Nanowire-Based Sublithographic Programmable Logic Arrays

André DeHon
Dept. of CS, 256-80
California Institute of Technology
Pasadena, CA 91125
andre@acm.org

Michael J. Wilson
Dept. of CS, 256-80
California Institute of Technology
Pasadena, CA 91125
mwilson@caltech.edu

ABSTRACT

How can Programmable Logic Arrays (PLAs) be built without relying on lithography to pattern their smallest features? In this paper, we detail designs which exploit emerging, bottom-up material synthesis techniques to build PLAs using molecular-scale nanowires. Our new designs accommodate technologies where the only post-fabrication programmable element is a non-restoring diode. We introduce stochastic techniques which allow us to restore the diode logic at the nanoscale so that it can be cascaded and interconnected for general logic evaluation. Under conservative assumptions using 10nm nanowires and 90nm lithographic support, we project yielded logic density around 500,000nm^2/OR term for a 60 OR-term array; a complete 60-term, two-level PLA is roughly the same size as a single 4-LUT logic block in 22nm lithography. Each OR term is comparable in area to a 4-transistor hardwired gate at 22nm. Mapping sample datapaths and conventional programmable logic benchmarks, we estimate that each 60-OR-term PLA plane will provide equivalent logic to 5–10 4-input LUTs.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles—*logic arrays*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*advanced technologies*

General Terms

Design

Keywords

Sublithographic architecture, nanowires, programmable logic arrays

1. INTRODUCTION

Programmable logic arrays (PLAs) have been a child of lithography. Before lithographic integrated circuits, it did not make sense to build PLAs; logic was "customized" by discrete wiring (*e.g.* patch cables). Once lithography could support enough logic on a single chip to accommodate programmable configuration elements, it became interesting and useful to include memory elements which could configure the state of the device. As a result we developed PALs, PLDs, and ultimately FPGAs.

Now it is becoming possible to look beyond lithography and explore how we can build devices without relying on lithography to pattern our smallest feature sizes. Scientists are demonstrating bottom up techniques for defining key feature sizes. These techniques suggest a path to molecular-scale dimensions—they show us how to build features which are just a few atoms across. At this scale, programmable elements may be necessary in order to build any logic.

In this paper, we explore how to use nanowires (NWs) to build sublithographic PLAs and interconnected PLAs. These PLA blocks are likely to form leaf clusters in large-scale, sublithographic programmable logic devices just as small PLAs form leaf clusters in conventional, "complex" PLDs. Owing to the fabrication regularity demanded when using bottom-up techniques, the leaf clusters, and even the programmable interconnect, are likely to be engineered and parameterized quite differently from today's, lithographic-scale CPLDs. We identify those issues and explore where the technology limitations and costs drive our designs.

We build a two-plane PLA with decorated silicon NWs (See Figure 1). The NWs serve as our primary interconnect and device building block (Section 2.1). We build upon programmable crosspoint diodes (Section 2.2). We show that we can address individual NWs from the lithographic scale which is necessary for bootstrap programming (Section 2.3). We show how we achieve restoring logic and connect that to the programmable diode logic (Section 3), and we detail a precharge clocking scheme for logic evaluation (Section 4). We show how the basic PLA cycle can be embellished to allow PLA evaluation through a configurable number of logic planes (Section 5). We calculate the area, timing, and yield of these devices (Section 6) and sketch the discovery of the basic state of the devices to enable configuration and defect avoidance (Section 7). Finally, we map small, conventional programmable logic benchmarks and a select set of datapaths onto these devices to estimate their effective density compared to conventional alternatives (Section 8).

New contributions of this work include:

- explicit formulation of NW-based PLAs with separate programmable and restoring devices

FPGA'04, February 22-24, 2004, Monterey, California, USA.

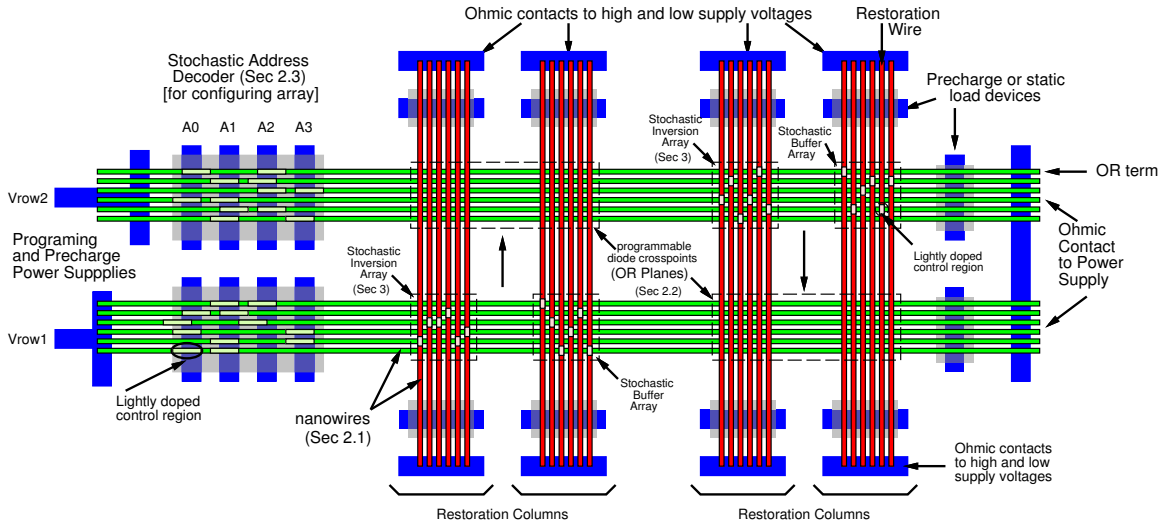


Figure 1: Simple Nanowire PLA Composition

- introduction of stochastic techniques for building fixed, restoring logic
- introduction of a PLA topology which requires lithographic programming lines in only one dimension and allows programming to be shared across multiple arrays
- introduction of a clocked, precharge scheme for sublithographic PLA logic
- introduction of simple topologies for physically configurable PLA cycles
- estimation of net logic density using mapped logic instances from both standard programmable logic benchmarks and focused datapath elements

2. BACKGROUND

2.1 Nanowires

Semiconducting nanowires (NWs) can be grown to controlled dimensions on the nanometer scale using seed catalysts (*e.g.* gold balls) to define their diameter. NWs with diameters down to 3nm have been demonstrated [9] [31]. By controlling the mix of elements in the environment during growth, semiconducting NWs can be doped to control their electrical properties [8]. Conduction through doped NWs can be controlled via an electrical field like Field-Effect Transistors (FETs) [21]. The doping profile along the length of a NW can be controlled by varying the dopant level in the growth environment over time [18]; as a result, our control over growth rate allows us to control the physical dimensions of these features down to almost atomic precision. The doping profile can also be controlled along the radius of these NWs, allowing us to sheath NWs in insulators (*e.g.* silicon-dioxide) [26] to control spacing between conductors [38] and between gated wires and control wires.

Flow techniques can be used to align a set of NWs into a single orientation, close pack them, and transfer them onto a surface [22] [38]. This step can be rotated and repeated so that we get multiple layers of NWs [22] [39] such as crossed NWs for building a crossbar array or memory core.

Each of the capabilities described above has been demonstrated in a lab setting as detailed in the papers cited. We assume it will be possible to combine these capabilities and scale them into a repeatable manufacturing process.

2.2 Programmable Diode Crosspoints

Over the past few years, many technologies have been demonstrated for molecular-scale memories. So far, they all seem to have: (1) resistance which changes significantly between "on" and "off" states, (2) the ability to be made rectifying, and (3) the ability to turn the device "on" or "off" by applying a voltage differential across the junction. Rueckes *et. al.* demonstrate switched devices using suspended nanotubes to realize a bistable junction with an energy barrier between the two states [34]. In the "off" state the junction exhibits only small tunneling current (high resistance $\sim$GΩs); when the devices are in contact in the "on" state, there is little resistance ($\sim$100KΩ) between the tubes. UCLA and HP have demonstrated a number of molecules which exhibits hysteresis [5] [6]. HP has demonstrated an 8×8 crossbar made from [2]rotaxane molecules and observed that they could force an order of magnitude resistance difference between "on" and "off" state junctions [3].

2.3 Addressing Nanowires from Lithographic Scale Wires

The preceding technologies allow us to pack NWs at a tight pitch into crossbars with programmable crosspoints at their junctions. The pitch of the NWs can be much smaller than our lithographic patterning. We will be using the crosspoint programmability to configure logic functions into our nanoscale devices. In order to do this, we need a way to selectively place a defined voltage on a single row and column wire in order to set the state of the crosspoint.

By constructing NWs with doping profiles on their ends, we can give each NW an address (See left end of Figure 1). The dimensions of the address bit control regions can be set to the lithographic pitch so that a set of crossed, lithographic wires can be used to address a single NW. If we code up all the NWs along one dimension of an array with suitably different codes, we can get unique NW addressability and effectively implement a demultiplexer between a small number of lithographic wires and a large number of NWs. We cannot control exactly which NW codes appear in a single array or how they are aligned, but if we randomly select NWs from a sufficiently large code space, we will achieve uniqueness with very high probability (over

99% easily achievable). The addresses do not have to be entirely unique for this application; allowing a little redundancy will allow us to use a tighter code space. The basic stochastic addressing scheme is developed in detail in [14]. Calculations allowing redundancy are summarized in [15].

2.4 Technology Roundup

We can create wires which are nanometers in diameter and can be arranged into crossbar arrays with nanometer pitch. Crosspoints which both switch conduction between the crossed wires and store their own state can be placed at every wire crossing without increasing the pitch of the crossbar array. NWs can be controlled in FET-like manner and can be designed with selectively gateable regions. The NWs can be individually addressed from the lithographic scale. No lithography is required to define the nanoscale features in the crossbar; lithography is used to define the extents of the crossbar, provide addressing for bootstrap programming, and provide voltage supplies for the NW array.

2.5 Related Work

Several researchers have begun to explore programmable logic structures at this scale. Heath [20] articulates a vision for this kind of molecular-scale logic. Goldstein's nanoFabrics [16] are a possible implementation of this vision using only the two-terminal diode crosspoints introduced above. Tour's nanoCell [37] employs a random network of configurable Negative Differential Resistance (NDR), avoiding the need for carefully ordered conductors on the nanoscale but still requiring lithographic scale wires to interconnect and restore nanoCells. Niemier [32] suggests an FPGA based on Quantum Cellular Automata (QCA) which may require lithographic-scale wires to control clock domains. NWs give us unique capabilities currently not exploited in other efforts. The techniques introduced here elaborate and complement the parallel research efforts, demonstrating what we can build with NWs, NW field-effect gating, doping profiles on NWs, and stochastic population.

3. RESTORATION

Programmable diode crosspoints in a crossbar array give us a programmable OR plane (See marked regions in Figure 1). Each output NW in the plane can be programmed to perform the OR of its set of inputs. That is, there is a low resistance path between the inputs and the output NW only where the crosspoints are programmed into the "on" position. If any of those inputs are high, they will be able to deliver current through the "on" crosspoint and pull the output line to a high value.

3.1 Limits of Diode Logic

Diodes alone do not give us arbitrary or cascadable logic. We know that OR gates are not universal logic building blocks. With diodes alone we cannot invert signals which will be necessary to realize arbitrary logic. Further, whenever an input is used by multiple outputs, we divide the current among the outputs; this cannot continue through arbitrary stages as it will eventually not be possible to distinguish the divided current from the leakage current of an "off" crosspoint. The diode junction may further provide a voltage drop at every crosspoint such that the maximum output high voltage drops at every stage.

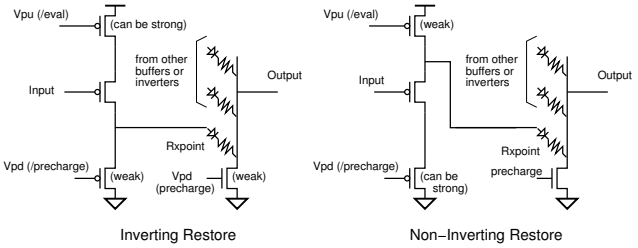


Figure 2: Semi-Static Stages for Inversion or Buffering

3.2 Basic Programmable Logic Stages

We overcome the limitations of diode logic by inserting rectifying field-effect stages between diode stages (Restoration Columns in Figure 1) [13]. As noted above, a doped NW can be gated like a FET. If the input field allows conduction, the NW will allow a source voltage to flow through the gated junction; otherwise conduction is cut off and the output is isolated, through a high impedance junction, from the supply.

When we place a field-effect buffer or inverter on the output of a diode OR NW, the entire OR stage is capacitively loaded rather than resistively loaded. The OR stage simply needs to charge up its output which provides the field for the field-effect based restoration stage. When the field is high enough (low enough for P-type NWs) to enable conduction in the field-effect stage, the NW will allow the source voltage to drive its output. The field-effect stage provides isolation as there is no current flow between the diode stage and the field-effect stage output. We have previously shown that NW field effects have good enough thresholds that we can get gain and logic level restoration with these stages [13].

Figure 2 shows a restoring buffer and restoring inverter logic stage built from NW field-effect gating and diode crosspoints. The restoring NWs are P-type and the receiving OR terms are N-type to define diode directionality for the crosspoints. We define the canonical stage to be a restoring gate followed by a diode plane because that corresponds to the total current path. The diode plane's output only drives the next such stage capacitively

The inverter case is simply a static load inverter followed by the programmable diode. If the input to the inverter is high, it depletes carriers in the depletion-mode P-type NW and cuts off conduction. As a result, the output of the inverter stage is connected only to the weak pulldown resistance and the output is held low. As long as this output remains low, the succeeding diode plane cannot be pulled high by this line. When the input to the inverter is low, there is current flow through the gate and the line is pulled high; this couples through any "on" diode points and pulls the associated diode output lines high. The strong pullup is ratioed appropriately with the weak pulldown, both in the inverter stage and in the diode stage, so that the pullup can drive the outputs to suitably high output voltages.

The buffer case exhibits the same idea in reverse. By taking the output from the high-voltage side of the input gate, the buffer output is coupled to a high voltage when the input is high and coupled to a low voltage when the input is low. In this case, the high voltage needs a weak pullup and the low voltage needs the strong drive for ratioed logic. Since the weak pullup must drive through the diode and pull up the diode outputs, it needs to be strong relative to

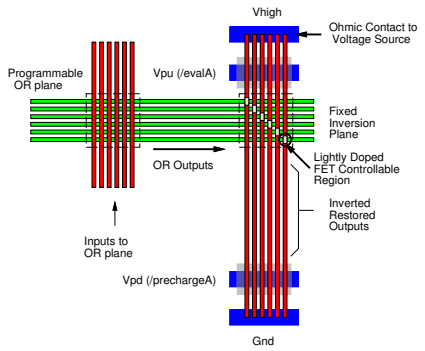


Figure 3: Ideal OR-Invert Array Pair

the diode pull down resistor. Either we can make the diode pull down resistance very weak so that the "weak" pullup is still strong enough to overpower it, or we simply provide a way to turn off the diode pulldown during logic evaluation so that it does not need to form a ratioed voltage divider with the weak pullup; the right side of Figure 2 shows this control input as a precharge signal.

3.3 Stochastic Construction

An ideal restoration stage would be an array of NWs where each NW restored a different one of the OR terms which crossed it (See Figure 3). To create the restoring field-effect devices in Figure 2, we use our ability to define the doping profile along the length of the NW to create a NW that has only a single controllable region. This single region is made wide enough for a single input, and only this single region is doped lightly enough to be gateable by its crossed input. The rest of the NW is heavily doped so that it is oblivious to its input.

We cannot precisely select and place restoration wires into a plane, but we can still define a useful restoration plane using our stochastic population technique. That is we code up batches of NWs with control regions in the appropriate places for each of the input locations. We mix these together and randomly select the NWs which go into each of the restoration arrays. This gives us a random selection of code wires. Given that we have a code space, N_{inputs}, (the number of input positions for diode lines into this array) and we populate our restoration array with $N_{restore}$ wires, how many of the input lines will we restore? Table 1 summarizes some $N_{restore}$, N_{inputs} relations; see [15] for calculation details. For example, Table 1 says that if we have 100 input lines and randomly select 100 restoring lines, we should expect to provide restoration for 56 different input lines.

3.4 Selective Inversion

In traditional PLA structures and PLA optimization, we are accustomed to having both the true and complement of a logic signal available. We can design this capability into our sublithographic PLA by dividing the "restoration plane" into two pieces and making half of the restoration plane inverting and half non-inverting. As shown in Figures 2 and 4 this means we simply need to setup the supply and control voltages on the restoring and non-inverting wires appropriately. The programmable diode OR stage which follows the restoration stage will take all the restored lines from both the buffered and inverted lines as potential inputs. We simply need to program this OR plane to select its inputs appropriately based on the needed polarity.

N_{inputs}	$N_{restore}$								
	20	30	40	50	60	70	80	90	100
20	10	12	14	16	17	17	18	18	19
40	12	17	21	24	27	29	30	32	33
60	14	19	24	29	33	36	39	41	43
80	15	21	27	32	36	41	44	48	51
100	15	22	28	34	39	44	48	52	56
120	16	23	29	36	41	47	52	56	60
140	16	23	30	37	43	49	54	59	64
160	16	24	31	38	44	50	56	62	67
180	16	24	32	39	45	52	58	64	69
200	17	25	32	39	46	53	59	65	71

Each table entry gives the number of different codes (different inputs restored) we can expect with 99% certainty for the N_{inputs}, $N_{restore}$ pair.

Table 1: Wires Restored with 99% Probability

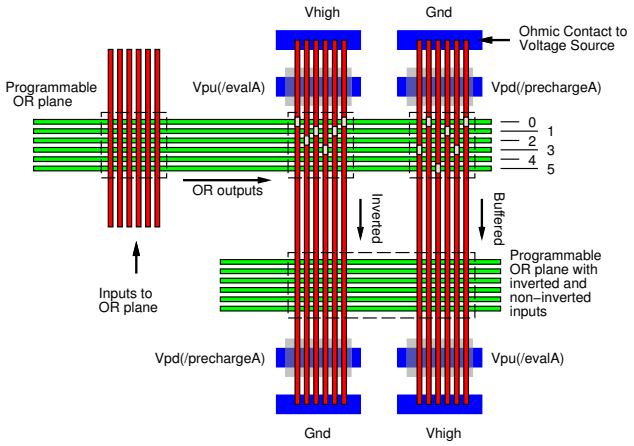


Figure 4: Selective Inversion from a Pair of Stochastic Restoring Stages

The population of both planes will be stochastic. This means we will get a distribution of inverting and buffering lines (See Figure 4). Some inputs from the previous stage will be buffered only (*e.g.* 5 in Figure 4), some will be inverted only (*e.g.* 2 in Figure 4), some will be both buffered and inverted (*e.g.* 0, 1, and 3 in Figure 4), and some will be neither buffered nor inverted (*e.g.* 4 in Figure 4). We must assign logic which we want inverted/buffered or inverted and buffered to the appropriate lines when selecting output lines in the OR array which precedes the restoration plane.

4. LOGIC DISCIPLINE AND CLOCKING

With slight modification in how we drive the control signals on the identified logic stages, we can turn this into a clocked logic scheme. An immediate benefit is the ability to create a finite-state machine out of a single pair of PLA planes. A second benefit, which we can currently only realize for inverting restoration stages, is the ability to replace the ratioed logic with precharge logic evaluation.

4.1 Clocking

The basic PLA cycle shown in Figure 1 is simply two restoring stages back-to-back (See Figure 5). For our clocking scheme, we evaluate the two stages at altering times.

First, note that if we turn off all three of our control transistors in our restoring stages (restoring precharge and eval-

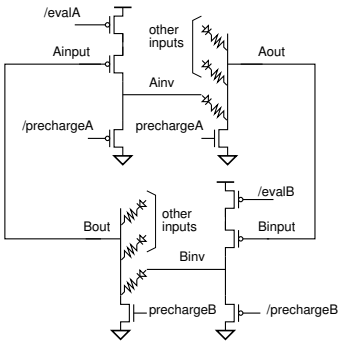


Figure 5: Precharge Clocked INV-OR-INV-OR (NAND-NAND, NOR-NOR, AND-OR) Cycle

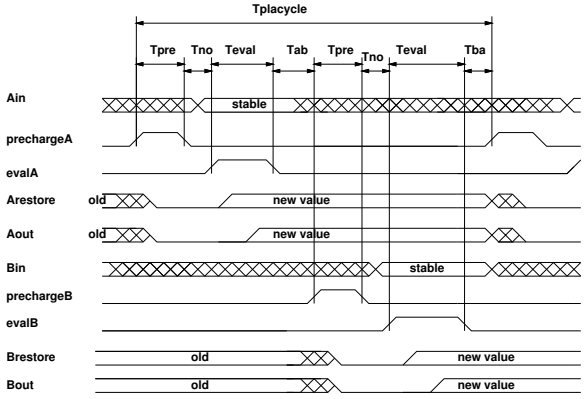


Figure 6: Clocking/Precharge Timing Diagram

uate and diode precharge), there is no current path from the input to the diode output stage. We effectively isolate the input from the output. Since the output stage is capacitively loaded, the output will hold its value. As with any dynamic scheme, eventually leakage on the output will be an issue, which will set a lower bound on the clock frequency. Owing to the small charges, this lower bound will be higher in NW logic than in conventional lithographic logic.

With a stage isolated and holding its output, we can evaluate the following stage. It computes its value from its input, the output of the previous stage, and produces its result by suitably charging its output line. Once this is done, we can isolate this stage and evaluate its succeeding stage, which, in this simple case, is also its predecessor.

In this manner, we never have an open current path all the way around the PLA (See Figure 5 and 6). In the two phases of operation, we effectively have a single register on any PLA outputs which feed back to PLA inputs.

4.2 Precharge Evaluation

For the inverting stage, we can dispense entirely with the weak pulldown transistor. Instead, we drive the pulldown gate hard during precharge and turn it off during evaluation. In this manner, we precharge the line low and pull it up only if the input is low. This works conveniently in this case because the output will also be precharged low. If the input is high, then we do not want to pullup the output and simply leave it low. If the input is low, we enable the current path to pullup the output. The net benefit is that inverter pulldown and pullup are both controlled by strongly driven gates and can be fast, whereas in the static logic scheme, the pulldown

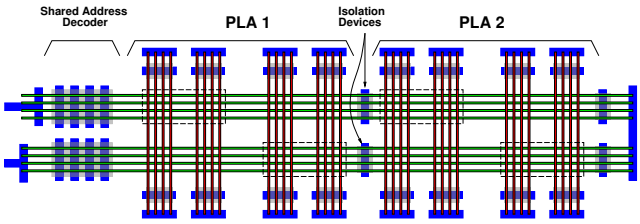


Figure 7: Address Decoder Shared Across PLAs

transistor had to be weak, making pulldown slow compared to pullup. Typically, the weak pulldown transistor would be set to have an order of magnitude higher resistance than the pullup transistor, so this can be a significant reduction in worst-case gate evaluate latency (See Section 6.4).

Unfortunately, we can neither precharge to high nor turn off the weak pullup resistor in the buffer case, so we do not get comparable benefits there. Perhaps new devices or circuit organizations will eventually allow us to build precharge buffer stages.

5. ORGANIZATION

The two-level PLA is adequate to implement logic in two-level sum-of-products form. Nonetheless, it is well known that many common functions require an exponential number of product terms when forced to two-level form, whereas they can be implemented in a linear number of gates (*e.g.* XOR). Research on optimal PLA block size to include in conventional, lithographic FPGAs suggests PLA blocks contain modest (*e.g.* 10) product terms and programmable interconnect [23]. However, the fact that we wish to amortize out the lithographic programming lines to get the benefits of sublithographic PLAs will likely shift the beneficial PLA size to larger numbers of product terms.

Here we explore how we can spread PLA evaluation over multiple planes to avoid unreasonable growth in product term requirements. We focus on two options for spreading out PLA evaluation and their interaction:

1. physically creating PLA cycles with s stages
2. looping the evaluation of some function w times through a set of s stages

5.1 Simple Cycle Organization

When we put together stochastic addressing, stochastic inversion, programmable diode crosspoints, and precharge clocking, we get a tight, two-plane array. This topology requires addressing overhead for programming only in one of the two dimensions (See left side of Figure 1).

A second feature of this array and clocking scheme is that the row addresses can be shared among multiple such PLA planes (See Figure 7). An isolation transistor serves to electrically separate the row segments of the planes during operation; during programming, the isolation transistor allows the row decoder to address all of the PLAs. The key feature which allows this sharing is that all rows on the same phase can be pulled down simultaneously during their associated precharge phase. We can use a single supply connection to precharge all of the rows low simultaneously, then isolate them for their evaluate and hold phases.

5.2 Physical Multistage Cycles

We can arrange the connection of NWs between arrays so that we can build larger cycles (See Figure 8). This is how

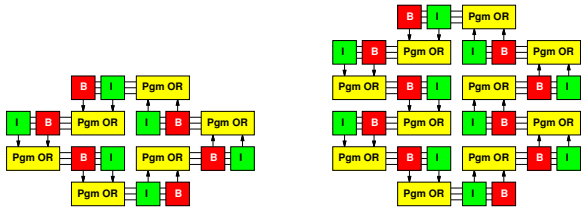


Figure 8: Length 6 and 10 PLA Plane Cycles

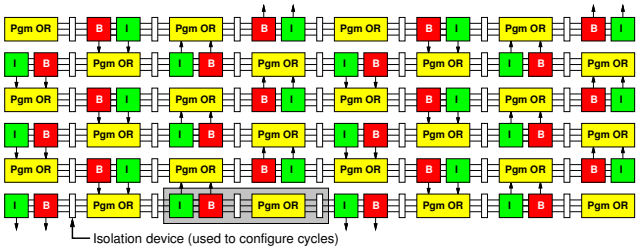


Figure 9: PLA Array for Variable Plane Cycles

we physically vary s. Observing that there is no directionality to the output of a diode NW plane, we can build a large array of these plane slices (Figure 9) and configure the isolation devices to logically arrange the large array into a series of cycles.

5.3 Wrapped Logical Stage Cycles

Rather than using a separate physical plane for every logical stage of evaluation in a spread PLA mapping, we can wrap the logic around the PLA multiple times.

Consider a 4-input XOR. XOR is known to require exponential product terms in two-level logic, while requiring only linear products if mapped in log(inputs) levels. We can wrap the XOR twice through the PLA, computing the 4-input XOR as a cascade of two levels of 2-input XORs (Figure 10). The wrapped logic requires 6 active OR terms for a total of 7 OR terms including inputs, outputs, and array feedbacks. The flat logic requires 8 OR terms. The difference grows as we go to larger XORs and deeper wrapping.

In a sense, the wrapped logic evaluation underutilizes the PLA by a factor of w since the inputs and outputs from each level of evaluation which are mapped to the same plane are independent. This is shown in cartoon form in Figure 11. The highlighted diagonal region is performing useful logic, while the off-diagonal portions of the OR planes are simply not being used. Nonetheless, for functions which require exponentially more logic when flattened to two levels, this is a net benefit. Further, since addressing and control present large fixed cost, it is beneficial to build larger arrays to amortize out that cost (See Section 6.3). Consequently, it may be more useful to wrap a computation into one pair of physical planes rather than to build a cycle of four physical planes.

6. PHYSICAL PROPERTIES

6.1 Array Area

As shown in Figure 1, each basic PLA unit has both lithographic scale support wires and NWs. For brevity we focus on the configurable cycle variant of Figure 9; this variant will have a single set of NW rows between precharge contacts rather than the pair of rows shown in Figure 1. Using:

- W_{litho} – minimum lithographic wire pitch

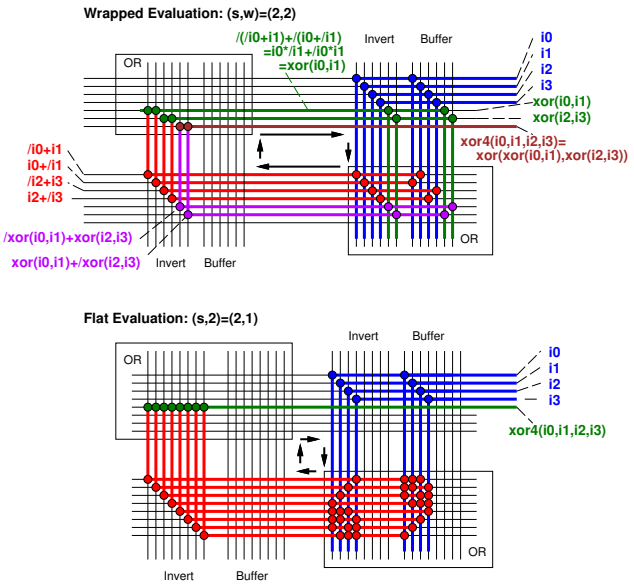


Figure 10: Demonstration of Wrapped Logic for Multi-Level Evaluation in a Single PLA Plane

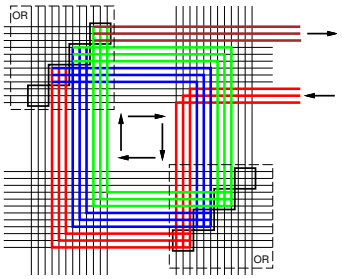


Figure 11: Wrapped Logic Cartoon (s,w)=(2,3)

- W_{nano} – pitch of NWs
- N_a – number of address bits
- N_{row} – number of raw row lines placed in array
- N_{buf} – number of raw buffer columns
- N_{inv} – number of raw inverting columns

We have:

$$L_{row} = (N_a + 9) \times W_{litho} + 2 \times (N_{buf} + N_{inv}) \times W_{nano} \quad (1)$$

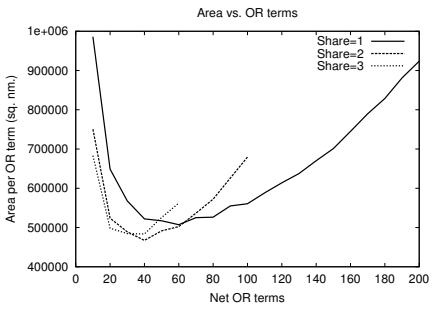
$$L_{column} = 6 \times W_{litho} + N_{row} \times W_{nano} \quad (2)$$

$$A_{plane} = L_{row} \times L_{column} \quad (3)$$

The 6 lithographic pitches in Equation 2 accounts for the 2 microscale wires above and below the OR NWs and a lithographic spacing between these microscale wires and the NWs. Similarly, the 9 extra lithographic scale pitches in Equation 1 account for lithographic wires and spacings at the ends of the array and between the distinct columns in the array.

6.2 Yield

To achieve a target PLA size, we will need to populate the array with a number of spare wires to accommodate broken wires and stochastic population effects. We use the different code calculation to compute the population needed to accommodate stochastic selection, and we use an M-of-N calculation to compute yield sparing [15].



W_{litho} = 210nm; W_{nano}=10nm; $W_{overlap}$=1nm; 50% binate, 25% invert, 25% buffer; 96% plane yield rate

Figure 12: Area per OR Term vs. Number of Lines

The whole yield process is best illustrated with an example. To yield an array with 60 OR terms where 30 of its plane-to-plane connections are binate (both inverting and non-inverting), 15 inverting only, and 15 buffered only, we populate the array with $N_{row} = 133$ and $N_{inv} = N_{buf} = 173$. With a wire and address alignment yield of 80%, our 133 row wires yield us 94 unbroken row wires over 99% of the time. The 80% yield is calculated based on the length of the wire and the probability of a bad address alignment [15]. We use $N_a = 14$ address bits which give us 3432 distinct 7-hot codes. Making 94 selections from 3432 codes gives us at least 90 uniquely coded row wires over 99% of the time. In the restoration stage, wires may be broken (wire yield 86% based on height with 133 row wires), wires may be uncontrollable because the control region overlaps multiple row wires (90% are controllable assuming $W_{overlap}$=1nm and W_{nano} = 10nm [15]), or wires may have their control regions aligned with unusable row wires (68%=90/133). Together, this means the yield rate of a restoring wire is 53%. Consequently, our 173 inverting or buffering inputs provide us with 75 good wires 99% of the time. Selecting 75 wires from the 90 good inputs, we expect 45 or more of them to be unique 99% of the time. With 45 useful buffering and inverting lines, we can use 30 of each for the binate inputs and the remaining 15 of each for the buffered only or inverted only inputs. At four steps in this process we said we would get a number of good or distinct wires 99% of the time. That means we yield this complete assembly 96% of the time ($0.99^4 > 0.96$). In practice, we can run the calculations described above in reverse in order to determine the N_{row}, N_{inv}, N_{buf} required to achieve a target array shape and to determine the best number of address bits (N_a) or total row wires (here we used 90 instead of the 60 required) to minimize area.

6.3 Net Area

Continuing the example, with W_{litho} = 210nm (90nm process [1]) and W_{nano} = 10nm, the plane will be 11.7μm wide and 2.6μm tall. 4.8μm of the width is in lithographic scale wires, as is 1.3μm of the height. Of the 133 NW rows, only 60 will be used for net logic. Of the $692 = 173 \times 4$ NW columns, only $180 = 45 \times 4$ provide useful restoration. Clearly, most of the area of the array is going into lithographic and yield overhead. The total area is $11.7\mu\text{m} \times 2.6\mu\text{m} \approx 3 \times 10^7\text{nm}^2$; the plane supports 60 OR terms, making each OR term about 500,000nm^2.

Figure 12 plots area per OR-term versus number of terms supported for a number of array sizes. In each case, we calculate appropriate sparing as illustrated in the previous section then calculate the area of that plane and divide by the number of supported OR terms to get an area per OR term. The three curves represent different levels of row address sharing. Here we see there is a definite minimum OR-term area around 60 OR-term arrays when we do not share the address and around 40 when we share an address unit across two arrays. The decreasing area per OR-term from 10 to 40 or 60 arises from amortizing out the fixed overhead for the microscale address and control wires. The increasing area after the minimum comes from decreasing yield probability for longer NWs and the increased number of restoring wires needed to support the outputs.

half pitch	W_{litho}	OR-term area (nm^2)	Address Share	OR-terms for min. array
90nm	210nm	470,000	2	40
65nm	150nm	330,000	2	20
45nm	105nm	240,000	2	20
22nm	50nm	140,000	2	20

W_{nano} = 10nm; $W_{overlap}$=1nm; 50% binate, 25% invert, 25% buffer; 96% plane yield rate

Table 2: Achievable OR-Term Area as a Function of Supporting Lithography

The lithographic overhead wires have a big effect on density both from the area they consume and because the NWs must span across them in order to yield. If we use smaller lithography for the programming scaffolding, the minimum achievable area will decrease further as shown in Table 2.

The prediction for 4-transistor gate area at the 22nm node is 300,000-500,000nm^2 [1]. A 4-LUT logic block in modest size arrays runs around 500,000 to 1,000,000λ^2 [11]; with $\lambda \approx$ 11nm for the 22nm node, this means a 4-LUT logic block will be 50–120,000,000nm^2. We note the areas of the 22nm 4-LUT and the 4-transistor gate for calibration only. Section 8 quantifies typical, mapped logic densities.

6.4 Timing

The time for a single plane evaluation (See Figure 6) is:

$$T_{plane} = T_{precharge} + T_{no} + T_{eval} + T_{ab} \quad (4)$$

Precharge just needs to discharge row and column capacitances through a contact (resistance R_c) and an "on" field-effect NW junction (resistance R_{on}):

$$C_{wire} = \max\left(C_{rowwire}, C_{colwire}\right) \quad (5)$$

$$T_{precharge} = \left(R_c + R_{on}\right) C_{wire} \quad (6)$$

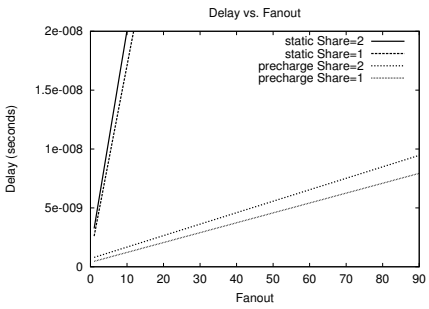
Evaluation in the precharge scheme, charges a number of rows through "on" diode junctions (resistance R_{mem-on}):

$$T_{eval} = \left(R_c + 2R_{on}\right)\left(C_{colwire} + f \times C_{rowwire}\right) + R_{mem-on} \times C_{rowwire} \quad (7)$$

f here is the fanout—the number of rows which use the restored term as input. For the static scheme, we need to make the static pullup resistance weak (See Figure 2, Section 3.2) compared to the pulldown resistance through the diode and the contact:

$$R_{pu} \approx 10\left(R_c + 2R_{on} + R_{mem-on}\right) \quad (8)$$

It has a similar evaluation equation which is roughly an order



60-OR-term array; W_{litho} = 210nm; W_{nano}=10nm; $W_{overlap}$=1nm; $R_c = 100\text{K}\Omega$, $R_{mem-on} = 100\text{K}\Omega$

Figure 13: Delay vs. Fanout for Static and Precharge Restoration Disciplines

of magnitude slower due to the weak pullup resistance.

$$T_{eval} = (R_c + R_{pu})(C_{colwire} + f \times C_{rowwire}) + R_{mem-on} \times C_{rowwire} \quad (9)$$

We assume $T_{no} = T_{ab} = T_{precharge}$. Capacitances, resistances, and other technology parameters are discussed in [15].

Figure 13 plots the phase time versus fanout for a 60-OR-term plane assuming $R_c = 100\text{K}\Omega$ and $R_{mem-on} = 100\text{K}\Omega$. Fanout effects are definitely a dominant term such that speed is roughly proportional to fanout in all cases. The precharge scheme is an order of magnitude faster than the static scheme, achieving 1ns phase times for a fanout of 7. The precharge scheme is viable now if we force logic to not require selective buffering, perhaps by computing logic in dual rail form so we always have the true and complement of every signal in the array.

7. DISCOVERY

Since addressing and restoration is stochastic, we will need to discover the live addresses and their restoration polarity. We use the restoration columns (See Figure 1) to help us identify useful addresses. The gate-side supply (*e.g.* top set of lithographic wire contacts in Figure 4) can be driven to a high value, and we look for voltage on the opposite supply line (*e.g.* bottom set of lithographic wire contacts in Figure 4; these contacts are marked V_{high} and Gnd but will be controlled independently as described here during discovery). There will be current flow into the bottom supply only if the control associated with the P-type restoration wire can be driven to a zero. We start by driving all the row lines high using the row precharge path. We then apply a test address and drive the supply (V_{row}) low. If the address is present, only that line will now be strongly pulled low. If the associated row line can uniquely control one or more wires in the restoration plane, the selected wires will now see a low voltage on their field-effect control regions and enable conduction from the top supply to the bottom supply. By sensing the voltage change on the bottom supply, we can deduce the presence of a restored address. We sense the buffering and inverting column supplies separately so we will know whether the line is buffering, inverting, or binate.

We need no more than $O\left((N_{row})^2\right)$ unique addresses to achieve virtually unique row addressing [14], so the search will require at most $O\left((N_{row})^2\right)$ such probes. For the example in the previous section, we used a $N_a = 14$ bit address with 3432 distinct codes. We might thus need to probe 3432 addresses to find the 90 live row wires.

Once we know all the present addresses in an array and the restoration status associated with each address, we can assign logic to logical addresses within each plane based on the required restoration for the output. With logic assigned to live addresses in each row, we can now use the address of the producing and consuming row wires to select and program a single junction in a diode-programmable OR plane.

Design	60-OR-term size	(s, w) org.	2-in gates	4-in LUT +FF	augment 4-LUT +FF
add	3	(2,1)	46	6	3
	4	(2,4)	60	8	4
	8	(2,8)	125	16	8
alu181	2	(2,2)	155	76	8
enable	9	(2,1)	61+9f	13	9
counter	15	(2,2)	111+15f	25	15
multiply	3×3	(2,1)	62	21	15
register	8×1	(2,1)	78+8f	22	3
file	3×4	(2,1)	78+12f	19	6
shiftreg	60	(2,1)	0+60f	60	5
xor	6	(2,1)	27	2	2
	30	(2,5)	145	10	10

s is the number of planes used; w is the number of wraps. See Section 5 and Figures 10 and 11.

Table 3: Datapath Benchmarks

8. MAPPED LOGIC

To measure the capacity of our PLAs, we map a series of benchmark circuits and measure the number of 2-input gates and the number of 4-LUTs it requires to implement the same benchmark. We use three benchmark sets:

1. Select datapath elements
2. Small FSMs from the IWLS93 benchmark suite [28]
3. PLA Book examples [17]

For PLA mapping, we use `espresso` to compute 2-level implementations [2] [33]; for multi-level planes, we use `sis` [36], first optimizing the logic with `script.rugged`, then targeting a given logic depth using `reduce_depth`. We see this gives us interesting results, but we also expect we can get better mappings using a more optimized CAD flow. Notably, multi-level optimization [30], smarter pterm-covering [4], and design levelization and balancing [12] [11] [27] should all help us to pack logic into fewer of these PLA planes.

For the 2-input gates, we use `sis` to map to the minimal `sis` library (an inverter, a 2-input NAND, and a 2-input NOR); we optimize the logic with `script.rugged` and map for minimum area using `map -m 0`. We use UCLA's RASP Suite [7] to map to 4-LUTs.

Table 3 summarizes a set of datapath mappings to a pair of 60-OR-term planes along with the resources required to map the same datapaths to 2-input gates and 4-LUTs. For both our PLA and the 4-LUT designs, we include hand-mapping results when those are better than the CAD results. Since FPGAs add strategic logic for datapaths (*e.g.* dedicated carry logic so an adder bit fits into an augmented 4-LUT), the final column estimates the typical "LUT" count for datapath augmented 4-LUTs. We show registers separately for the 2-input gates (+f in the table), since registers are typically counted as a few (2–4) 2-input gates.

Design	FSM Ins	Outs	S	60-OR-term (s,w) org.	2-in gates	ratio PLA	4-in LUTs	ratio PLA
tbk	6	3	32	(8,1)	213	27	98	12
pma	8	8	24	(18,1)	208	12	83	5
s1	8	6	20	(10,1)	188	19	75	8
ex1	9	19	20	(2,1)	186	93	70	35
keyb	7	2	19	(10,1)	184	18	77	8
s420	19	2	18	1/2 (2,1)	67	67	25	25
s208	11	2	18	1/2 (2,1)	67	67	25	25
sse	7	7	16	(2,1)	111	56	40	20
kirkman	12	6	16	(6,1)	139	23	58	10
bbsse	7	7	16	(2,1)	111	56	40	20
cse	7	7	16	(10,1)	178	18	71	7
dk512	1	3	15	1/2 (2,1)	51	51	16	16
mark1	5	16	15	(4,1)	84	21	38	10
ex4	6	9	14	1/2 (2,1)	60	60	20	20
s386	7	7	13	(2,1)	109	54	39	20
bbara	4	2	10	1/2 (2,1)	46	46	17	17
opus	5	6	10	1/2 (2,1)	79	79	25	25
dk17	2	3	8	1/2 (2,1)	56	56	15	15
shiftreg	1	1	8	1/2 (2,1)	11	11	4	4
ex6	5	8	8	(2,1)	83	42	32	16
dk14	3	5	7	(2,1)	72	36	24	12
beecount	3	4	7	1/2 (2,1)	22	22	9	9
dk27	1	2	7	1/2 (2,1)	20	20	5	5
s27	4	1	6	1/2 (2,1)	23	23	8	8
bbtas	2	2	6	1/2 (2,1)	21	21	6	6
mc	3	5	4	1/2 (2,1)	26	26	7	7
lion	2	1	4	1/2 (2,1)	13	13	3	3
dk15	3	5	4	1/2 (2,1)	61	61	19	19
tav	4	4	4	1/2 (2,1)	27	27	9	9
geometric mean						33		11

Restoration: 30 binate, 15 invert, 15 buffer; S=states

Table 4: Mapped FSM Statistics for 60-sterm PLA

Table 4 summarizes the results of mapping all IWLS93 FSMs with fewer than 30 states. We mapped both one-hot (`nova`) and dense encodings (`nova` [35] and `jedi` [25] with default options) for each of the design targets (PLA, 2-input gates, 4-LUTs) and then selected the least area result for each target. Since some of the FSMs are very small, if the FSM fits in a PLA half the size of our target PLA, we normalized it to one of the PLA planes instead of two; we did not attempt to get any greater measurement granularity than this half PLA unit. The PLA benchmarks we were able to map achieved comparable densities to the FSMs shown in Table 4.

Table 4 shows that we fit about 10 4-LUTs worth of logic in each, 30M nm^2 60-OR-term plane for control logic, and Table 3 shows that our densest datapath mappings provide a little under 10 4-LUTs worth of logic for a pair of planes. At 60M nm^2, a pair of 60-OR-term planes is comparable in area to a single, 22nm 4-LUT (50–120M nm^2 from Section 6.3).

Table 3 suggests the densest logic mappings achieve a density of roughly one hardwired 2-input gate per OR term. The FSMs (Table 4) and 2-level implementations of the datapath elements (Table 3) indicate that we yield about half a 2-input gate per OR term. Each OR term takes similar area to a 22nm 2-input gate (Section 6.3).

9. FUTURE WORK

The designs reported here are an early effort to completely detail a sublithographic PLA scheme. It represents a concrete starting point to improve upon rather than a final answer about how to organize sublithographic computation or the density achievable. The technologies for this scale are in their infancy and new devices and techniques are rapidly becoming available. Each advance potentially impacts the way we want to architect these devices.

Our earlier sketch suggested general ways to achieve arbitrary interconnect using NWs for block-to-block interconnect [13], and we certainly intend to work out details for providing such general interconnect among sublithographic PLA cells such as the ones introduced here.

The relative speed numbers for the static and precharge clocking schemes motivate search for circuits and devices to enable non-inverting precharge. Further, the robustness and viability of precharge at this scale needs to be explored.

Faults in the logic during operation are likely at this scale and even in lithographic logic as it approaches this scale. We will need to both understand the expected fault rates and develop and exploit techniques to map logic which is robust to the expected fault rates.

10. SUMMARY

We have shown how we can use emerging, bottom up, fabrication technologies to build PLAs where the key feature sizes determining device density and operation are defined without using lithography. This suggests we may be able to reach down to feature sizes and densities of 10's of nanometers without requiring lithography at that scale. Based on the current lithographic support architectures, pitches, and yield estimates, with 90nm lithographic support, PLA arrays with 40-60 yielded OR terms provide the densest unmapped logic resources, requiring 500,000nm^2 per OR term. Mapped logic suggests we can place the equivalent functionality of 10–20 4-LUTs into the area of a single, 2-plane, 60-term PLA, which is comparable in size to a single 4-LUT in 22nm lithography. With the rapid advance of technology, it is unlikely the devices ultimately produced will look *exactly* like the ones described here, nonetheless, many of the techniques introduced here will likely be important components in future sublithographic programmable logic designs.

11. ACKNOWLEDGMENTS

This research was funded in part by the DARPA Moletronics program under grant ONR N00014-01-0651. This architectural work would not have been possible or meaningful without close cooperation and support from Charles Lieber. Thanks to Fan Mo for tips on using `sis`.

12. REFERENCES

[1] International technology roadmap for semiconductors. <http://public.itrs.net/Files/2001ITRS/>, 2001.

[2] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, 1984.

[3] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, 2003.

[4] D. Cheng, J. Cong, M. Ercegovac, and Z. Huang. Performance-Driven Mapping for CPLD Architectures. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 39–47, 2001.

[5] C. Collier, G. Mattersteig, E. Wong, Y. Luo, K. Beverly, J. Sampaio, F. Raymo, J. Stoddart, and J. Heath. A [2]Catenane-Based Solid State Reconfigurable Switch. *Science*, 289:1172–1175, 2000.

[6] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddard, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, 1999.

[7] J. Cong, E. Ding, Y.-Y. Hwang, J. Peck, C. Wu, and S. Xu. RASP_SYN release B 1.1: LUT-Based FPGA Technology Mapping Package. <http://cadlab.cs.ucla.edu/~xfpga/software/raspsyn.htm>, 1999.

[8] Y. Cui, X. Duan, J. Hu, and C. M. Lieber. Doping and electrical transport in silicon nanowires. *Journal of Physical Chemistry B*, 104(22):5213–5216, June 8 2000.

[9] Y. Cui, L. J. Lauhon, M. S. Gudiksen, J. Wang, and C. M. Lieber. Diameter-controlled synthesis of single crystal silicon nanowires. *Applied Physics Letters*, 78(15):2214–2216, 2001.

[10] Y. Cui, Z. Zhong, D. Wang, W. U. Wang, and C. M. Lieber. High Performance Silicon Nanowire Field Effect Transistors. *Nanoletters*, 3(2):149–152, 2003.

[11] A. DeHon. Reconfigurable Architectures for General-Purpose Computing. AI Technical Report 1586, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, October 1996.

[12] A. DeHon. DPGA Utilization and Application. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, February 1996.

[13] A. DeHon. Array-Based Architecture for FET-based, Nanoscale Electronics. *IEEE Transactions on Nanotechnology*, 2(1):23–32, March 2003.

[14] A. DeHon, P. Lincoln, and J. Savage. Stochastic Assembly of Sublithographic Nanoscale Interfaces. *IEEE Transactions on Nanotechnology*, 2(3):165–174, 2003.

[15] A. DeHon and M. J. Wilson. Nanowire-Based Sublithographic Programmable Logic Arrays [Extended Version with Detail Appendix]. URL: <http://www.cs.caltech.edu/research/ic/abstracts/nanopla_fpga2004.html>, January 2004.

[16] S. C. Goldstein and M. Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proceedings of the International Symposium on Computer Architecture*, pages 178–189, June 2001.

[17] U. C. Group. Espresso examples. Online <ftp://ic.eecs.berkeley.edu/pub/Espresso/espresso-book-examples.tar.gz>, June 1993.

[18] M. S. Gudiksen, L. J. Lauhon, J. Wang, D. C. Smith, and C. M. Lieber. Growth of nanowire superlattice structures for nanoscale photonics and electronics. *Nature*, 415:617–620, February 7 2002.

[19] M. S. Gudiksen, J. Wang, and C. M. Lieber. Synthetic control of the diameter and length of semiconductor nanowires. *Journal of Physical Chemistry B*, 105:4062–4064, 2001.

[20] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, June 12 1998.

[21] Y. Huang, X. Duan, Y. Cui, L. Lauhon, K. Kim, and C. M. Lieber. Logic gates and computation from assembled nanowire building blocks. *Science*, 294:1313–1317, 2001.

[22] Y. Huang, X. Duan, Q. Wei, and C. M. Lieber. Directed assembley of one-dimensional nanostructures into functional networks. *Science*, 291:630–633, January 26 2001.

[23] J. Kouloheris and A. E. Gamal. Pla-based fpga area versus cell granularity. In *Proceedings of the Custom Integrated Circuits Conference*, pages 4.3.1–4. IEEE, May 1992.

[24] C. M. Lieber. Nanowire contact resistance. *Personal Communications*, July 2003.

[25] B. Lin and R. Newton. Synthesis of multiple-level logic from symbolic high-level description languages. In *Proceedings of the IFIP International Conference on VLSI*, pages 187–196, 1989.

[26] M. S. G. Lincoln J. Lauhon, D. Wang, and C. M. Lieber. Epitaxial core-shell and core-multi-shell nanowire heterostructures. *Nature*, 420:57–61, 2002.

[27] H. Liu and D. F. Wong. Network flow based partitioning for time-multiplexed fpgas. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 497–504, 1998.

[28] K. McElvain. LGSynth93 Benchmark Set: Version 4.0. Online <http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth93/doc/iwls93.ps>, May 1993.

[29] P. L. McEuen, M. Fuhrer, and H. Park. Single-walled carbon nanotubes electronics. *IEEE Transactions on Nanotechnology*, 1(1):75–85, March 2002.

[30] F. Mo and R. K. Brayton. Whirlpool PLAs: A Regular Logic Structure and Their Synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 543–550, November 2002.

[31] A. M. Morales and C. M. Lieber. A laser ablation method for synthesis of crystalline semiconductor nanowires. *Science*, 279:208–211, 1998.

[32] M. T. Niemier, A. F. Rodrigues, and P. M. Kogge. A Potentially Implementable FPGA for Quantum Dot Cellular Automata. In *Proceedings of the First Workshop on Non-Silicon Computation (NSC-1)*, Boston, MA, February 2002.

[33] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 6(5):727–751, September 1987.

[34] T. Rueckes, K. Kim, E. Joselevich, G. Y. Tseng, C.-L. Cheung, and C. M. Lieber. Carbon nanotube based nonvolatile random access memory for molecular computing. *Science*, 289:94–97, 2000.

[35] T. V. A. Sangiovanni-Vincentelli. NOVA: state assignment of finite state machines for optimal two-level logic implementations. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 327–332, 1989.

[36] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. UCB/ERL M92/41, University of California, Berkeley, May 1992.

[37] J. M. Tour. *Molecular Electronics: Commercial Insights, Chemistry, Devices, Architecture and Programming*. World Scientific Publishing Company, New Jersey, 2003.

[38] D. Whang, S. Jin, and C. M. Lieber. Nanolithography using hierarchically assembled nanowire masks. *Nanoletters*, 3(7):951–954, July 9 2003.

[39] D. Whang, S. Jin, Y. Wu, and C. M. Lieber. Large-scale hierarchical organization of nanowire arrays for integrated nanosystems. *Nanoletters*, 3(9):1255–1259, September 2003.

APPENDIX

A. TECHNOLOGY ASSUMPTIONS

Table 5 summarizes some of the key technology parameters assumed for this analysis along with references.

Resistance For nanotubes, the fundamental limit on contact resistance is 6.5KΩ and it appears to be possible to approach this limit [29]; consequently, it should be possible to decrease contact resistance with further technology mastery.

Capacitance From [8] the capacitance of a NW junction is:

$$C_{junc} \approx 2\pi\epsilon_{SiO_2}\frac{L}{\ln(2h/r)}$$

Where r is the NW radius, h is the distance between conductors, and l is the length of the overlap. For the NW junctions, we assume: $r = 1\text{nm}$, $h = 1\text{nm}$, $L = 2r$.

$$\begin{aligned} C_{nanoj} &\approx 2\pi(3.4\times10^{-11}\text{F/m})\left(\frac{2\text{nm}}{\ln(2)}\right) \\ &\approx 6\times10^{-19}\text{F} \end{aligned} \quad (10)$$

We use $C_{nanoj} = 10^{-18}$F as a rough approximation here. For the NW runs over the microwires, $L = W_{litho}/2$ assuming half the pitch is in the wire and half in the spacing between wires. Here height is oxide thickness between the address lines and the nanoscale core memory wires which is likely to be 5–10nm (assume $h = 5\text{nm}$).

$$C_{microj} \approx 2\pi(3.4\times10^{-11}\text{F/m})\left(\frac{W_{litho}/2}{\ln(2\times5/1)}\right) \quad (11)$$

Assuming the dominant capacitance is junction capacitance between crossed NW junctions and NW over microwire junctions, we can calculate the nanoscale junction capacitance (C_{nanoj}) and nano-micro junction capacitance (C_{microj}) and estimate the total row and column wire capacitances:

$$\begin{aligned} C_{colwire} &= N_{row}\times C_{nanoj} + 2\times C_{microj} \\ C_{rowwire} &= N_{column}\times C_{nanoj} + (N_a+1)\times C_{microj} \end{aligned}$$

This assumes we have to charge up all the capacitance associated with the address lines. We can isolate the address lines during operation, so that we only pay for two microwire junctions in the row case as well.

Wire Yield For an N junction long NW to yield, it must make good contact on both ends and contain no junction failures along its length:

$$P_{goodwire} = (P_c)^2\times(P_j)^N \quad (12)$$

[19] reports reliable growth of SiNWs which are over 9μm long (*i.e.* no breaks over a distance equivalent to 900 10nm junction lengths). More recently, Lieber is seeing over 90% yield of NWs up to 20μm long [24], which would correspond to $P_j > 0.99995$.

B. STOCHASTIC SELECTION

Let C be the number of codes in our code space, N be the number of selection we make (number of wires chosen), and u be the number of distinct codes appearing in our selection. We can calculate the probability distribution for the number of different codes, u, in an array, using the recurrence relation:

$$\begin{aligned} &P_{different}(C,N,u) = \\ &\quad\left(\frac{C-(u-1)}{C}\right)\times P_{different}(C,N-1,u-1) \\ &\quad+\left(\frac{u}{C}\right)\times P_{different}(C,N-1,u) \end{aligned} \quad (13)$$

The base cases are:

- $P_{different}(C,1,1) = 1$ (if we pick one thing, we get one different thing)
- $P_{different}(C,1,u\neq1) = 0$ (if we pick one, we will get exactly one different thing)
- $P_{different}(C,0,0) = 1$ (if we pick nothing, we get nothing)
- $P_{different}(C,0,u>0) = 0$ (if we pick nothing, we get nothing)
- $P_{different}(C,N,u<0) = 0$ (we cannot have less than nothing)

This recurrence counts each code once even if it appears multiple times, which is what we want if we allow duplicate codes in the addressing or duplicate restoration.

Since we are generally interested in achieving at least a certain number of codes, we are interested in the cumulative distribution function (CDF) for the probability that we have at least a certain number of codes. We calculate this:

$$P_{at\_least}(C,M,u) = \sum_{i=u}^{M} P_{different}(C,M,i) \quad (14)$$

Table 1 is generated with $C = N_{inputs}$ and $M = N_{restore}$; we set a target for P_{at_least}, 99% in this case, and find the largest u for which P_{at_least} exceeds the target.

C. N-CHOOSE-M CALCULATION

A common calculation used for defect tolerant design is the probability that we yield at least M elements form a full set of N. Assuming each element yields independently with probability, P, the probability that we will yield $M \leq N$ things is:

$$P_{MofN} = \sum_{M\leq i\leq N}\left(\binom{N}{i}P^i(1-P)^{N-i}\right) \quad (15)$$

OR term yield is a common example where this calculation is relevant in our nanoscale PLA. Here, N is the number of NWs we assemble into the array, M is the number of NWs we yield, and P is the probability that the NW yields (*e.g.* is not broken or shorted; see Equation 12). P_{MofN} in the equation above, tells us the probability that we will yield the specified M NWs. In design, we typically pick a target value for P_{MofN}, such as 0.99, then look for the largest M for which the calculated P_{MofN} exceeds the target probability.

Symbol	Description	Value	Ref.
R_c	micro-to-nanoscale contact resistance	100KΩ 1MΩ	[10] [21]
R_{mem-on}	diode "on" resistance	100KΩ	[34]
$R_{mem-off}$	diode "off" resistance	>1GΩ	[34]
C_{nanoj}	Capacitance of a NW-NW junction	1×10^{-18}F	(text) [8]
P_c	Probability of a good micro-to-nanoscale contact	$> 95\%$	[21]
P_j	Probability of a good junction per nanopitch unit length	> 0.9999	[19]

Table 5: Key Technology Parameter Assumptions

D. TOLERATING MISALIGNMENT

We likely have little or no control over the alignment of the NWs relative to each other in the array. We tolerate this misalignment using three techniques:

1. repeat coding
2. control region size selection
3. multiple voltage control

Repeat Coding For the stochastic restoration wires, we code as controllable one control region within the height of the restoration plane. However, the wire will be longer than the height of the restoration plane as it also extends over the succeeding OR plane and the control ends. We code wires by repeating the code every restoration plane height. That way, there will always be a coded region within the height of the restoration plane. This may also place control regions within the OR plane or on the ends of the wire. Placing it outside of the NW array is tolerable as it does not interfere with operation. For some of the memory technologies we are considering, the memory function will isolate the control region from the crossed OR lines so that it will not interfere with operation in the OR plane.

Control Region Width To tolerate sub bit-pitch misalignments within the control plane, we make the control region be just over one NW pitch wide. Assuming a NW pitch of W_{nano} and a necessary control overlap $W_{overlap}$, we make the control region:

$$W_{ctrl} = W_{nano} + 2 \times W_{overlap} < 2 \times W_{nano} \tag{16}$$

This guarantees that the control region always overlaps at least one NW by a length $W_{overlap}$. It is possible, that the control region will overlap two adjacent NWs. When it overlaps two NWs, we can program both inputs the same so it is no different from a single connection.

The probability that a control region overlaps multiple input NWs is:

$$P_{ctrl2} = \frac{2 \times W_{overlap}}{W_{nano}} \tag{17}$$

When this does happen, we can use the two lines as one, so, for the sake of yield calculations, we only discount half the cases where this occurs, so:

$$P_{single\_nw\_ctrl} = 1 - \frac{P_{ctrl2}}{2} = 1 - \frac{W_{overlap}}{W_{nano}} \tag{18}$$

For Section 6, we assumed $W_{overlap} = 1$nm. This gives us $P_{single\_nw\_ctrl} = 0.90$ for $W_{nano} = 10$nm.

Multiple Voltages At the ends of every row (and column) NW, we use NW field-effect gating to control static voltages and precharge operations. The key observation here is that we can run these control voltages at a different level from the operational voltages in the array. In particular, we can pick them to be high enough to exceed the high (strongly doped) threshold in the NW. As a result, these control wires do not need to be aligned with a lightly doped control region. They will operate regardless of NW alignment.

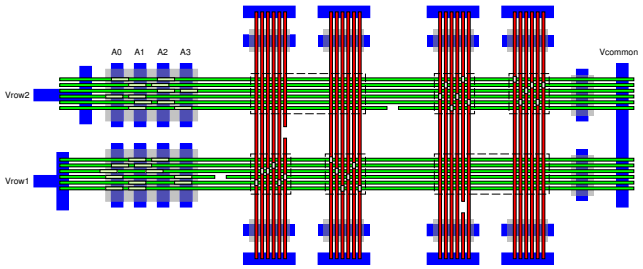


Figure 14: Illustration PLA with Broken NWs

E. DISCOVERY AND PROGRAMMING

E.1 Programming

To program any diode crosspoint in the OR planes (Figure 1), we drive one address into the top address decoder and the second into the bottom. Here the stochastic restoration performs the corner turn for us so that we effectively place the desired programming voltage differential across a single crosspoint. We set the voltages and control gating on the restoration columns to define which programmable diode array is actually programmed during a programming operation (*e.g.* in Figure 4 the contacts marked V_{high} and Gnd are the control voltages; V_{pu} and V_{pd} are the signals used for control gating).

E.2 Scaling

Note that each NW PLA array is addressed separately from its set of microscale wires ($A0$, $A1$, ... in Figure 1 and V_{row}, V_{bot}, and V_{top} in Figure 16, illustrated in Section *E.3.2*). Programming is not effected or complicated by the cascade structure of the programmed logic. Since the programming task is localized to each PLA plane, the work required to program a collection of planes (*e.g.* Figure 9) only scales linearly with the number of planes.

E.3 Example

To illustrate the discovery and programming process, this section walks through the steps involved in discovering and programing the illustration PLA in Figure 1 to perform a 2-input XOR. We will assume two of the wires are broken as shown in Figure 14 to better illustrate defect handling.

E.3.1 Discovering Present Addresses

First we must discover which addresses are present in each of the two planes. For this example, we have 4 address lines for addressing the OR-term NWs. We use a 2-hot code,

meaning there are $\binom{4}{2} = 6$ possible addresses for the OR terms in each plane (1100, 1010, 0110, 1001, 0101, 0011). So, for each plane, we need to test each of the 6 addresses.

To test for an address's presence, we:

1. Drive the right end common supply (V_{common} in Figure 14) to ground, then release it.
2. Drive the address lines ($A0$, $A1$, ...) to the test address.
3. Drive the common row line (*i.e.* V_{row1} or V_{row2} in Figure 14) to high.
4. Observe the voltage on the common line (V_{common}).

The common line will be raised to high *only if* the test address is present allowing a complete path between V_{row} and V_{common}. Figure 15A shows an attempt to read the address 1001 on the top plane. Since the 1001 wire is not present, this results in no current path from V_{row2} to V_{common} and V_{common} remains low. Figure 15B shows an attempt to read the address 1100 on the top plane. Since the 1100 wire is present and unbroken, this does succeed in raising the voltage on V_{common}. In Figure 15C, we see that an attempt to read address 0101 does not raise V_{common} since the wire has a break in it. After testing all six addresses, we know that the present and functional addresses in the top plane are 1100, 1010, 0110, and 0011. Similar testing for the bottom plane tells us the present and function addresses there are 1100, 1010, 0110, and 0101.

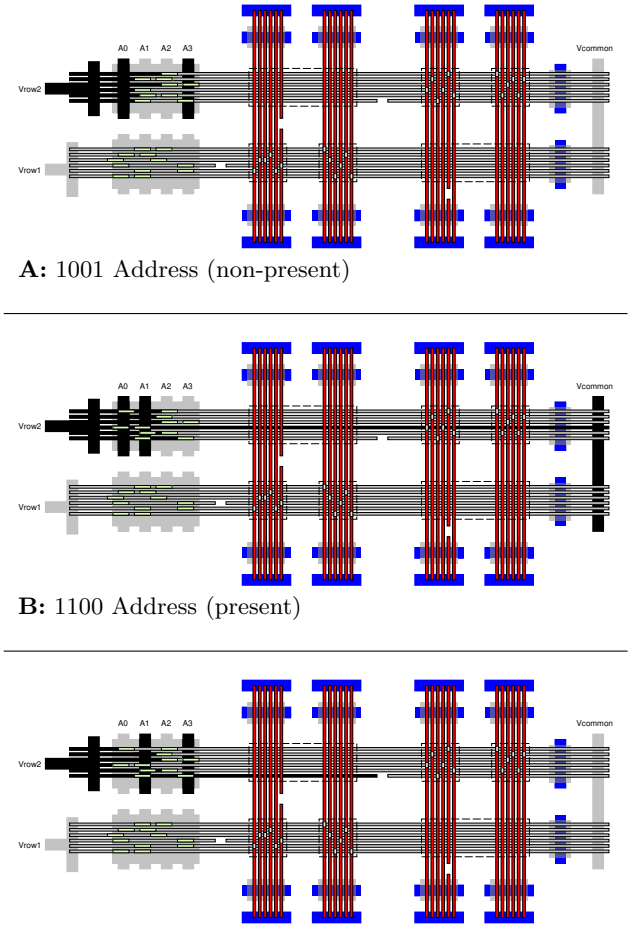
E.3.2 Discovering Polarities

Knowing which addresses are present, we can determine which polarities they provide. We drive each good address to a low voltage, while keeping the other wires high, and observe if the output is restored. For each address we:

1. Set the gate-side supplies on the restoration column (V_{top} in Figure 16) to a low voltage.
2. Drive the opposite supplies (V_{bot} in Figure 16) to a low voltage and release.
3. Use V_{common} and V_{row} to precharge alls lines to a high voltage. Drive the addresses ($A0$, $A1$,...) to all ones during this precharge. Note that we drive high from both sides; this way even wires with a single break are charged to a high voltage.
4. Release V_{common}, V_{row} and return the addresses to zeros.
5. Drive the intended address on the address lines.
6. Drive V_{row} to a low voltage.
7. After the row line has had time to discharge, drive the gate-side supplies on the appropriate restoration columns (V_{top} in Figure 16) to a high voltage.
8. Observe the voltage on the opposite supply (*e.g.* V_{bot}) once the restoration line has had a chance to charge.

Remember the restoration wires are P-type NWs. A high voltage across their lightly-doped control region will deplete carries and prevent conduction, while a low voltage will allow conduction. In steps 3–6, we guarantee that only the addressed row is low; all other rows are driven to a high value. As a result, we will only see conduction between V_{top} and V_{bot} in a column if the addressed NW controls some NW in that column.

Figure 16A shows how we test the 1100 wire. As described above, we drive only the 1100 wire to a low voltage. The restoration columns for this wire are bracketed by V_{top3}/V_{bot3} and V_{top4}/V_{bot4}, so we drive V_{top3} and V_{top4} to high voltages and watch the voltage on V_{bot3} and V_{bot4}. Notice that the 1100 NW intersects with two control regions in restoration column 3 and no control regions in restoration column 4. Consequently, V_{bot3} is pulled high while V_{bot4} remains low. By convention (See Figures 1 and 4), column 3 is our inverting column. This tells us the 1100 OR term can only be used in its inverting sense.



A: 1001 Address (non-present)

B: 1100 Address (present)

C: 0101 Address (defective NW)

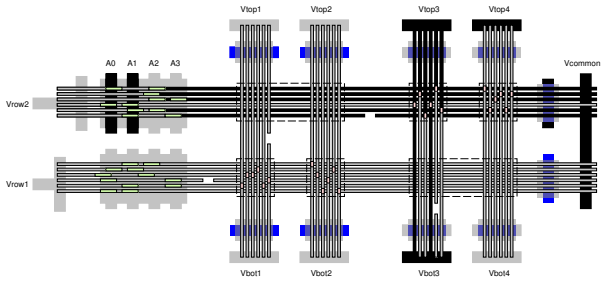
Light grey lines are at ground;
black lines driven to high voltage.

Figure 15: Testing Functional Address Presence

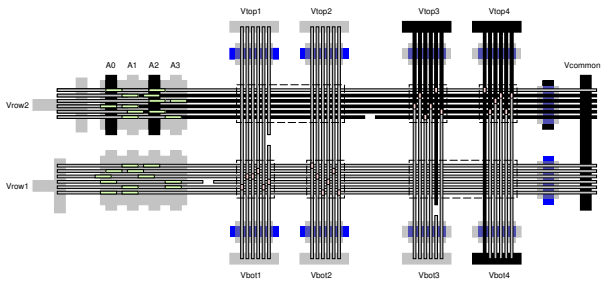
In Figure 16B we show testing of the 1010 wire. The 1010 wire controls restoration wires in both columns 3 and 4. However, the restoration wire in column 3 is broken. Consequently only the restoration in column 4 is usable. V_{bot4} is pulled high, but V_{bot3} remains low because of the broken column 3 wire. We conclude that we can use the 1010 OR term only in its non-inverting sense.

Figure 16C shows the result of testing the 0110 wire. There are two 0110 wires in the top plane. Consequently, when we address 0110 we affect both of them. Since the two wires have the same address, we will always select them in tandem. Both OR terms are charged low. It further turns out that there are multiple wires in both column 3 and column 4 controlled by either of the two 0110 OR terms. Both V_{bot3} and V_{bot4} are driven high telling us that we can use both polarities of the 0110 OR-term (*i.e.* it is binate). Testing the 0011 OR-term in a similar manner, we learn that it can only be used in the non-inverted sense.

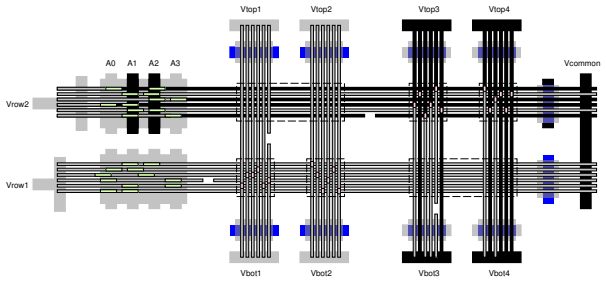
We can perform similar test for the lower OR plane. In this case, the outputs of this OR plane are restored by columns 1



A: 1100 NW (inverting)



B: 1010 NW (non-inverting)



C: 0110 NW (binate)

Figure 16: Discovering Restoration Polarities

and 2. We drive the high test values into V_{bot1} and V_{bot2} and observe the voltages at V_{top1} and V_{top2}; the role of top and bottom supplies are reversed compared to the top OR plane to match the fact that the position of the restoration array and the succeeding OR array are reversed. After performing the test, we learn the that 1100 and 1010 are binate, 0110 is non-inverting, and 0101 is inverting.

E.3.3 Programming Diode Crosspoints

Knowing which polarities are available from each of the present addresses, we can program up the intended function.

Figure 17 shows our assignment of known, good OR terms to the XOR calculation. We bring in the inputs A and B on the bottom OR terms 1100 and 1010. We need both polarities of A and B, and we know both of these terms are binate. We compute $\overline{A} + B$ on the top OR term 1100 knowing it is inverting, and we compute $A + \overline{B}$ on the top OR term 0110 which is binate so can provide an inverted output. Finally, we use bottom OR term 0110 to OR together $\overline{\overline{A} + B}$ and $\overline{A + \overline{B}}$ to produce the XOR of A and B.

To program up each crosspoint, we must apply suitable voltages to both the wires in the junction. For example, to make the restored B an input to the $\overline{A} + B$ in the top plane,

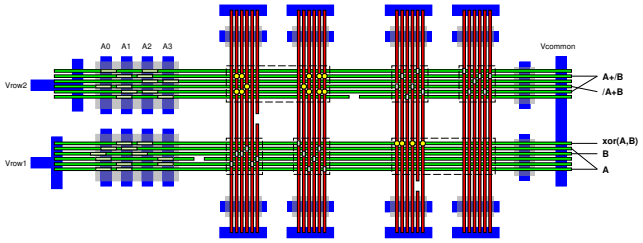
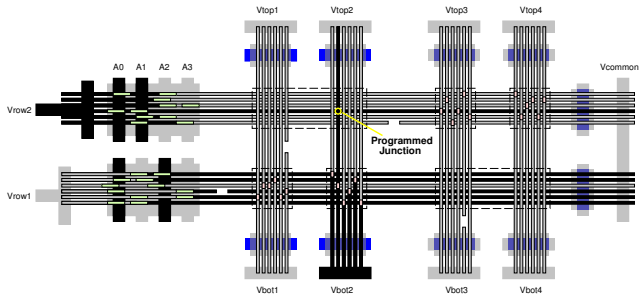
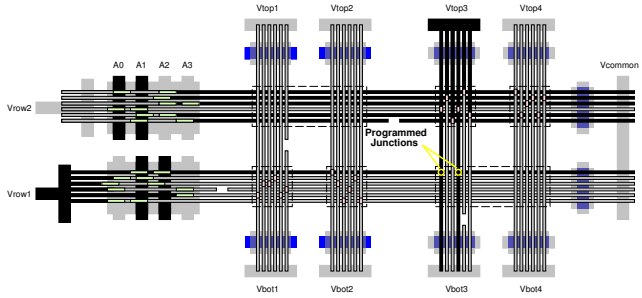


Figure 17: Assign OR-terms for Computing XOR2



A: Program B (bottom 1010) to $\overline{A} + B$ (top 1100) junction



B: Program $\overline{A} + B$ (top 1100) to XOR (bottom 0110) junction

Black lines are at suitable programming voltages or are driven high for control; light lines are at non-programming voltages.

Figure 18: Programming Diode Junctions

we set the low address to 1010 to select B's OR term and the high address to 1100 to select the $\overline{A} + B$ OR wire (See Figure 18A). Similar to polarity testing above, we precharge the bottom plane wires to high and then drive V_{row1} to low so that only the 1010 address is low and enables conduction to the OR plane. We drive V_{row2} directly to the low voltage needed for junction programming. We drive V_{bot2} to the high voltage needed for junction programming, and leave V_{bot1} at a nominal voltage so that we only program on the non-inverting B input. We keep V_{top3}, V_{top4}, V_{bot3}, V_{bot4} at nominal voltages so we do not program any junctions in the bottom-right OR plane while we are programming our intended junction in the top-left OR plane.

When we want to program a junction in the bottom-right OR plane, we suitably drive programming voltages on V_{top3} or V_{top4} and keep V_{bot1} and V_{bot2} at nominal voltages. For example, in Figure 18B we show programing of the $\overline{\overline{A} + B}$ to XOR junction. Here V_{top3} is placed at the high programming voltage since we want an inverting connection, and V_{top4} is held at a nominal voltage along with V_{bot1} and V_{bot2}.