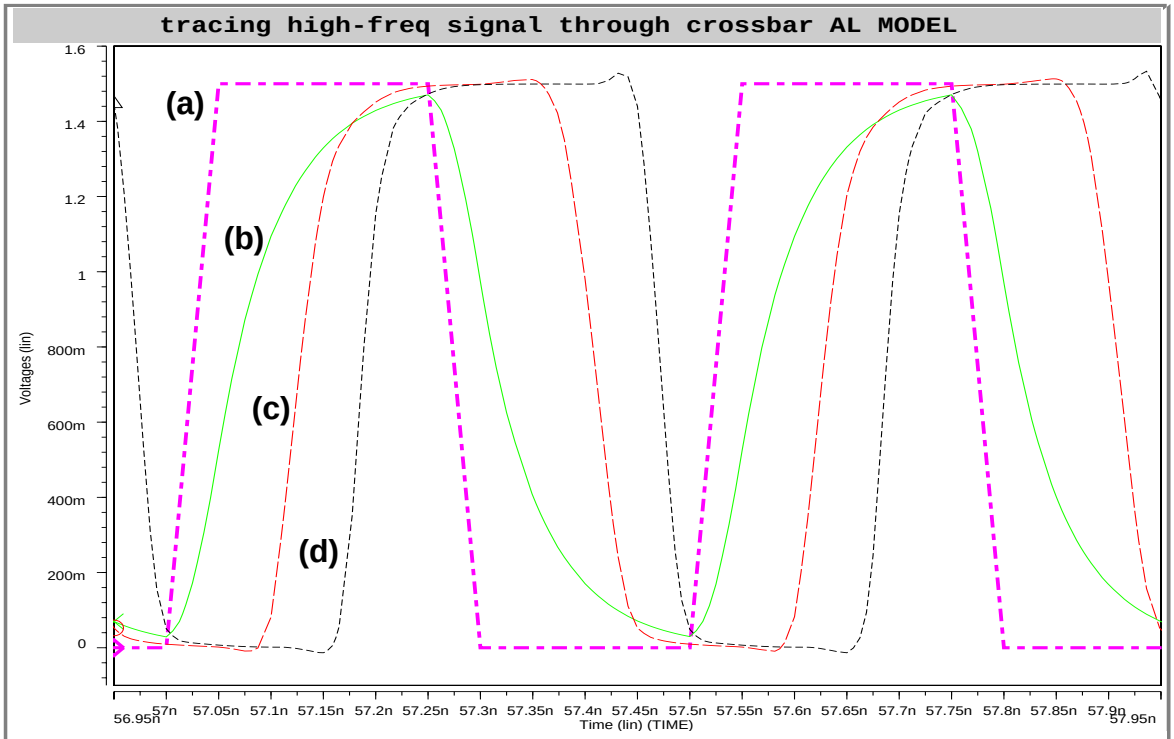

Figure 31a. Tracing a signal through the crossbar, Aluminum model. Input T=0.5ns. Nodes above are (a) input, (b) input to crossbar cell, (c) output of crossbar cell, and (d) output of initial OR gate.

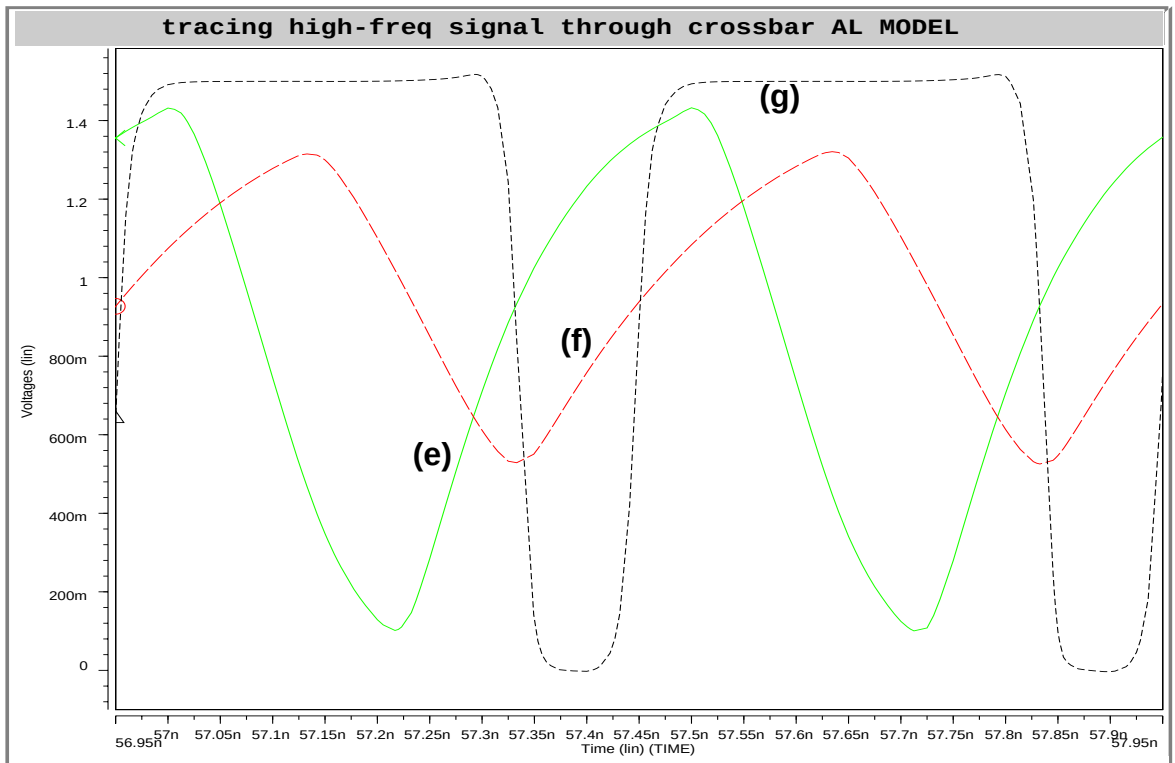

Figure 31b. Tracing a signal through the crossbar, Aluminum model. Nodes above are (e) next-to-last OR output, (f) final OR output, (g) circuit output

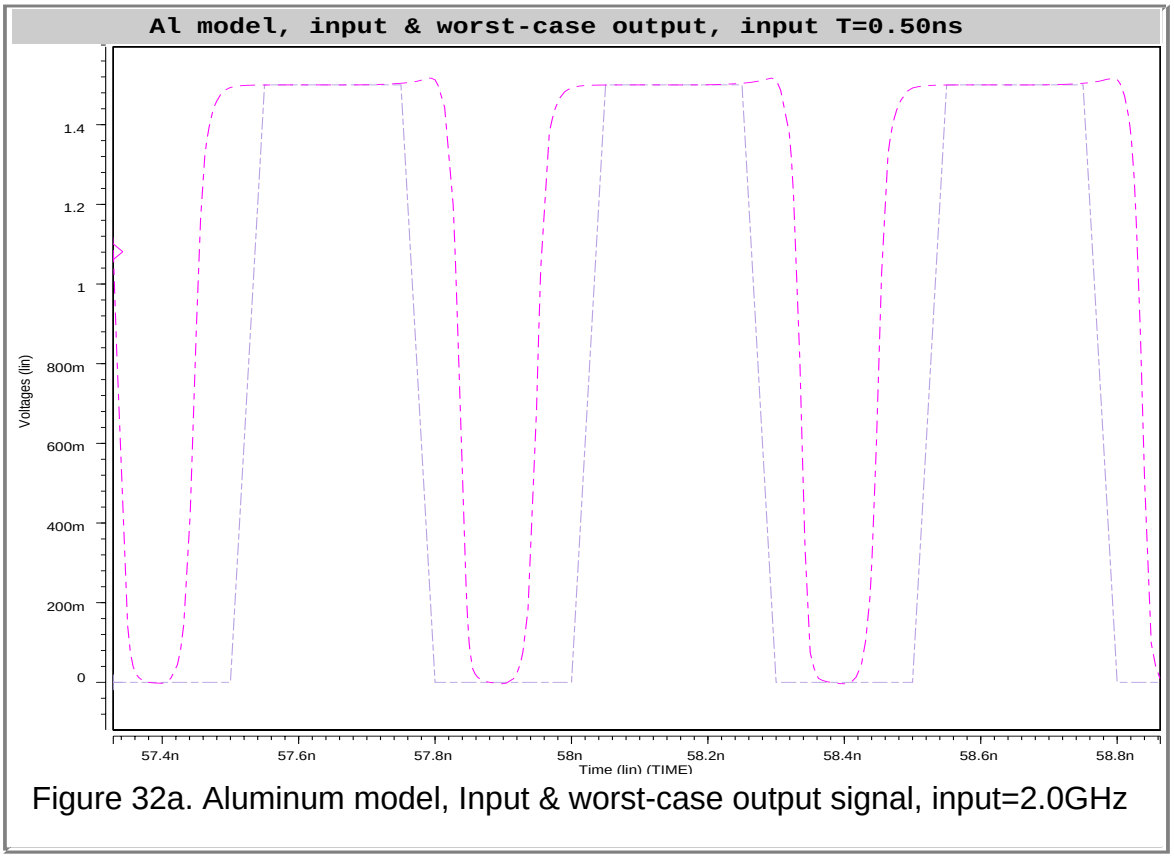

Figure 32a. Aluminum model, Input & worst-case output signal, input=2.0GHz

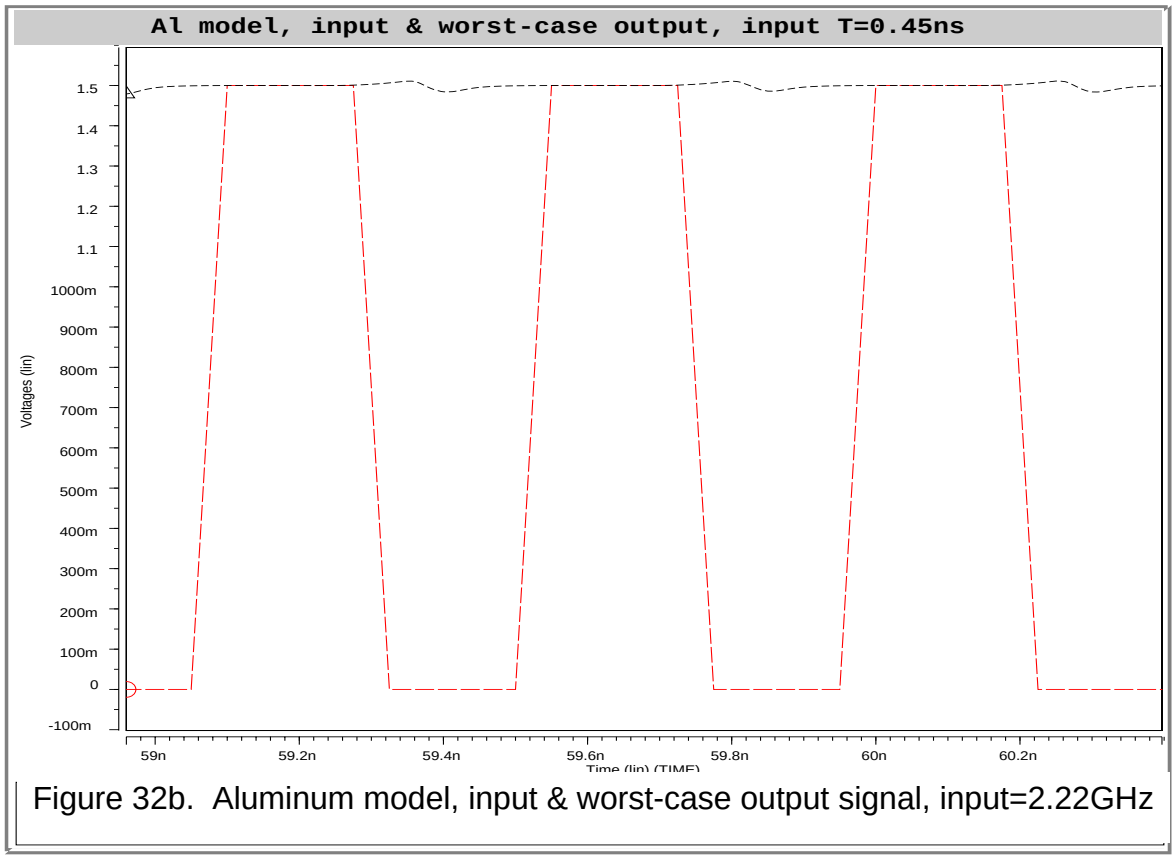

Figure 32b. Aluminum model, input & worst-case output signal, input=2.22GHz

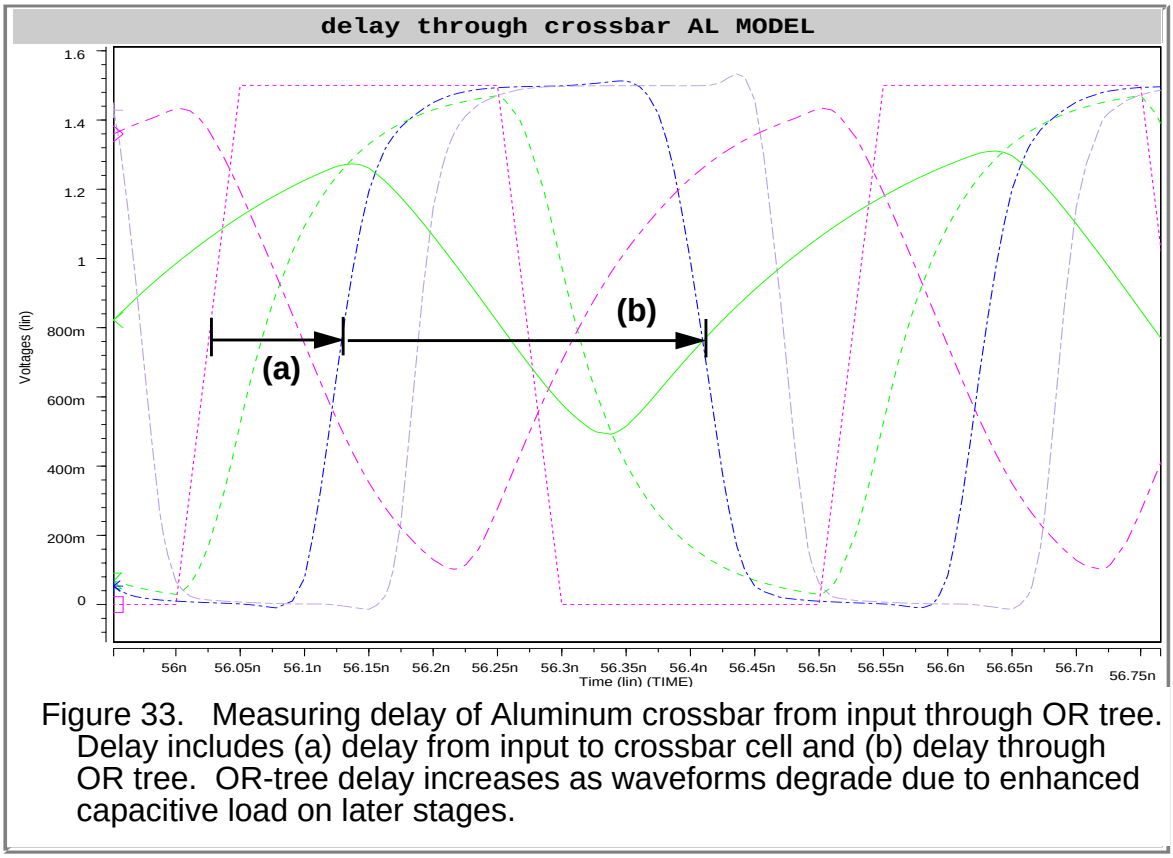

Figure 33. Measuring delay of Aluminum crossbar from input through OR tree. Delay includes (a) delay from input to crossbar cell and (b) delay through OR tree. OR-tree delay increases as waveforms degrade due to enhanced capacitive load on later stages.

Appendix A
HSPICE netlist for Copper Crossbar Circuit

```
* # FILE NAME: /AFS/EOS.NCSU.EDU/PROJECT/ERL/PAULF_GROUP/USER/JDAMIAN/
* cadence/simulation/8x8array/hspiceS/schematic/netlist/8x8array.c.raw
* Netlist output for hspiceS.
* Generated on Nov 22 10:45:13 1999
* global net definitions
* File name: copperstuff_8x8array_schematic.s.
* Subcircuit for cell: 8x8array.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:15 1999.

.GLOBAL VDD!
.options dccap NOMOD POST
.tran 0.05ns 95nS

* This design uses the 1.5V transistor models exclusively

V1 VDD! 0  1.5

* The following lines reference the write-enable circuitry  - the inputs are either
* from select lines or the inverted select input.  Each of eight combinations is
* represented, plus an additional write-enable (WE) bit.

XI504 SELA SELB SELC WE BL7 WE_CKT8_G5
XI503 SELA SELB NET714 WE BL6 WE_CKT8_G5
XI502 SELA NET716 SELC WE BL5 WE_CKT8_G5
XI501 SELA NET716 NET714 WE BL4 WE_CKT8_G5
XI500 NET718 SELB SELC WE BL3 WE_CKT8_G5
XI499 NET718 SELB NET714 WE BL2 WE_CKT8_G5
XI498 NET718 NET716 SELC WE BL1 WE_CKT8_G5
XI497 NET718 NET716 NET714 WE BL0 WE_CKT8_G5

*output drivers for each output lines

Xouta OUT0 INTER0 INV_BIG1_G3
Xoutb INTER0 IOUT0 INV_BIG1_G3
Xoutc OUT1 INTER1 INV_BIG1_G3
Xoutd INTER1 IOUT1 INV_BIG1_G3
Xoute OUT2 INTER2 INV_BIG1_G3
Xoutf INTER2 IOUT2 INV_BIG1_G3
Xoutg OUT3 INTER3 INV_BIG1_G3
Xouth INTER3 IOUT3 INV_BIG1_G3

Xouti OUT4 INTER4 INV_BIG1_G3
Xoutj INTER4 IOUT4 INV_BIG1_G3
Xoutk OUT5 INTER5 INV_BIG1_G3
```

```
Xoutl INTER5 IOUT5 INV_BIG1_G3
Xoutm OUT6 INTER6 INV_BIG1_G3
Xoutn INTER6 IOUT6 INV_BIG1_G3
Xouto OUT7 INTER7 INV_BIG1_G3
Xoutp INTER7 IOUT7 INV_BIG1_G3

* The following lines add RC values for the output lines
* copper values = 77fF/41.1, Al values = 112fF/41.1

C10 OUT0 0  77E-15 M=1.0
C11 OUT1 0  77E-15 M=1.0
C12 OUT2 0  77E-15 M=1.0
C13 OUT3 0  77E-15 M=1.0
C14 OUT4 0  77E-15 M=1.0
C15 OUT5 0  77E-15 M=1.0
C16 OUT6 0  77E-15 M=1.0
C17 OUT7 0  77E-15 M=1.0

R11 OUT0 NET1196  41.1 M=1.0
R12 OUT1 NET589  41.1 M=1.0
R13 OUT2 NET591  41.1 M=1.0
R14 OUT3 NET593  41.1 M=1.0
R15 OUT4 NET595  41.1 M=1.0
R16 OUT5 NET597  41.1 M=1.0
R17 OUT6 NET599  41.1 M=1.0
R18 OUT7 NET1187  41.1 M=1.0

*The following add RC values to the 2nd longest branches of the OR tree
*copper values = 38fF/20, Al values = 56fF/20

*out0
Clump1 net1106 0 38E-15 M=1.0
Clump2 net1088 0 38E-15 M=1.0
Rlump1 net1106 netl1 20.0 M=1.0
Rlump2 net1088 netl2 20.0 M=1.0
*out1
Clump3 net642 0 38E-15 M=1.0
Clump4 net624 0 38E-15 M=1.0
Rlump3 net642 netl3 20.0 M=1.0
Rlump4 net624 netl4 20.0 M=1.0
*out2
Clump5 net1026 0 38E-15 M=1.0
Clump6 net1040 0 38E-15 M=1.0
Rlump5 net1026 netl5 20.0 M=1.0
Rlump6 net1040 netl6 20.0 M=1.0
*out3
Clump7 net850 0 38E-15 M=1.0
Clump8 net864 0 38E-15 M=1.0
Rlump7 net850 netl7 20.0 M=1.0
Rlump8 net864 netl8 20.0 M=1.0
*out4
Clump9 net794 0 38E-15 M=1.0
Clump10 net776 0 38E-15 M=1.0
Rlump9 net794 netl9 20.0 M=1.0
```

```
Rlump10 net776 netl10 20.0 M=1.0
*out5
Clump11 net898 0 38E-15 M=1.0
Clump12 net912 0 38E-15 M=1.0
Rlump11 net898 netl11 20.0 M=1.0
Rlump12 net912 netl12 20.0 M=1.0
*out6
Clump13 net722 0 38E-15 M=1.0
Clump14 net736 0 38E-15 M=1.0
Rlump13 net722 netl13 20.0 M=1.0
Rlump14 net736 netl14 20.0 M=1.0
*out7
Clump15 net978 0 38E-15 M=1.0
Clump16 net960 0 38E-15 M=1.0
Rlump15 net978 netl15 20.0 M=1.0
Rlump16 net960 netl16 20.0 M=1.0

* The following lines are the RC constants for the input line, including
* device and interconnect RCs with input lines on M5
* copper values = 705fF/38, Al values = 1478fF/38

C0 NET543 0  705E-15 M=1.0
C3 NET545 0  705E-15 M=1.0
C4 NET547 0  705E-15 M=1.0
C5 NET532 0  705E-15 M=1.0
C6 NET536 0  705E-15 M=1.0
C7 NET538 0  705E-15 M=1.0
C8 NET540 0  705E-15 M=1.0
C9 NET542 0  705E-15 M=1.0

R0 IN7 NET543  38.0 M=1.0
R4 IN6 NET545  38.0 M=1.0
R5 IN5 NET547  38.0 M=1.0
R6 IN4 NET532  38.0 M=1.0
R7 IN3 NET536  38.0 M=1.0
R8 IN2 NET538  38.0 M=1.0
R9 IN1 NET540  38.0 M=1.0
R10 IN0 NET542  38.0 M=1.0

* The following are 0V voltage sources inserted to measure currents through the output lines.

V29 NET1019 NET1187  0.0
V31 NET835 NET595  0.0
V32 NET939 NET597  0.0
V33 NET1147 NET1196  0.0
V34 NET1617 NET593  0.0
V35 NET1067 NET591  0.0
V36 NET683 NET589  0.0
V30 NET763 NET599  0.0

* The following generate the write-enable inputs used, with the input lines, to
* program the crossbar circuit.  The pulses on the WE line are conincident with input
* signals below.
```

```
V41 SELA 0  PULSE 0.0 1.5 1E-9 50E-12 50E-12 24E-9 1E-6
V42 SELB 0  PULSE 0.0 1.5 1E-9 50E-12 50E-12 12E-9 24E-9
V44 SELC 0  PULSE 0.0 1.5 1E-9 50E-12 50E-12 6E-9 12E-9
V43 WE netp  PULSE 0.0 1.5 5.5E-9 50E-12 50E-12 1E-9 1E-6
vw1 netp netq pulse 0.0 1.5 11.5E-9 50E-12 50E-12 1E-9 1E-6
vw2 netq netr pulse 0.0 1.5 17.5E-9 50E-12 50E-12 1E-9 1E-6
vw3 netr nets pulse 0.0 1.5 23.5E-9 50E-12 50E-12 1E-9 1E-6
vw4 nets nett pulse 0.0 1.5 29.5E-9 50E-12 50E-12 1E-9 1E-6
vw5 nett netu pulse 0.0 1.5 35.5E-9 50E-12 50E-12 1E-9 1E-6
vw6 netu netv pulse 0.0 1.5 41.5E-9 50E-12 50E-12 1E-9 1E-6
vw7 netv 0 pulse 0.0 1.5 47E-9 50.5E-12 50E-12 1E-9 1E-6

* The following are the test input lines consisting of (a) a programming signal related
* to the above WE inputs, and (b) a test signal with each input have different freq. to enable
* tracing the signal through the crossbar and studying the freq. range of the crossbar

V14 IN0 NET586  PULSE 0.0 1.5 5E-9 50E-12 50E-12 3E-9 1E-6
V13 NET586 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 150E-12 400E-12

V16 IN1 NET578  PULSE 0.0 1.5 11E-9 50E-12 50E-12 3E-9 1E-6
V15 NET578 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 175E-12 450E-12

V11 IN2 NET587  PULSE 0.0 1.5 17E-9 50E-12 50E-12 3E-9 1E-6
V12 NET587 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 200E-12 500E-12

V9 IN3 NET601  PULSE 0.0 1.5 23E-9 50E-12 50E-12 3E-9 1E-6
V10 NET601 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 225E-12 550E-12

V6 IN4 NET600  PULSE 0.0 1.5 29E-9 50E-12 50E-12 3E-9 1E-6
V37 NET600 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 250E-12 600E-12

V8 IN5 NET603  PULSE 0.0 1.5 35E-9 50E-12 50E-12 3E-9 1E-6
V38 NET603 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 275E-12 650E-12

V3 IN6 NET615  PULSE 0.0 1.5 41E-9 50E-12 50E-12 3E-9 1E-6
V39 NET615 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 300E-12 700E-12

V2 IN7 NET576  PULSE 0.0 1.5 47E-9 50E-12 50E-12 3E-9 1E-6
V40 NET576 0  PULSE 0.0 1.5 55E-9 50E-12 50E-12 325E-12 750E-12

* The following voltage sources are used to test reset/pre-configure functions

V28 RESET NET608  PULSE 0.0 1.5 65E-9 50E-12 50E-12 4E-9 15E-9
V27 NET608 0  PULSE 0.0 1.5 0.0 50E-12 50E-12 3E-9 1E-6
V26 PRESET2 0  PULSE 0.0 1.5 85E-9 50E-12 50E-12 4E-9 1E-6
V25 PRESET1 0  PULSE 0.0 1.5 70E-9 50E-12 50E-12 4E-9 1E-6
XI442 PRESET2 PRESET1 NET621 OR_G1
XI443 NET621 NET1132 INV_G4

* These inverters provide inverted select line input for the write-enable circuits

XI507 SELC NET714 INV_G4
XI506 SELB NET716 INV_G4
XI505 SELA NET718 INV_G4
```

* Unused OR gates at the bottom crossbar cell are tied to ground to prevent SPICE errors

va orin01 0 0
vb orin02 0 0
vc orin11 0 0
vd orin12 0 0
ve orin21 0 0
vf orin22 0 0
vg orin31 0 0
vh orin32 0 0
vi orin41 0 0
vj orin42 0 0
vk orin51 0 0
vl orin52 0 0
vm orin61 0 0
vn orin62 0 0
vo orin71 0 0
vp orin72 0 0

* The crossbar array composed of individual crossbar cells

XI244 BL1 IN5 NET626 NET648 NET624 NET623 0 RESET CROSSBAR_CELL_G6
XI245 BL1 IN4 NET623 NET631 NET626 NET631 0 RESET CROSSBAR_CELL_G6
XI246 BL1 IN3 NETl3 NETl4 NET683 NET639 0 RESET CROSSBAR_CELL_G6
XI247 BL1 IN6 NET671 NET647 NET648 NET647 0 RESET CROSSBAR_CELL_G6
XI248 BL1 IN1 NET658 NET680 NET642 NET655 PRESET1 RESET CROSSBAR_CELL_G6
XI249 BL1 IN0 NET655 NET663 NET658 NET663 PRESET2 RESET CROSSBAR_CELL_G6
XI243 BL1 IN7 ORIN11 ORIN12 OROUT1 NET671 0 RESET CROSSBAR_CELL_G6
XI242 BL1 IN2 NET639 NET679 NET680 NET679 0 RESET CROSSBAR_CELL_G6
XI390 BL7 NET542 NET991 NET687 NET994 NET687 PRESET2 RESET CROSSBAR_CELL_G6
XI389 BL6 NET542 NET703 NET695 NET706 NET695 PRESET2 RESET CROSSBAR_CELL_G6
XI388 BL6 NET540 NET706 NET752 NET722 NET703 0 RESET CROSSBAR_CELL_G6
XI387 BL6 NET545 NET743 NET711 NET712 NET711 PRESET1 RESET CROSSBAR_CELL_G6
XI386 BL6 NET536 NETl13 NETl14 NET763 NET719 0 RESET CROSSBAR_CELL_G6
XI385 BL6 NET532 NET735 NET727 NET738 NET727 0 RESET CROSSBAR_CELL_G6
XI384 BL6 NET547 NET738 NET712 NET736 NET735 0 RESET CROSSBAR_CELL_G6
XI383 BL6 NET543 ORIN61 ORIN62 OROUT6 NET743 0 RESET CROSSBAR_CELL_G6
XI382 BL6 NET538 NET719 NET751 NET752 NET751 0 RESET CROSSBAR_CELL_G6
XI312 BL4 NET538 NET791 NET759 NET760 NET759 0 RESET CROSSBAR_CELL_G6
XI313 BL4 NET543 ORIN41 ORIN42 OROUT4 NET767 0 RESET CROSSBAR_CELL_G6
XI314 BL4 NET547 NET778 NET800 NET776 NET775 0 RESET CROSSBAR_CELL_G6
XI315 BL4 NET532 NET775 NET783 NET778 NET783 PRESET1 RESET CROSSBAR_CELL_G6
XI316 BL4 NET536 NETl9 NETl10 NET835 NET791 0 RESET CROSSBAR_CELL_G6
XI317 BL4 NET545 NET767 NET799 NET800 NET799 0 RESET CROSSBAR_CELL_G6
XI318 BL4 NET540 NET810 NET760 NET794 NET807 0 RESET CROSSBAR_CELL_G6
XI319 BL4 NET542 NET807 NET815 NET810 NET815 PRESET2 RESET CROSSBAR_CELL_G6
XI320 BL3 IN0 NET831 NET823 NET834 NET823 PRESET2 RESET CROSSBAR_CELL_G6
XI321 BL3 IN1 NET834 NET880 NET850 NET831 0 RESET CROSSBAR_CELL_G6
XI322 BL3 IN6 NET871 NET839 NET840 NET839 0 RESET CROSSBAR_CELL_G6
XI323 BL3 IN3 NETl7 NETl8 NET1617 NET847 PRESET1 RESET CROSSBAR_CELL_G6
XI324 BL3 IN4 NET863 NET855 NET866 NET855 0 RESET CROSSBAR_CELL_G6
XI325 BL3 IN5 NET866 NET840 NET864 NET863 0 RESET CROSSBAR_CELL_G6

```
XI326 BL3 IN7 ORIN31 ORIN32 OROUT3 NET871 0 RESET CROSSBAR_CELL_G6
XI327 BL3 IN2 NET847 NET879 NET880 NET879 0 RESET CROSSBAR_CELL_G6
XI403 BL5 NET545 NET919 NET887 NET888 NET887 0 RESET CROSSBAR_CELL_G6
XI402 BL5 NET536 NETl11 NETl12 NET939 NET895 0 RESET CROSSBAR_CELL_G6
XI401 BL5 NET532 NET911 NET903 NET914 NET903 0 RESET CROSSBAR_CELL_G6
XI400 BL5 NET547 NET914 NET888 NET912 NET911 PRESET1 RESET CROSSBAR_CELL_G6
XI399 BL5 NET543 ORIN51 ORIN52 OROUT5 NET919 0 RESET CROSSBAR_CELL_G6
XI398 BL5 NET538 NET895 NET927 NET928 NET927 0 RESET CROSSBAR_CELL_G6
XI397 BL7 NET538 NET975 NET935 NET936 NET935 0 RESET CROSSBAR_CELL_G6
XI396 BL7 NET543 ORIN71 ORIN72 OROUT7 NET943 PRESET1 RESET CROSSBAR_CELL_G6
XI405 BL5 NET542 NET1079 NET951 NET1082 NET951 PRESET2 RESET CROSSBAR_CELL_G6
XI395 BL7 NET547 NET962 NET984 NET960 NET959 0 RESET CROSSBAR_CELL_G6
XI394 BL7 NET532 NET959 NET967 NET962 NET967 0 RESET CROSSBAR_CELL_G6
XI393 BL7 NET536 NETl15 NETl16 NET1019 NET975 0 RESET CROSSBAR_CELL_G6
XI392 BL7 NET545 NET943 NET983 NET984 NET983 0 RESET CROSSBAR_CELL_G6
XI391 BL7 NET540 NET994 NET936 NET978 NET991 0 RESET CROSSBAR_CELL_G6
XI250 BL2 IN0 NET1007 NET999 NET1010 NET999 PRESET2 RESET CROSSBAR_CELL_G6
XI251 BL2 IN1 NET1010 NET1056 NET1026 NET1007 0 RESET CROSSBAR_CELL_G6
XI252 BL2 IN6 NET1047 NET1015 NET1016 NET1015 0 RESET CROSSBAR_CELL_G6
XI253 BL2 IN3 NETl5 NETl6 NET1067 NET1023 0 RESET CROSSBAR_CELL_G6
XI254 BL2 IN4 NET1039 NET1031 NET1042 NET1031 0 RESET CROSSBAR_CELL_G6
XI255 BL2 IN5 NET1042 NET1016 NET1040 NET1039 0 RESET CROSSBAR_CELL_G6
XI256 BL2 IN7 ORIN21 ORIN22 OROUT2 NET1047 0 RESET CROSSBAR_CELL_G6
XI257 BL2 IN2 NET1023 NET1055 NET1056 NET1055 PRESET1 RESET CROSSBAR_CELL_G6
XI10 BL0 IN2 NET1103 NET1063 NET1064 NET1063 0 RESET CROSSBAR_CELL_G6
XI12 BL0 IN7 ORIN01 ORIN02 OROUT0 NET1071 0 RESET CROSSBAR_CELL_G6
XI404 BL5 NET540 NET1082 NET928 NET898 NET1079 0 RESET CROSSBAR_CELL_G6
XI13 BL0 IN5 NET1090 NET1112 NET1088 NET1087 0 RESET CROSSBAR_CELL_G6
XI14 BL0 IN4 NET1087 NET1095 NET1090 NET1095 0 RESET CROSSBAR_CELL_G6
XI9 BL0 IN3 NETL1 NETL2 NET1147 NET1103 0 RESET CROSSBAR_CELL_G6
XI11 BL0 IN6 NET1071 NET1111 NET1112 NET1111 0 RESET CROSSBAR_CELL_G6
XI1 BL0 IN1 NET1122 NET1064 NET1106 NET1119 0 RESET CROSSBAR_CELL_G6
XI0 BL0 IN0 NET1119 NET1127 NET1122 NET1127 NET1132 RESET CROSSBAR_CELL_G6
* File name: copperstuff_we_ckt8_schematic.s.
* Subcircuit for cell: we_ckt8.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:14 1999.
* terminal mapping: A = A
*                   B = B
*                   C = C
*                   WE = WE
*                   out = out
* End of subcircuit definition.
* File name: copperstuff_inv_schematic.s.
* Subcircuit for cell: inv.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:14 1999.
* terminal mapping: in = in
*                   out = out
* End of subcircuit definition.
* File name: copperstuff_inv_big1_schematic.s.
* Subcircuit for cell: inv_big1.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:14 1999.
```

```
* terminal mapping: in = in
*                  out = out
* End of subcircuit definition.
* File name: copperstuff_2nand_schematic.s.
* Subcircuit for cell: 2nand.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:13 1999.
* terminal mapping: ina = ina
*                  inb = inb
*                  out = out
* End of subcircuit definition.
* File name: copperstuff_2or_schematic.s.
* Subcircuit for cell: 2or.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:13 1999.
* terminal mapping: ina = ina
*                  inb = inb
*                  out = out
* End of subcircuit definition.
* File name: copperstuff_crossbar_cell_schematic.s.
* Subcircuit for cell: crossbar_cell.
* Generated for: hspiceS.
* Generated on Nov 22 10:45:14 1999.
* terminal mapping: bitline = bitline
*                  input = input
*                  orin1 = orin1
*                  orin2 = orin2
*                  orout = orout
*                  output = output
*                  preset = preset
*                  reset = reset
* End of subcircuit definition.
.lib "/afs/eos/project/erl/paulf_group/cu_contest/models/umc18P.m" L18U15V_TT
.lib "/afs/eos/project/erl/paulf_group/cu_contest/models/umc18N.m" L18U15V_TT
* Include files
* End of Netlist

.SUBCKT WE_CKT8_G5 A B C WE OUT
XI2 NET7 NET11 OUT OR_G1
XI1 C WE NET11 NAND_G2
XI0 A B NET7 NAND_G2
.ENDS WE_CKT8_G5

.SUBCKT INV_G4 IN OUT
M18 OUT IN VDD! VDD!  UMC18P  L=180E-9 W=1040E-9 AD=468E-15 AS=468E-15
+PD=2.98E-6 PS=2.98E-6 M=1
M17 OUT IN 0 0  UMC18N  L=180E-9 W=440E-9 AD=198E-15 AS=198E-15 PD=1.78E-6
+PS=1.78E-6 M=1
.ENDS INV_G4

.SUBCKT INV_BIG1_G3 IN OUT
M18 OUT IN VDD! VDD!  UMC18P  L=180E-9 W=2.08E-6 AD=936E-15 AS=936E-15
+PD=5.06E-6 PS=5.06E-6 M=1
M17 OUT IN 0 0  UMC18N  L=180E-9 W=880E-9 AD=396E-15 AS=396E-15 PD=2.66E-6
```

```
+PS=2.66E-6 M=1
.ENDS INV_BIG1_G3

*NAND gate near-minimum to reduce input line capacitance

.SUBCKT NAND_G2 INA INB OUT
M16 OUT INB VDD! VDD!  UMC18P  L=180E-9 W=360E-9 AD=162E-15 AS=162E-15
+PD=1.62E-6 PS=1.62E-6 M=1
M18 OUT INA VDD! VDD!  UMC18P  L=180E-9 W=360E-9 AD=162E-15 AS=162E-15
+PD=1.62E-6 PS=1.62E-6 M=1
M17 OUT INA NET38 0  UMC18N  L=180E-9 W=240E-9 AD=108E-15 AS=108E-15
+PD=1.38E-6 PS=1.38E-6 M=1
M19 NET38 INB 0 0  UMC18N  L=180E-9 W=240E-9 AD=108E-15 AS=108E-15 PD=1.38E-6
+PS=1.38E-6 M=1
.ENDS NAND_G2

*OR gate, larger than minimum to handle long M3 lines, balance NMOS/PMOS

.SUBCKT OR_G1 INA INB OUT
M16 NET17 INB VDD! VDD!  UMC18P  L=180E-9 W=2980E-9 AD=1341E-15 AS=1341E-15
+PD=6.86E-6 PS=6.86E-6 M=1
M18 OUT INA NET17 VDD!  UMC18P  L=180E-9 W=2980E-9 AD=1341E-15 AS=1341E-15
+PD=6.86E-6 PS=6.86E-6 M=1
M17 OUT INA 0 0  UMC18N  L=180E-9 W=880E-9 AD=396E-15 AS=396E-15 PD=2.66E-6
+PS=2.66E-6 M=1
M19 OUT INB 0 0  UMC18N  L=180E-9 W=880E-9 AD=396E-15 AS=396E-15 PD=2.66E-6
+PS=2.66E-6 M=1
.ENDS OR_G1


.SUBCKT CROSSBAR_CELL_G6 BITLINE INPUT ORIN1 ORIN2 OROUT OUTPUT PRESET RESET
XI29 NET30 OROUT INV_BIG1_G3
XI18 ORIN1 ORIN2 NET30 OR_G1
XI14 NET34 INPUT NET41 NAND_G2
XI20 PRESET NET81 INV_G4
XI16 NET47 NET36 INV_G4
XI17 NET41 OUTPUT INV_G4
XI15 NET36 NET34 INV_G4
M7 NET47 NET81 VDD! VDD!  UMC18P  L=180E-9 W=1040E-9 AD=468E-15 AS=468E-15
+PD=2.98E-6 PS=2.98E-6 M=1
M4 NET47 BITLINE NET34 VDD!  UMC18P  L=180E-9 W=1040E-9 AD=468E-15 AS=468E-15
+PD=2.98E-6 PS=2.98E-6 M=1
M1 NET47 RESET 0 0  UMC18N  L=180E-9 W=440E-9 AD=198E-15 AS=198E-15 PD=1.78E-6
+PS=1.78E-6 M=1
M0 NET47 BITLINE INPUT 0  UMC18N  L=180E-9 W=440E-9 AD=198E-15 AS=198E-15
+PD=1.78E-6 PS=1.78E-6 M=1
.ENDS CROSSBAR_CELL_G6
.TEMP    27.0000
.END
```