FLOATING-GATE-PROGRAMMABLE AND RECONFIGURABLE, DIGITAL AND MIXED-SIGNAL SYSTEMS

A Thesis
Presented to
The Academic Faculty

By

Richard B. Wunderlich

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
January 2014

CONTENTS

CHAPTER 1

OVERVIEW

This body of work as whole has the theme of using floating-gates and reconfigurable systems to explore and implement non-traditional computing solutions to difficult problems. Various computational methodologies are used simultaneously to solve problems by mapping pieces of them to the appropriate type of computer. There exists no systematic approach to simultaneously apply analog, digital, and neuromorphic techniques to solving general problems. Typically, this is a very difficult task, and one that few attempt to undertake. However, when done right, solutions can be found with orders-of-magnitude improvement over existing solutions restricted to using only one type computational domain. To that end, I have helped build large and complicated reconfigurable systems (and software tools for helping to use these systems) capable of implementing solutions to problems in all three of those domains simultaneously. These systems are used to explore and implement these cross domain solutions to difficult problems.

The earlier work was involved with simply applying floating-gate technology to improving the building blocks of digital systems. Through that early work a new logic family built from floating-gate transistors was discovered, a Logical Effort compatable power analysis technique was developed, and low power floating-gate based FPGA was implemented. This work was then merged with existing research in the group involving solving problems using reconfigurable analog, and neuromorphic techniques. Thus converging on the mentioned systems that allow one to solve problems using techniques from all three domains: analog, neuromorphic, and digital.

CHAPTER 2

BACKGROUND

Since we are trying to solve a wide variety of computational problems, we will be implementing systems that are to certain degree, general purpose. The approach we will take will largely be one of reconfigurable systems. Low level computational elements (digital or analog devices) are placed in a sea of flexible wiring. High level functionality is produced by building large circuits out of these devices by using the flexible wiring (interconnect) to reconfigure the devices into a new topology. Switches are the smallest elements of the interconnect, and provide the utility of being able to either connect two nets together, or not, decided by the state of some memory element assigned to that switch.

In the majority of modern FPGAs, this is implemented by a single nFET (or sometimes a transmission gate) whose gate is driven by SRAM. Our reconfigurable systems, however, use floating-gate transistors as both the switch and the storage device. Figure 1 shows examples of SRAM based and floating-gate based switches.

2.0.1 SRAM Based Switches

One way to increase the efficiency of an SRAM based switch is to use just just use a single nFET, instead of a transmission gate (a pFET and nFET in parallel). This reduces parasitic capacitance at a cost of conductance, but the net effect is usually a faster switch. This circuit, when both the SRAM and logic are on the same voltage, only passes up to about a threshold voltage less than the rail for a logic high output. This reduces headroom and noise margin and significantly increases the leakage power in gates driven by this lower logic level.

Consider the tristate inverter of Figure 2a. Its function is to invert the input signal when V_{SEL} is high, connecting the output through a low impedance path to either the V_{DD} or GND , and to disconnect the output from any rail voltage when V_{SEL} is low. A

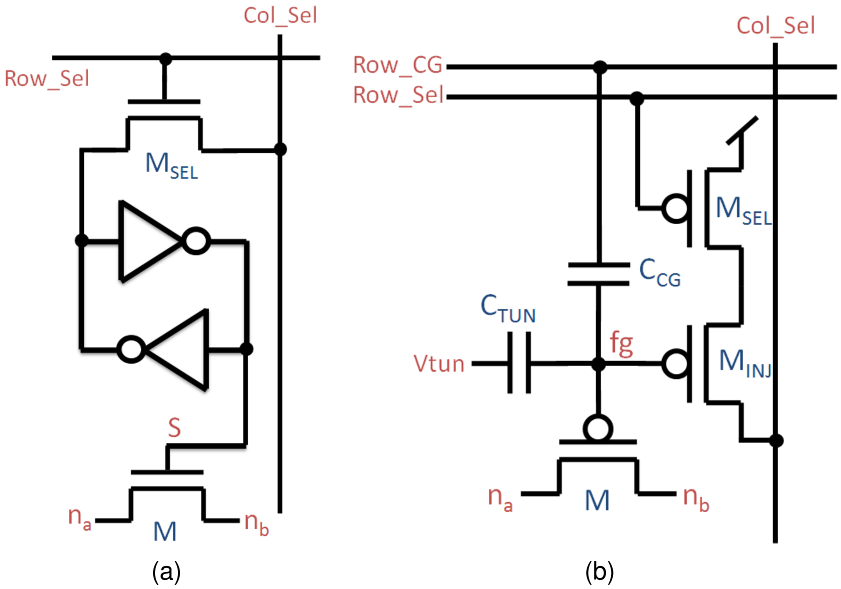


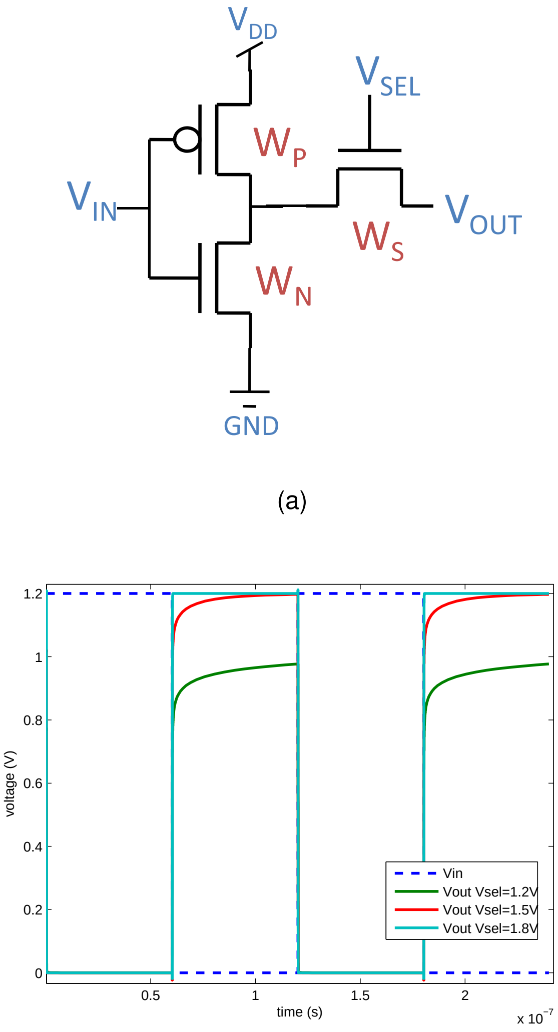
Figure 1. a) Switch element M is an nFET with SRAM based state storage as is typical of modern FPGAs. $M's$ state is set by turning on M_{SEL} and driving $ColSel$ high or low. b) Switch element M is a pFET whose gate has no DC paths under run-time bias, and whose state is stored as charge trapped on the fg node. Negative charge is added through M_{INJ} by channel hot electron injection or removed through C_{TUN} by Fowler-Nordheim tunneling

transient simulation of input and output voltage for this device can be seen in Figure 2b for various values of V_{SEL}. An idiosyncratic output is observed for when the gate tries to pull the output high. This is the well known threshold drop behavior of nFETs attempting to pass logic level high signals.

This response is due to the output of the gate being the source terminal of the passfet nFET. Assuming that the output is initially low, and that the input makes a transition from high to low, the tristate's pull-up network turns on and starts to pull the output high. However, as the output voltage starts to rise, the source voltage of the nFET starts to increase and eventually causes the passfet to enter subthreshold, where conduction is significantly weaker. The EKV model predicts this to be when

$$\kappa(V_G - V_{T0}) - V_S < 0, \qquad (1)$$

where κ is the capacitive coupling ratio of the gate voltage, V_G , into the surface potential



(a)

(b)

Figure 2. Transient input and output voltages for a tristate inverter with $V_{SEL} = 1.2V$, $1.5V$, and $1.8V$ for an arbitrary load. V_{DD} was $1.2V$ and the threshold voltage for this process is about $0.4V$.

of the channel, and all voltages are referenced to the bulk. In this case, $V_G = V_{SEL}$ and $V_S = V_{OUT}$, such that the maximum voltage the output can go to before the onset of subthreshold is

$$V_{out} = \kappa(V_{SEL} - V_{T0}) \tag{2}$$

To keep the passfet above threshold during the entire transition, V_{SEL} must be biased higher than V_{DD} :

$$V_{SEL} = \frac{V_{DD}}{\kappa} + V_{T0} \tag{3}$$

Figure 2b shows that for $V_{SEL} = V_{DD}$ that the output quickly reaches to about a threshold drop down from V_{DD} before the passfet leaves above threshold and then starts to slowly conduct up towards V_{DD} in subthreshold. Larger values of V_{SEL} cause the passfet to stay in above threshold longer, and for $V_{SEL} = 1.8V$ the passfet is in above threshold the entire time and the output voltage quickly converges on V_{DD}.

If the delay of the gate is measured as the 50% rail-to-rail propagation delay, then the time scale of this figure is too large to see any speed differences, as for all values of V_{SEL} in this figure, the onset of subthreshold is after the output goes to $V_{DD}/2$.

Figure 3 shows the average propagation delay speedup of a tristate inverter in a chain of tristate inverters versus V_{SEL} for various h , where each stage $h_i = h$, for various h. Here h stands for the *Logical Effort* term, *electrical effort*, which is closely related to fanout and load capacitance, and is defined as the ratio of load capacitance seen by the driving gate to the driving gate's input capacitance, $h = C_{out}/C_{in}$.

Note that speedup increases monotonically with V_{SEL} and h, which makes sense, as the smaller the load capacitance, the more delay is dominated by the gate's own parasitic capacitances which are not discharged by the passfet. For this particular process and gate topology, simulation shows speedups in excess of 30% when $V_{SEL} > V_{DD} + V_{T0}$.

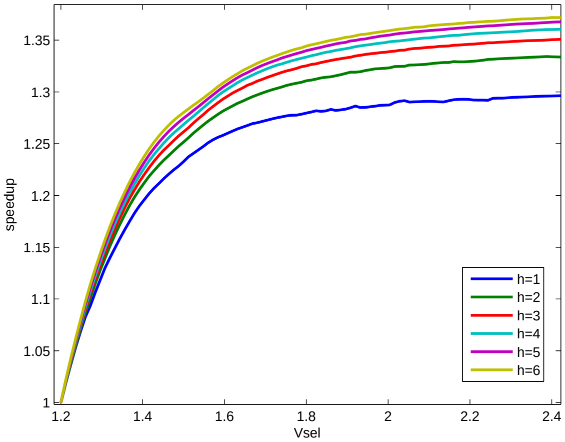


Figure 3. Average propagation delay speedup of a tristate inverter in a chain of tristate inverters versus V_{SEL} for various h. Where each stage of the chain has the same h.

SRAM can achieve these results by running the SRAM rail voltage at a higher voltage than the rest of the logic in the chip. But this comes at a cost of an increase in leakage current in the SRAM. Floating gates, however, can do this without increasing memory stand-by power, while also simultaneously reducing all stand-by power from off switches.

2.0.2 Floating-Gate Based Switches

Floating-gate transistors are normal MOSFET devices where the gate is completely insulated by silicon-dioxide. This means that all terminals capacitively couple onto the gate, and the device's effective threshold is modified by charge trapped on the gate. The modeling of the behavior of this device can be achieved by substituting for the gate voltage the following in any MOSFET current equation

$$V_{fg} = \frac{1}{c_{tot}}(Q_{fg} + \sum v_i c_i) \tag{4}$$

where c_{tot} is the total capacitance at the floating-gate, c_i and v_i are the capacitance coupling into the floating-gate and the voltage at the i^{th} node of the device. Q is the net total charge on the floating gate that is modified (or programmed) to set the state of the device.

Since the gate has no DC path to ground, the charge Q is trapped and remains on the floating-gate during normal circuit operation. However, for the floating-gate to implement the state storage of the switches, this charge has to be modifiable in a selective manner.

Figure 4 shows an array of floating-gates. All floating-gates are erased by applying a very large voltage to the global signal $V_{TUN}!$ which enables electrons to flow off of the floating-gate nodes into the $V_{TUN}!$ net by Fowler-Nordheim tunneling. This makes all floating-gates very positive in potential, and effectively turns off all floating-gate pFETs. To selectively turn on a pFET floating-gate, electrons are injected onto the floating-gate by way of impact based channel hot electron injection through M_{inj} (Figures 1b and 4) [1]. If A_{VDD} is set to a high enough voltage to allow for injection, the i^{th} floating-gate is injected by setting RS_i low and CS_j low. Injection takes place in regions where the electric field is strong enough to heat a significant portion of the minority carriers in the pFET channel to energies high enough to conduct in the silicon-dioxide, and the field in the oxide is such that those carriers are attracted to the floating-gate. The highest electric fields are achieved by operating the device in subthreshold, where almost all of the source to drain voltage is dropped in a very short region near the drain, and setting the source to drain voltage as high as possible. In this case, the probability of injection of each carrier is maximized, but when this is maximized the amount of available carriers tends to be very low. Because of this, injection rate is maximized somewhere near the onset of above threshold, and tends to be very poor in regions of high above threshold (lots of carriers, not very high fields) and deep subthreshold (very high fields, but not many carriers) [1]. In order to bias the device in regions conducive to injection, and further optimize the dynamic range of programmability, there is a control gate on each floating-gate that is controllable during injection.

Figure5 shows the floating gate erase and program method. The erase comprises a

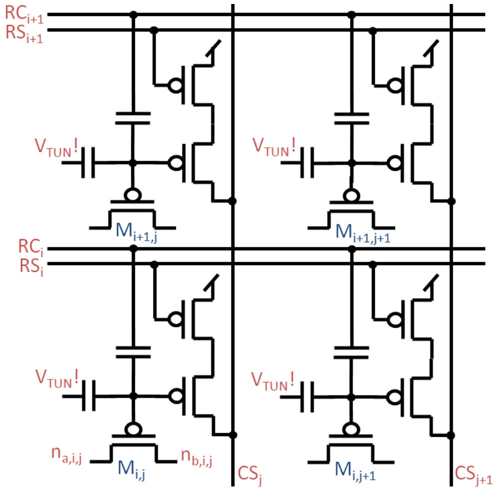


Figure 4. Programming is achieved by globally removing charge from the floating-gate nodes through C_{TUN} via Fowler-Nordheim tunneling, and then selectively adding charge through $M_{i,j}$ with impact carrier hot channel electron injection. Injection of charge per row is controlled by the selection lines CS_i, and per column by the drain lines CS_j.

forward tunneling operation that removes electrons from all floating gates in the system. Due to different starting conditions, and different tunneling rates, floating-gates end up at different states at the end of the tunneling. The different starting conditions are due to previously programmed state, or floating-gate drift due to multiple erase exposures. The rates are largely layout dependent, but are also significantly impacted by process variations across the chip.

In general, tunneling slower, ie. at a lower voltage, for longer periods of time, can bring the entire population of floating-gates to the same average. But this can take an unreasonable amount of time. Therefore, an aggressive forward tunneling operation takes place that pushes the worst case floating-gates into the accepted off range. But also happens to push some of the floating-gates into such an off state that they can not be programmed with injection (injection being proportional to current). A reverse tunneling pulse is used to bring the populations back together. Then individual floating gates are programmed selectively using hot electron injection.

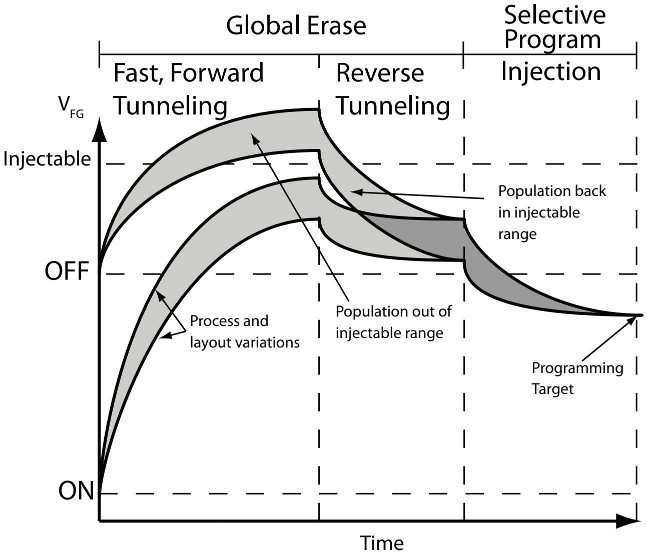


Figure 5. The floating-gate programming procedure uses a global erase comprising a forward and reverse tunneling phase, and then individual floating-gates are selectively programmed using hot electron injection.

Well designed floating-gates and programming techniques are able to produce run-time voltage ranges on the floating gates (this range is fundamentally limited by tunneling) that are higher and lower than the supply nets. This makes a switch with a much higher I_{on}/I_{off} value than an SRAM based switch would be able to achieve. And since accurate programming algorithms exist that can precisely set the charge on floating gate, a continuous spectrum of currents between the on and off state can be achieved. This property makes the floating-gate transistor useful as an explicit circuit element to modify the behavior of analog and digital circuits, or can be used to perform meaningful computation with interconnect devices.

CHAPTER 3

CAPACITIVELY COUPLED FLOATING GATE DIGITAL CIRCUITS

In synchronous digital design the delay of the critical path is used to set the clock frequency in order to guarantee that the propagation delays of all paths will satisfy the setup and hold times of the latches used to synchronize the data. In the classical CMOS design space, the choices allowed on these paths include gate mapping and transistor sizing. While a larger, more parallelized gate architecture with transistors sized through methods like Logical Effort are able to sacrifice area and power in order to further enhance the performance of the critical path, a smaller, less complicated architecture with minimally sized transistors can be used on non-critical paths, sacrificing speed for area and power. The synchronization, however, still becomes a necessity as given by these quantized design options, and the intrinsic process variability of fabrication, it becomes nigh impossible to equate the propagation delays of all paths. All paths will end up with a varying amount of slack Figure 6. Giving the designers more options to allow them to optimize out the slack, as well as to further control the power, area, and delays can pave the way for a better design and has been the topic of much research, giving rise to commercially implemented techniques such as: Dynamic Voltage Scaling (DVS), and variable threshold transistors [2].

While lowering the clock frequency of a chip at run time can certainly reduce the average power consumed by the chip, it hurts average MIPS/Watt performance giving rise to an increase in energy used per computation. This is because it increases the amount of slack in every path on the chip, and while the dynamic energy consumed per computation may stay the same, an increase in time is spent dissipating static power. In order to decrease the energy consumed per computation, techniques like DVS scale the rail voltage of the chip with the frequency to reduce the slack.

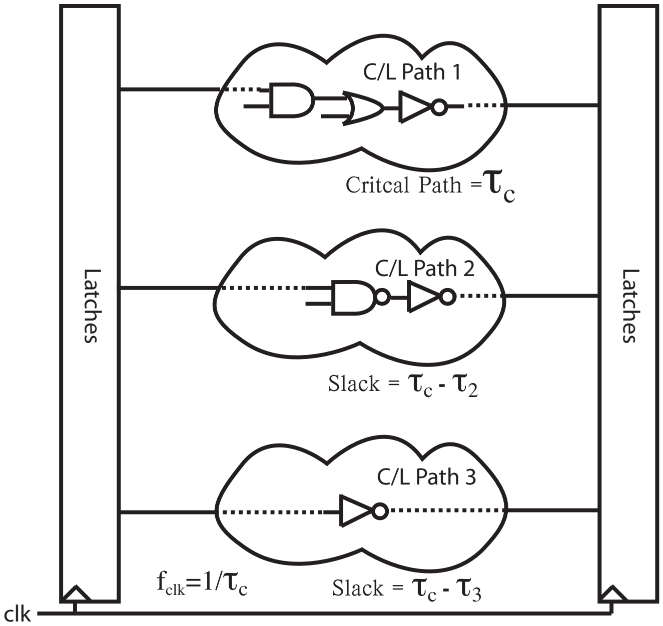


Figure 6. Various cominational logic paths. The maximum delay worst case delay path sets the clock period, and all other paths operate with timing slack.

Changing the rail voltage has a many fold effect on a digital circuit's operation. Increasing VDD increases the amount of charge that has to be moved onto a gate in order to fully charge it up which linearly increases dynamic energy per switch, and because the driving gate's ability to source current increases polynomially with VDD, a net decrease in switching time results. An increase in VDD, while usually not significantly increasing subthreshold leakage currents, increase the voltage drop over which the leakage currents pass and therefore linearly increases static power dissipation.

Adjusting the threshold voltages of the transistors that make up the gates gives rise to a new design trade-off. Increasing the threshold of a device exponentially decreases the subthreshold leakage currents but polynomially decreases the driving current when the devices are supposed to be on. Adjusting the threshold allows the designer to trade off static power dissipation and speed and can be done at fabrication time by changing the doping profiles or the oxide thicknesses.

An important consequence here is that given a particular path architecture and a target path delay, there is a two dimensional design space for V_{DD} and V_T in which a subset of points exist that will produce the target frequency, and within this space dynamic and static power dissipation are traded off, and that there exists a particular VDD and V_T for that path that globally minimized total power [3].

There exist significant barriers to picking the optimal V_T for all paths. While the threshold voltages can be adjusted rather harmlessly by changing the doping profiles, the select ability of this offset is coarse-grained in that doping profiles are applied to either the entire chip or very large portions. Changing the doping can also produce adverse effects on carrier mobility within the devices, hindering speed even further. Threshold adjustments in the form of varying oxide thicknesses can be applied very selectively to individual paths but the choice of thickness is usually process limited to only a couple. Varying V_T in these manners are static at run time and allow for no run time trade off of energy per cycle and cycle time.

Having varying VDDs for spatially local paths becomes logistically implausible. For every new VDD desired a new rail routing network needs to be introduced and a new power source needs to be fabricated. This aside, two adjacent paths with varying VDD will either be forced to have separate wells for their pFETS or suffer significant well leakage between the rails. Another barrier is that paths with different VDDs produce different logic levels, and upon mixing these signals it may be necessary to introduce logic level shifters. This is why in VDS VDD is often varied globally across the entire chip, or at best variable VDDs exist on the chip but in very coarse grained regions to minimize the area penalties. To further limit the choices of DVS, there exists maximum and minimum values for acceptable CMOS operation for VDD [2]. Significant problems arise when VDD approaches the reverse bias breakdown voltage of the PN diode junctions or within twice the threshold voltage of the devices.

We present a capacitively-coupled, floating-gate based digital CMOS logic family (CCFG CMOS) that allows a designer to trade-off dynamic power versus delay, and static power versus delay, without the limitations of DVS.

3.1 CCFG CMOS

As can be seen in Fig. 7, there are two capacitively coupled inputs to the floating-gate of the transistor. The input labeled V_{IN}, coupled through C_C, serves as the regular transistor gate input used in CMOS devices, while the other input, labeled V_B and coupled through C_B, is used as a bias voltage for the transistor. The pull-up and pull-down networks of CMOS logic gates can be built using floating-gate nFETs and floating-gate pFETs as per normal CMOS design using the V_{IN} inputs as the regular gate inputs. The bias inputs of each floating-gate nFET are tied together to form the nFET bias net, V_{BN}, while the bias inputs of the pull-up network are assembled to form V_{BP}. An inverter and a two-input NAND gate made by this arrangement are shown in Figure. 7.

The capacitance C_{IN}, seen by the regular input, becomes the parallel combination of C_C and the rest of the capacitance seen at the floating gate node including C_B and the capacitance looking into the gate C_G. The smaller C_C, the smaller the input capacitance, and the less charge that has to be moved in order to charge up the node. This in turn linearly decreases the amount of energy consumed per switch with decrease in input capacitance. The caveat here is that the contribution of V_{IN} to the floating gate voltage is then proportional to the capacitive divider of C_C and the total capacitance seen at the floating gate, C_{total}, so while the driven gate requires less charge per switch, the strength of the driving gate at supplied current decreases polynomially with decrease in the amount coupled. This gives rise to a fabrication time trade off of dynamic power dissipation and speed.

The bias voltage gets coupled into the floating-gate much in the same way as the input voltage but through C_B. However, this voltage remains static during normal operation of the gate and after charging up once serves to simply couple an extra bias voltage into the floating-gate to effectively adjust the the threshold of the device. By increasing the V_{BN} voltage a polynomial increase in driving current results which increases speed, but an exponential increase in subthreshold current occurs increasing static power dissipation. This allows a run-time adjustment of speed and static power dissipation.

Dynamic power dissipation can be optimized on a per path basis using CCFG CMOS by choosing the capacitive dividers at fabrication time without any of the penalties associated with trying to apply multiple VDDs to spatially local paths, or without having to mix and convert different logic level signals.

Threshold adjustments of floating-gate CMOS inverters have been explored in the past by [4] and [5] by moving charge on to and off of the floating-gate through hot-electron injection / Fowler-Nordheim tunneling and UV light conductances respectively. In this scheme static power dissipation can be adjusted on a per path basis using CCFG CMOS and tying all of the nFET biases together and all of the pFET biases together within the path and applying bias voltages accordingly. This incurs a similar, but not nearly as large, problem as having multiple VDDs in that the finer grain the solution the more voltage sources are required. The benefit with CCFG CMOS is that the voltage sources are not constrained locally by well leakage, and simply act as references drawing little to no measurable power and can be routed therefore on a minimally invasive network.

3.2 Simulation

The design space for a few simple paths were explored using SPICE. The paths were variations on inverter chains. All of the inverts involved were minimum sized, the paths contained a varying number of stages, and simulations were carried out for standard

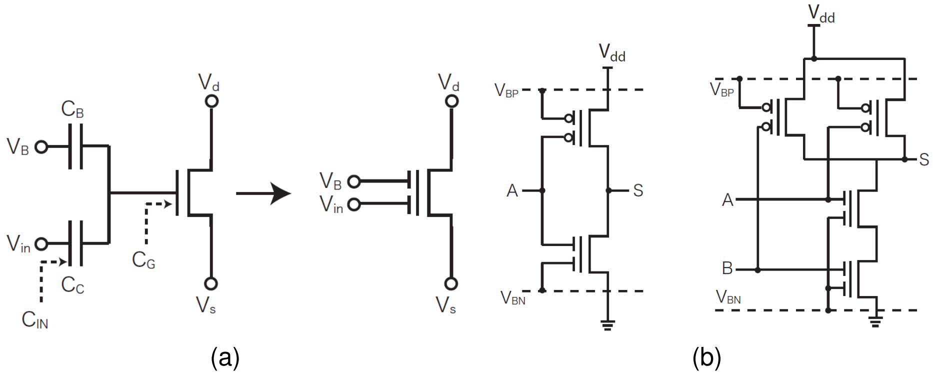


Figure 7. Multiple floating-gate transistor and its abbreviated drawing. One can see how the devices come together to create complex CCFG CMOS gates with local, shared biases.

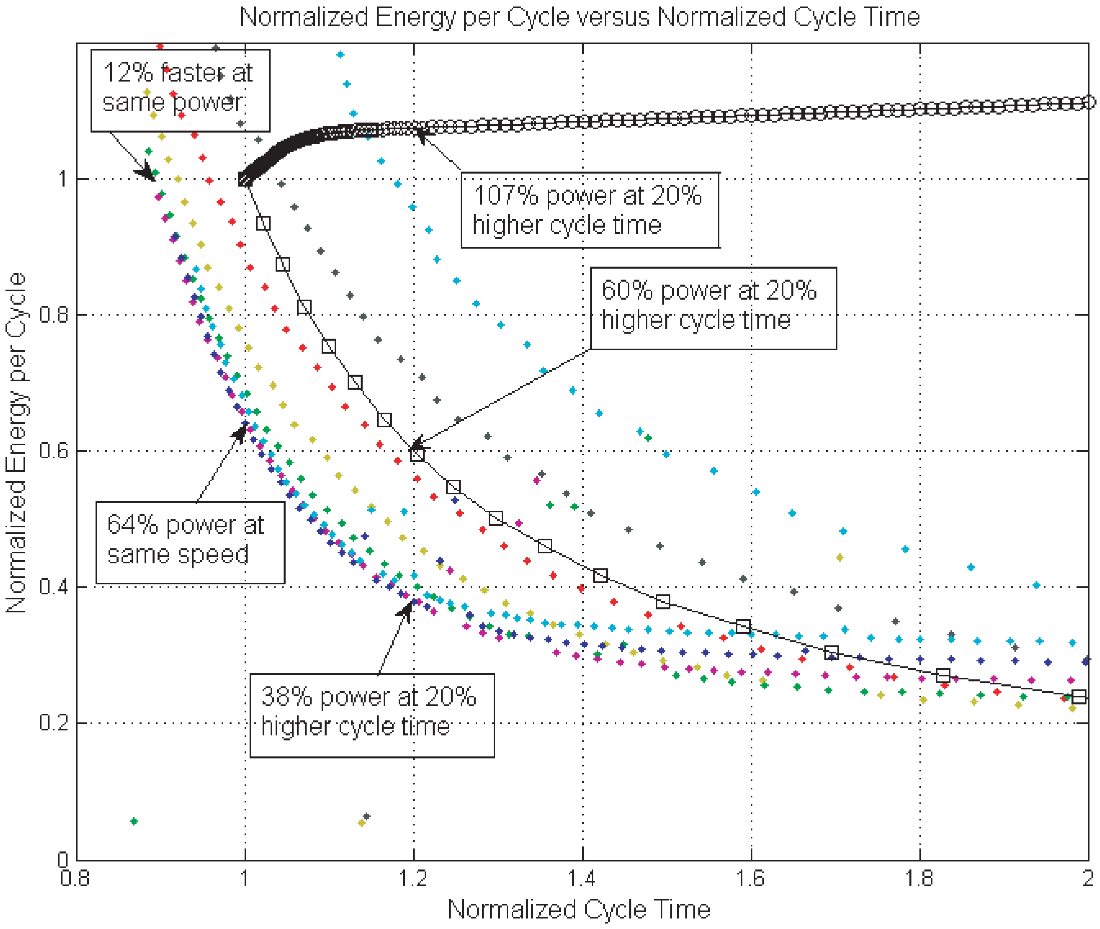


Figure 8. Normalized energy per cycle versus normalized cycle time for various six stage inverter chains simulated with SPICE3F5 using the BSIM3.3 models provided by MOSIS for the TSMC 0.35u process: standard CMOS implementation shown in dense black circles, standard CMOS using DVS shown in sparse black squares, and CCFG CMOS implementations of varying capacitor sizes shown in colors

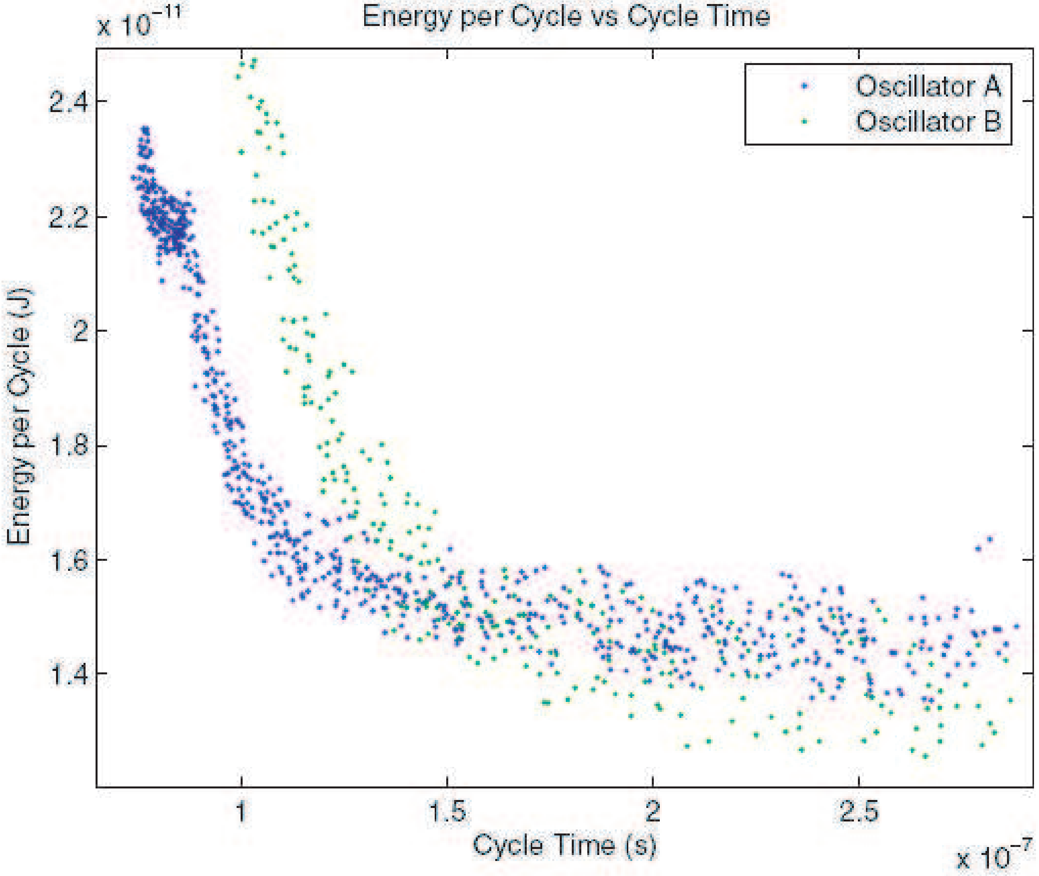
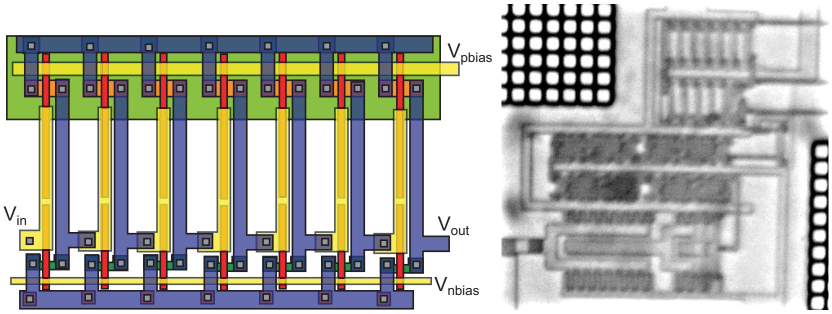


Figure 9. Energy versus cycle time plot for a CCFG CMOS inverter chain that was fabricated in a 0.35um process. The two different curves were for two different values of input capacitance for the CCFG inverters, and moving along each curve was done by varying the bias voltages on the coupling lines.



(a) (b)

Figure 10. a) Partial layout of the fabricated CCFG CMOS inverter chain. Using a two poly process the common poly1 gate is split and overlapped with a strip of poly2 which becomes the new gate input. The top and the bottom portions of the original poly1 strip are crossed with a horizontal strip of poly2 which spans all of the gates in a path to comprise the bias lines. b)Partial die photograph of the test chip, a CCFG CMOS ring oscillator is in the top right.

CMOS structures with and with out DVS, and CCFG CMOS structures (Figure. 7) of varying capacitive dividers and bias voltages.

The results for a particular path architecture of six minimum sized inverters in a 0.35um (the same process chosen for verification through fabrication) digital process can be seen in Fig. 8. This figure shows energy consumed per cycle (EPC) for various path implementations normalized by the EPC of the standard CMOS implementation running as fast as it can versus the cycle time normalized by the delay of the standard CMOS path.

The densely populated line of black circles represents the standard CMOS case. The data ready was taken at 50% of the rail to rail voltage, and as can be seen the EPC increases rapidly until it settles in a rather linear slope representing only further static power dissipation as the path waits the amount of slack time before switching again. The path populated with sparse black squares represents the same standard CMOS implementation with DVS, which does rather better when slack is introduced. Under a 20% increase in cycle time the standard CMOS case consumes 7% more power and the DVS case consumes 40% less power. In practice, these gains will vary with circuit topology.

Each colored curve represents a CCFG CMOS implementation with different capacitor sizes for C_C with $C_B = C_C/10$, moving curves from left to right represents a decrease in C_C, while moving along the an individual curve from left to right represents a decrease in bias voltage V_{BN} with $V_{BN} = -V_{BP}$. As can be seen, for the same EPC, a CCFG CMOS implementation can operate 12% faster, or at the same speed a reduction in EPC of 36\% can be achieved. In this case, the CCFG CMOS path really shines when a 20\% increase in slack is introduced, where a 62% reduction in EPC is observed.

3.3 Measurement

A chip was designed to help validate the results of simulation, as well as to further explore the area ramifications of designing with CCFG CMOS. A double-poly process was chosen because it allowed for a very tight layout. A partial layout involving a chain of inverters is shown in Fig. 10a, when layout is done in this manner a minimal amount of area penalty is incurred in implementing low capacitively coupled CCFG CMOS, however, in designing for very fast structures that require higher capacitive coupling of the inputs a larger area penalty will be incurred. CCFG CMOS oscillators were also fabricated using MOSFET capacitors to explore the layout consequences in the absense of a double poly process. Using MOSFETs certainly increases the area overhead but still works. To chips were also also erased by exposure to ultraviolet light to normalize any change that may have accumulated on the floating-gates during fabrication.

In order to get accurate speed measurements the inverter chains were implemented as 13 stage ring oscillators (12 CCFG CMOS inverters and one CCFG CMOS NAND gate) whose output was divided down many times to a reasonable speed to get off chip. A partial photo including a ring oscillator and instrumentation circuitry is shown in Fig. 10b. A few EPC versus cycle time sweeps are shown for two different CCFG CMOS ring oscillators built using double-poly capacitors in Fig. 9. The power dissipation was higher than expected for all floating-gate implementations due to fabrication errors which have been fixed in a new version of the chip that is in fabrication, but the functionality of CCFG CMOS and the trends for the trade offs as expected from theory and simulation are clearly shown.

CHAPTER 4

LOGICAL POWER

In this chapter, we propose a method to predict power that is compatible with Logical Effort (LE) that we call Logical Power (LP). Logical Power is designed for the characterization of digital CMOS logic gates at the level of hand analysis in order to predict delay and energy consumption. The models for delay and energy presented are naive, first-order, linear predictors that take into account load capacitance while ignoring input slew rate, but are shown to be quite accurate. LE has previously been shown, and LP will be shown, to be greater than 90% accurate versus SPICE simulation for a wide variety of cases. An approximate analytical solution to the optimal inverter chain sizing for minimized energy under constrained non-minimum delay is proposed and shown to be within 2% of optimal when compared to the exact numerical solutions to the LE and LP equations.

4.1 Introduction

Logical Effort [6] is a model for unitless approximations for normalized path propagation delay, d, for an N deep logic chain with last stage load capacitance, $c_{N+1} = c_L$, as

$$d = \sum_{i=1}^{N} f_i + p_i = \sum_{i=1}^{N} g_i h_i + p_i = \sum_{i=1}^{N} g_i \frac{b_i c_{i+1}}{c_i} + p_i, \tag{5}$$

where f_i is the *effort delay* of the i^{th} stage in the path associated with the gate driving its load, which is proportional to the ratio of the load capacitance to the input capacitance (called the *electrical effort*) of the driving gate by the gate dependent scalar, g_i, called the gate's *logical effort*. The *parasitic delay*, p_i , is the load and gate size independent delay associated with the gate driving its own internal parasitic capacitances. The capacitance coefficient c_i is the input capacitance to the i^{th} stage, with $b_i c_{i+1}$ (see Figure 11) being the total load capacitance seen by the i-th stage, and represents the effective

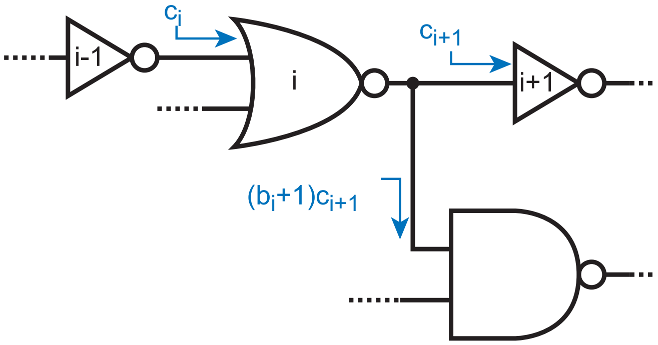


Figure 11. An arbitrary logic path is illustrated. The i^{th} stage is a two-input nor gate with output load capacitance equal to $b_i c_{i+1}$, the load capacitance of the previous stage is equal to the input capacitance of the i^{th} stage, c_i .

sizing of the gate- double the size of a gate, double the input capacitance. Absolute path delay is given by $d_{abs} = \tau d$ where τ is the absolute *effort delay* of a minimum sized reference inverter driving a minimum sized reference inverter that sets $g_{inv} \equiv 1$.

For the purpose of *Logical Effort* and this chapter, a gate is defined as any digital CMOS configuration of a pull-up network of pFETs and complementary pull-down network of nFETs with a single output and one or many inputs applied only to the MOSFET gate terminals, where the ratio of any two transistor sizes is fixed. It is important to emphasize that the parameters g_i and p_i are simply then gate dependent and are independent of the gate's particular sizing.

The LE formulation of delay is simple. It treats MOSFET capacitances as constant, pull-up and pull-down networks as either open circuits or effective resistances, and ignores input slew rate. However, it remarkably and consistantly is able to predict the delay of arbitrary logic paths in modern processes to within 10% error, and usually with much higher accuracy. In general, the parameters *logical effort* and *parasitic delay* will be different for a falling and rising transition, as well as being input-gate dependent. One can obtain decent results with average parameters, or can use specific parametres for transition and input.

The real power of LE is that it solves the sizing problem of gates in a path in order to minimize delay. When the first stage gate is set to be constant and the load capacitance also to be a constant, there then exist N-1 degrees of freedom for sizing choices for the path. There is an exact set of gate sizings that minimizes Equation 5 and the solution is obtained when the *effort delay* of all stages are made equal. This happens when

$$\hat{f} = g_i \frac{b_i c_{i+1}}{c_i} = (\prod_{i=1}^{N} g_i b_i h_i)^{\frac{1}{N}} = (GBH)^{(1/N)} = F^{(1/N)}, \tag{6}$$

where $G = \prod_{i=1}^{N} g_i$, $B = \prod_{i=1}^{N} b_i$, and $H = \prod_{i=1}^{N} h_i = \prod_{i=1}^{N} C_{i+1}/C_i = C_{L/}C_1$. The path effort, F, is then independent of the gate sizing choices, and minimum delay is immediately obtained as

$$d_{min} = NF^{(1/N)} + P \tag{7}$$

where $P = \sum_{i=1}^{N} p_i$. Predicting delay and gate sizing for speed is what *Logical Effort* does well, but what LE does not do is identify the energy ramifications of any particular set of gate sizings. The sizing for minimum delay has only one solution, but what about sizing for a target non-minimum delay that minimizes energy? In [7], for instance, *Logical Effort* is applied to various high valency adders, which provides a fast and efficient means of evaluating the delay trade-offs of the different architectures, but provides no insight into the energy trade-offs.

Figure 12 shows constant delay contours using text-book values for the *Logical Effort* parameters for the sizings of the second and third stages for a three-stage logic path of inverters with fixed load and first stage input capacitance as predicted by Equation 5. *Logical Effort* predicts the minimum delay by Equation 7 obtained with the sizings by Equation 6 and this is the point at the center of the graph. For any delay greater than the minimum delay, there are contours of values able to obtain that delay. These contours are of increasing delay as they move outward from that minimum delay point. Delay for each contour is labeled normalized by the minimum delay sizing.

If a path had a timing delay requirement of 5% higher than the minimum delay it

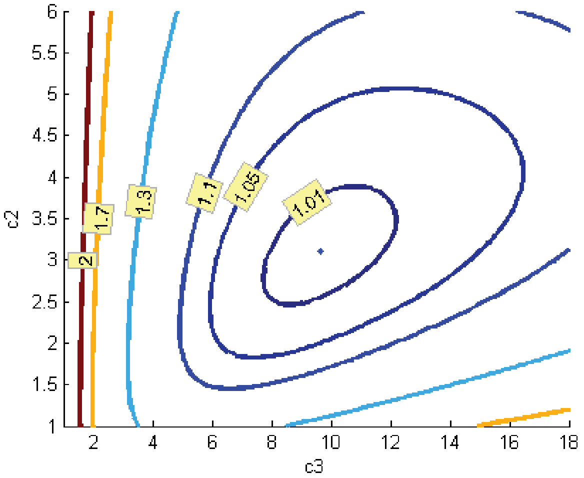


Figure 12. Constant delay contours for C_2 and C_3 of an arbitrary three-stage logic path of inverters with fixed load and first stage capacitance, $F = 30$. The contours are labeled in normalized delay to that of the minimum delay, with the optimal minimal delay sizing shown as the point in the center.

could obtain, then any point on the 1.05 contour shown would suffice. Moving to the top right in the graph corresponds to greater total capacitance, so perhaps a good choice on the constant delay contour would be the one towards the bottom left as this would trend to a smaller design and less dynamic energy. But to figure out exactly which point minimizes energy, the method of *Logical Effort* needs a compatible energy prediction methodology.

The method of *Logical Effort* is explained in detail in [8]. We show its validity in modern processes as well as go into further detail how to extract its parameters. In [9] *Logical Effort* is applied to cyclic asynchronous control paths, and a similar method for analyzing power is introduced that considers only active power as a result of the dominant parasitic and gate capacitances as a function of sizing, but provides no method for accounting for *static power,* the active power due to intermediate node capacitances, and limits its scope to only scenarios in which the delay of all stages are the same. In [10] the energy-delay trade-offs of sizing individual transistors are explored using the same linear models as in [9], but only heuristics are suggested for the solving the optimization problem.

This section served as a crash-course in *Logical Effort,* and explained the motivation for *Logical Power*. The rest of this chapter is organized as follows. Section 4.2 derives *Logical Power* as a method to predict path energy as a linear function of the sizing choices of gates in the path. Section 4.4 investigates the validity of the linear predictors of *Logical Effort* and *Logical Power* with a SPICE simulation based methodology, and a means of extracting the gate dependent parameters of the two methods. Section 4.3 details the pencil and paper method of estimating the parameters of *Logical Effort* and addresses the problems of using a similar method for estimating the paremters of *Logical Power*. Section 4.5 provides a statistical analysis of the two methods' abilities to predict power when compared to detailed SPICE simulations of randomly generated logic paths. Section 4.6 applies both methods to the optimization problem of sizing an inverter chain for the minimization of energy under delay constraints and proposes an approximate but analytical solution to the exact sizings required to do so.

4.2 Logical Power

Logical Power is a linear predictor for power designed to be compatibile with *Logical Effort*- that is, its degrees of freedom are the same and limited to gate sizings. Energy is predicted on a per gate basis, using only these variables, and two new gate-dependent paramters, *active internal* and *static internal*.

The per-cycle energy consumption of the i^{th} gate is split into its *active* and *static* components as

$$E_i = E_{act,i} + E_{stat,i}$$

The *active energy* is then expressed as the sum of the *dynamic* and *short circuit* energy as

$$\begin{aligned} E_{act,i} &= E_{dyn,i} + E_{sc,i} \\ &= V_{dd}^2[c_{out,i} + c_{p,i}] + E_{sc,i} \end{aligned}$$

with the first term simply being load capacitances as seen from the gate output of the gate, the second term being internal capacitances, and *short-circuit energy* making up the third term. We define a new parameter, a_i , to predict the internal active enery of a gate, the amount of dynamic energy consumed by internal capacitances plus the short circuit energy of the gate, which we will refer to as the *active internal* coefficient, defined as

$$a_i c_i = c_{p,i} + \frac{1}{V_{dd}} Q_{s.c.,i}. \tag{8}$$

If the right hand side of the above equation scales linearly with input capacitance, we can write the *active energy* as

$$E_{act,i} = V_{dd}^2[b_i c_{i+1} + a_i c_i],$$

which is a reasonable approximation, as both internal capacitance and short-circuit current should double when a gate's width is doubled when all other factors are held equal (input and output slew rate, for instance). In [11], the authors show that when the input and output slew rates of a gate are the same that *short-circuit power* is a constant scalar multiple of its *dynamic power ,* which makes it reasonable to lump the two together using the same coefficient accounting for both, the accuracy of such an approximation relying on circuits to not vary too far from the equal stage delay points.

The *static energy* is then

$$E_{stat,i} = V_{dd} I_{stat,i} T$$

where $I_{stat,i}$ is the static current dissipated by the gate and T is the time the gate is static state, which will be assumed to be the cycle time. We define a unitless time constant, d_s, as

$$d_s = \frac{\tau_s}{\tau} = \frac{V_{dd}C_{inv}}{I_{s,inv}\tau}$$

The process dependent variable, τ_s, and its normalized value, d_s, can be interpreted as the amount of time a minimum sized reference inverter needs to remain in static operation to have its *static energy* equal to its *active energy*. We define a new coefficient to predict *static energy* as a function of input capacitance as

$$s_i = \frac{c_{inv}}{I_{s,inv}} \frac{d}{dc_i} I_{s,i}$$

where c_i is the gate's input capacitance that varies with its size, and c_{inv} is the input capacitance for the reference inverter gate.

If the static current scales linearly with input capacitance, we can then represent the *static energy* of the i^{th} gate as

$$E_{stat,i} = V_{dd}^2 \frac{d}{d_s} s_i c_i$$

where d is time normalized per-cycle that the gate remains static, and s_i is a scaling factor comparing the gate's leakage current divided by input capacitance to that of a reference inverter, and will be called the *static internal* coefficient that takes into account only leakage currents sourced by the gate, not that are dissipated in the gate. That is, for gate's with non-trivial ammounts of gate leakage current, some of the leakage power dissipated in a gate could have been sourced by a previous gate, and we do not want to double count this.

As a matter of convenience, we will use e , which has units of capacitance; consequently, absolute energy is then obtained by multiplying by V_{dd}^2 :

$$E = V_{dd}^2 e$$

The amount of energy dissipated by the i^{th} stage is then

$$e_i = b_i c_{i+1} + a_i c_i + \frac{d}{d_s} s_i c_i,$$

And the contribution of energy dissipation by the entire path due to the size of the i^{th} stage is then

$$e = (b_{i-1} + a_i + \frac{d}{d_s} s_i) c_i \tag{9}$$

The total path energy per cycle then becomes

$$e_{path} = c_L + \sum_{i=1}^{N} (b_{i-1} + a_i + \frac{d}{d_s} s_i) c_i \tag{10}$$

where $b_0 = 0$ to remove the input capacitance of the first stage out of the energy calculation. The total path energy equation assumes that every gate makes a transition during every cycle. In general, this is not the case, and for completeness a per gate activity factor, α_i , can be added to all terms associated with *active power,* resulting in

$$e_{path} = \alpha_N c_L + \sum_{i=1}^{N} (\alpha_{i-1} b_{i-1} + \alpha_i a_i + \frac{d}{d_s} s_i) c_i. \tag{11}$$

4.3 Parameter Estimation

In order to apply these methods, the gate dependent parameters of *Logical Effort* and *Logical Power* must be determined for the logic paths. These parameters are either extracted through simulation (Section 4.4), experimentation, or through topological analysis with simple circuit principles. While simulation and experimentation certainly produce the most accurate parameters, hand analysis methods allow for an intuitive understanding of how different gate topologies will affect the various parameters. Extracting parameters for *Logical Power* is the topic of this section, and we start from the approximations used in *Logical Effort.*

4.3.1 Logical Effort

For a simple inverter, the pull-up network is a single pFET and the pull-down network a single nFET. The pull-up and pull-down networks source current as functions of their gates, v_{in}, and drains, v_{out}. If the input slew rate is ignored and approximated as constant with respect to time, and it is assumed that for rising output transitions the pull-up network does not fight the pull-down network, and vice versa, then the delay of a pull-up transition (the time for the output to rise from $V_{out} = V_1$ to $V_{out} = V_2$) of a gate driving a fixed load can be approximated as

$$\triangle t = C_L \int_{V_1}^{V_2} \frac{1}{I_{pu}(V_o, V_i)} dV_o = C_L R_{pu}$$

where regardless of the shape of I_{pu} so long as the choices of V_1 and V_2 remain fixed, the value of the integral, the effective resistance over the pull-up transition R_{pu}, remains valid for varying load capacitances. This approximation is scaled to more complicated pull-up and pull-down networks by ignoring the intermediate node capacitances leaving only the dominant pole of the Elmore delay model caused by the output parasitic and load capacitances. In this approximation, series transistors are treated as a single transistor of length equal to the addition of both original transistor lengths, and parallel combinations leading to an increased effective width.

We define a reference inverter in Figure 13, with approximations for the scaling of effective pull-up, R_u, pull-down, R_d, resistances, gate input capacitance, C_{in} , and output parasitic capacitance, C_p with width are as follows:

$$R_u = \frac{R_p W_{min}}{W_P} = \frac{R_p W_{min}}{rW}$$

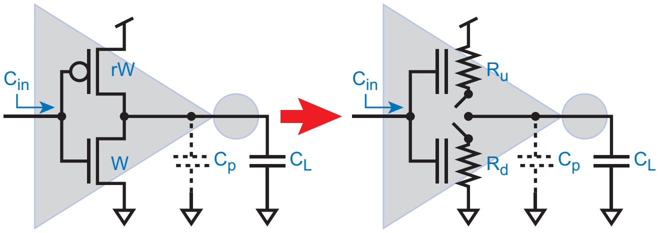


Figure 13. An inverter with pFET width r times that of the nFET showing input, output load, and parasitic capacitances is transformed into the pull-up and pull-down networks are replaced with their effective resistances.

$$R_d = \frac{R_n W_{min}}{W_N} = \frac{R_n W_{min}}{W}$$

$$C_{in} = \frac{C_{fet}}{W_{min}}(1+r)W$$

$$C_p = \frac{xC_{fet}}{W_{min}}(1+r)W$$

where R_P and R_N are the effective resistances of a minimum sized pFET and nFET respectively, and C_{fet} is the input capacitance of a minimum sized pFET or nFET (assumed to be relatively equal for both), and x is some scalar relating parasitic capacitance to gate capacitance.

Propagation delay is then modeled as some effect pull-op or pull-down resistance multiplied by the the total capacitance seen at the outout (load capacitance plus internal capacitance), so the pull-down propagation time is given by

$$\begin{aligned} d_d &= R_d(C_p + C_{out}) \\ &= \frac{R_n W_{min}}{W}[\frac{xC_{fet}}{W_{min}}(1+r)W + C_{out}] \\ &= \frac{R_n W_{min}}{W}C_{out} + xR_nC_{fet}(1+r) \end{aligned}$$

The first term is the part of the delay that scales with output capacitance and inversely with gate width, and the second term is independent of both capacitance and gate size. Transforming this equation into the form of *Logical Effort* results in

$$d_d = \tau(g_{inv}\frac{C_{out}}{C_{in}} + p_{inv})$$

Comparing these two equations, it becomes clear that the first term is the *electrical effort* and the second term the *parasitic delay*. Equating these terms results in

$$\begin{aligned}\tau g_{inv}\frac{C_{out}}{C_{in}} &= \frac{R_n W_{min}}{W}C_{out} \\ &= R_n C_{fet}(1+r)\frac{C_{out}}{C_{in}}\end{aligned}$$

where $g_{inv} \equiv 1$ and $\tau = R_n C_{fet}(1+r)$.

We define the *logical effort* of a reference inverter to be equal to one. The *Logical Effort* of any other gate is then approximated as the ratio of its driving strength multiplied by its input capacitance and the driving strength and input capacitance of the reference inverter:

$$g = \frac{RC_{in}}{R_{inv}C_{inv}} \tag{12}$$

The *parasitic delay* of the reference inverter is

$$p_{inv} = \frac{xR_n C_{fet}(1+r)}{\tau} = x \simeq 1$$

where in most processes x is nearly one.

In practice, it is difficult to size for equal rise and fall times, and generally impossible to do so for all possible values of h . In general, when the rising and falling *logical efforts*, g_r and g_f, are equal, the rising and falling *parasitic delays*, p_r and p_f, are not. However, for pedagogical reasons we will assume that a reference inverter is sized for equal rise and fall times and that this leads to $r = 2$.

Applying the model to an n-input NAND gate (Figure 14) and sizing for equal rise and fall times results in

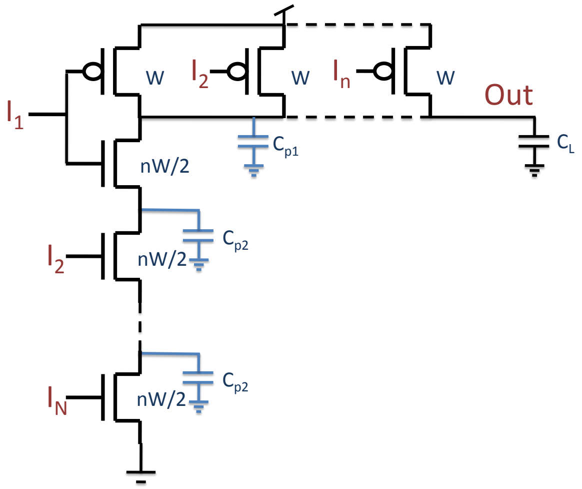


Figure 14. n-input NAND gate sized for equal rise and fall times.

$$R_u = \frac{2R_n}{W/W_{min}},$$

$$R_d = \frac{nR_n}{(n/2)(W/W_{min})},$$

$$C_{in} = (\frac{n}{2} + 1)C_{fet}\frac{W}{W_{min}},$$

and

$$g = \frac{2R_n(n/2+1)C_{fet}}{R_n 3C_{fet}} = \frac{n+2}{3}$$

In order to estimate simple parasitic delay, we continue with our approximation of ignoring internal parasitic capacitances and consider only those parasitic capacitances at the output that results in

$$\begin{aligned} \tau p &= \frac{2R_n}{W/W_{min}}\frac{(n+n/2)C_{fet}}{W_{min}/W} \\ &= n3R_nC_{fet} \\ &= n\tau \end{aligned}$$

which is the same result as obtained through the simplest reasonable *parasitic delay* approximation method: for na n-input gate, $p = n$.

4.3.2 Logical Power

In order to approximate the *active internal* coefficients, we need to get estimates of the parasitic capacitance for the gates. One way to do this is to re-extract from *parasitic delay:*

$$p = \frac{R_d C_p}{\tau}$$

Solving for C_p and equating to Equation 8 and dropping the *short circuit* term we find that

$$ac_{in} = C_p = \frac{p\tau}{R_d}$$

Solving for a and using the normalized equation for the *logical effort* parameter, Equation 12, we arrive at the very simple equation

$$a = \frac{p}{g} \tag{13}$$

When ignoring *short-circuit* power and miscellaneous intermediate parasitic capacitances, the above relationship provides a very simple approximation for our *active internal* coefficient *a* using only *Logical Effort* parameters.

Referencing Figure 15, it can be seen that for a switching A-input, that the intermediate parasitic capacitance, C_{p2}, remains discharged and should therefore negligibly contribute to active power dissipation, and Equation 13 represents an approximation for the A-input. However, when switching B-input, the intermediate capacitance is charged and discharged. Because in the delay approximation, delay is increased polynomially with the distance a capacitance is from the source of the signal, ignoring intermediate capacitances was justified because a differential amount of capacitance

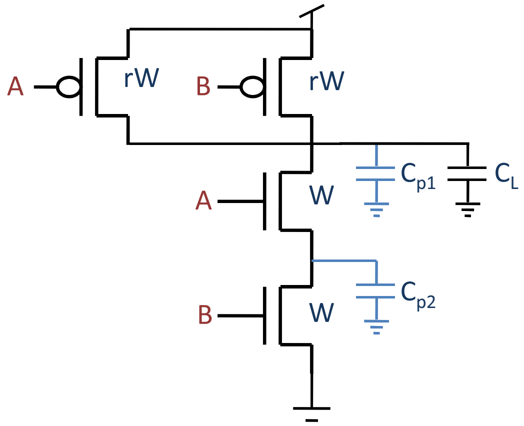


Figure 15. A two-input nand gate showing load capacitance, C_L , and various internal parasitic capacitances, C_{p1} and C_{p2}.

contributed less to delay the farther away it was from the end of the path of signal propagation. The energy consumed due to intermediate capacitances does not follow this trend, a differential amount of capacitance added anywhere will eventually get charged or discharged by the same amount regardless of location in the signal path and thus contributes equally to energy. Ignoring these intermediate nodes is not a good approximation in the energy case.

Instead of deriving *active internal* parameters from *Logical Effort* parameters, one could always be more thorough and re-extract these from the gate topology and include the internal parasitics. For the B-input of our 2-input NANDE gate this becomes

$$a_{nand2,b} = \frac{5}{2}$$

For an n-input NAND gate and an n-input NOR gate of equal rise and fall time with the assumed above parameters and a switching input of distance y in the transistor stack from the output ($y = 1$ for the closest input to the output), these become:

$$a_{nand} = \frac{2n(y-1) + 3n}{2+n} \tag{14}$$

and

$$a_{nor} = \frac{4n(y-1)+3n}{2n+1} \tag{15}$$

Unfortunately, there do not seem to be any estimation techniques of comparable simplicity for getting the *static internal* coefficients that are appropriate over varying technology nodes. This is due to different mechanisms of leakage being dominant in different processes, and will be explained in further detail in the following section.

4.4 Parameter Extraction

To extract the *Logical Power* and *Logical Effort* parameters, we need to be able to accurately measure the components of power and delay for a given logic gate versus its load capacitance. To do so, we employ the same characterization structure used to extract the parameters for LE.

Figure 16 shows the circuit used for gate characterization. It is simply the circuit suggested in chapter 5 of [8]. The circuit comprises multiple chains of gates, where each gate is the same gate to be characterized. The chains are five stages deep. In any chain each stage is subjected to the same *electrical effort* with varying *electrical efforts* between chains by increasing the number of branches at each stage.

A pulse is applied to the first stage and performance is measured on the third stage, the first two stages used to shape the input pulse to something reasonable. Integer values of *electrical effort* are achieved by loading a gate with multiples of itself, each load being loaded to reduce Miller capacitance effects on delay.

The parameters need to be extracted for each gate in order to use LP. In order to extract the parameters of *Logical Power,* we need a method to accurately measure the components of power for a given gate. To do so, we observe various currents associated with a gate, illustrated in Figure 27: I_{pd} the current entering the channels of the nFETs of the pull-down network connection to GND, I_b the current entering the

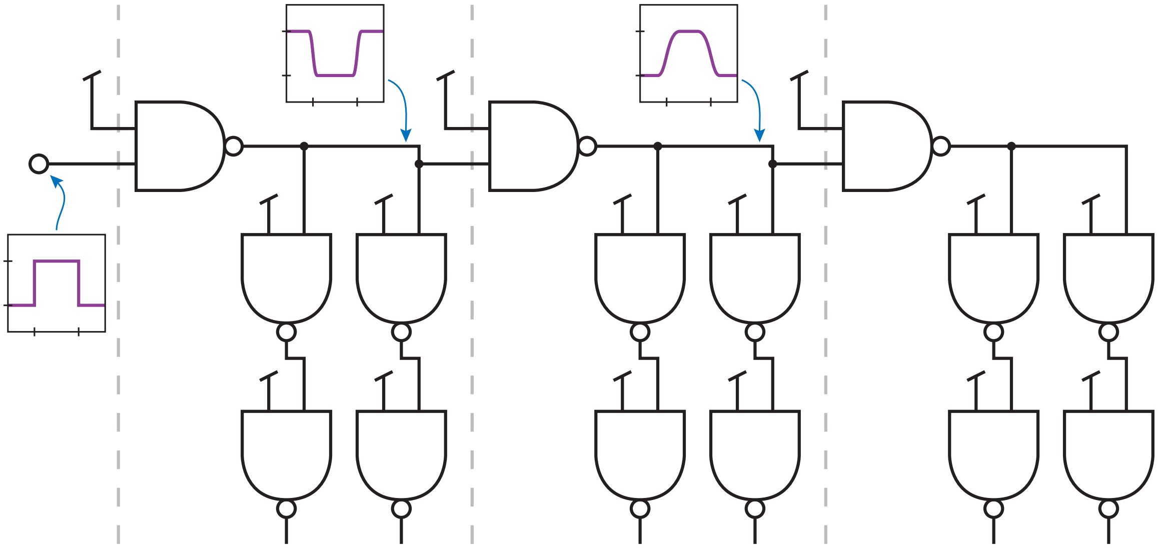


Figure 16. Three stages of the characterization circuit used for parameter extraction.

bulks of these devices, $I_{pu}$the current entering the channels of the pFETs in the pull-up network from V_{dd} , I_w the currents entering the wells of these devices, I_g the current entering the gates of the transitioning input, I_{bu} the current entering the gates of the non-transitioning inputs that are held high, I_{bd} the currents entering the gates of the non-transitioning inputs that are held low, and I_L the current entering the output of the gate. All currents are referenced as positive going into the gate, such that the sum of all of the above currents equals zero. The total currents coming out of the GND and VDD supplies is then:

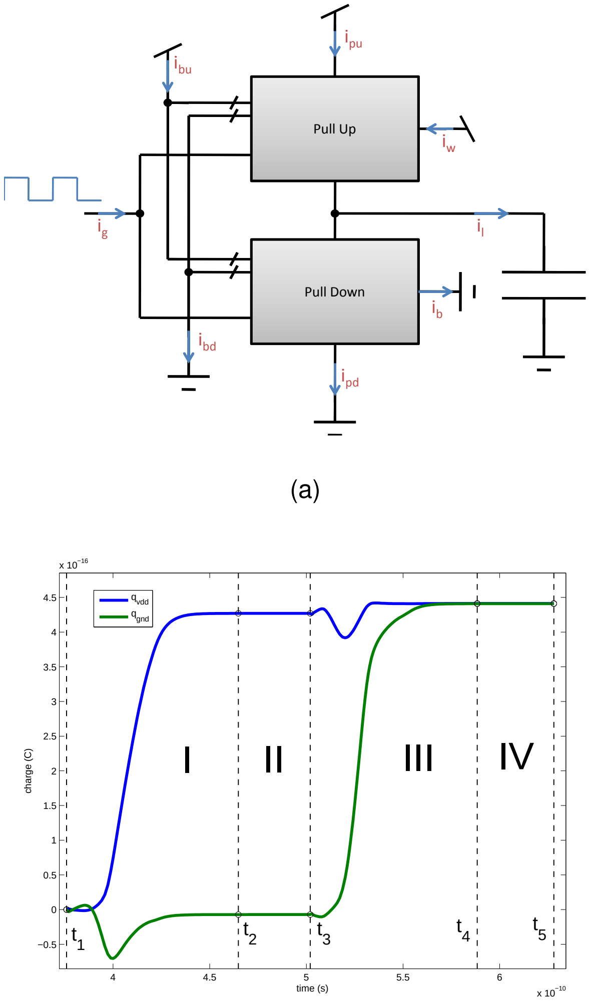
$$I_{GND} = I_b + I_{pd} + I_{bd}$$

and

$$I_{VDD} = I_w + I_{pu} + I_{bu}$$

The integrals of current are used to evaluate energy as

$$E = Vq = V \int idt \tag{16}$$



(a)

(b)

Figure 17. A) an arbitrary CMOS logic gate with pull up and pull down networks and corresponding currents, b) Integrals of current through the VDD and GND supply nets.

where the integral is taken over a time period that returns the gate to its starting state. This ensures that all flux that enters through the positive terminal of the voltage leaves through the negative terminal with no net flux accumulating in the circuit. That is, that all charge entering the circuit through V_{dd} leaves through GND. All power measurements are done through explicit integration of saved values of currents post simulation, as opposed to using power-meter subcircuits like the one proposed in [12] and used in [13].

Integrals of I_{VDD} and $-I_{GND}$ for a rising then falling transition for an arbitrary inverter in Figure 28. In this plot there are four distinct regions of interest. In Region I there is a significant amount of charge leaving through V_{DD} corresponding to a rising transition at the output of the gate. This current goes to charging capacitances and thus does not immediately show up through GND network of the gate. Region II is after the rising transition has settled and the gate has entered a static region, with Region III being the falling transition and Region IV the post falling static mode. One can see that after the gate has been restored to its original state does all charge that has entered the circuit leave the circuit.

Active energy is then extracted by integrating this current (either i_{vdd} or i_{gnd}, both being equivalent) over Region I and Region III

$$E_{act} = V_{dd}\left[\int_{t1}^{t2} i_{vdd}dt + \int_{t3}^{t4} i_{vdd}dt\right] \tag{17}$$

Effective input capacitance, c_{in} , is measured as

$$C_{in} = \frac{1}{V_{DD}}\int_{t_1}^{t_2} i_g dt \tag{18}$$

Assuming that after a rise transition that there is negligible voltage drop across the pull-up network, not all current coming out of V_{dd} gets dissipated in the target gate as some of this current leaves into the MOSFET gates of the load where it is then dissipated in the load gates. In order to avoid over-counting we need to consider only the leakage currents in the target gate that have an appreciable voltage drop in that gate. To do so, we measure different currents after different transitions. These transitions for a rise and fall respectively are

$$P_{stat,rise} = V_{DD}(-i_{pd} - i_b - i_g + i_{bu} - i_{bd})$$

and

$$P_{stat,fall} = V_{dd}(i_{pu} + i_w + i_g + i_{bu} - i_{bd})$$

with P_{stat} being the average of the two.

Figure 18 shows the input voltage and output voltage of a two-input NAND gate (switching the A-input) using the characterization circuit for various *electrical efforts* . The input slew rate changes as all stages in the characterization circuit have the same *electrical effort*. This choice tends to make *Logical Effort* more accurate in the cases where all stages have similar slew rates which is certainly the case for circuits optimized for maximum speed.

Rising and falling propagation delays are shown in Figure 19, which shows the linear behavior of propagation delay in this range of *electrical efforts*. From this rising, falling, and average *logical efforts* are extracted as

$$g = \frac{1}{\tau}\frac{d}{dh}d_{abs}$$

where τ is obtained from a reference inverter where $g \equiv 1$. The parasitic delay is

$$p = \frac{d_{abs}|_{h=0}}{\tau}$$

The total charge dissipated in the characterized gate flowing out of the V_{dd} and

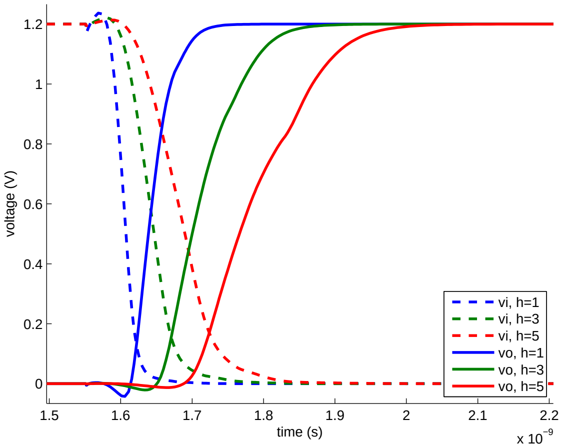


Figure 18. Input and output voltages versus time for input-a of a 2-input NAND gate for various h for a rising output transition.

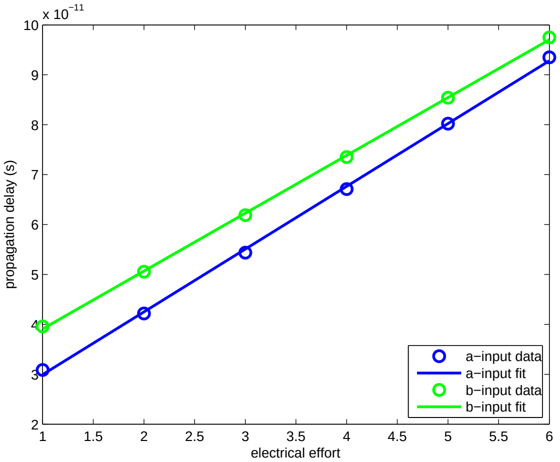


Figure 19. Average propagation delays versus h for for switching either input of a NAND2 gate.

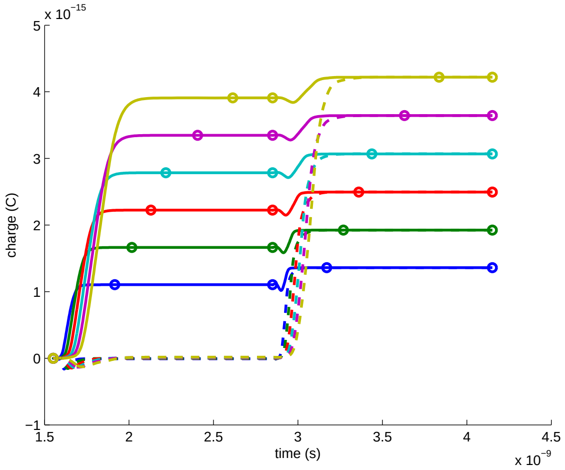


Figure 20. Integrals of i_{vdd} (solid lines) and i_{gnd} (dashed lines) versus time for various h for a 2-input NAND gate.

GND networks per time is shown in Figure 20 for various *electrical efforts*. The active energy parameter is then extracted from the active energy versus electrical effort plot by the following relationship:

$$a = \frac{E_{act}|_{h=0}}{V_{dd}^2 C_{in}}$$

The static current dissipation is dependent on the input vector to any complex gate. The static internal coefficient for each input for multiple gates is shown in Figure 22 as extracted from the following relationship:

$$s_i = \frac{c_{in,inv}}{P_{stat,inv}} \frac{d}{dc_{in,i}} P_{stat,i}$$

This figure shows that the relationship of of how the static internal coefficient changes with input number is very process dependent. The big difference here is that in the $130nm$ and $65nm$ LP models used, gate current is insignificant and not modeled. Because of this, simple relationships can be made for estimating how static internal coefficients will change with gate topology, however, these relationships are going to change depending on the presence of significant gate current or not.

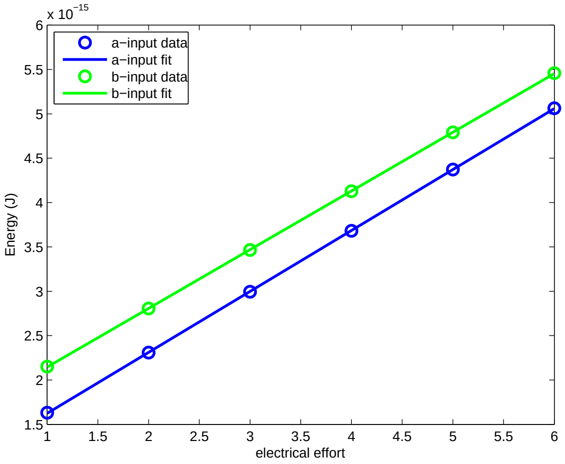


Figure 21. ***Active Energy*** **versus h for switching either input of a NAND2 gate. Since the B-input is farther away from the output, it has to charge and discharge extra parasitic capacitance for this transition.**

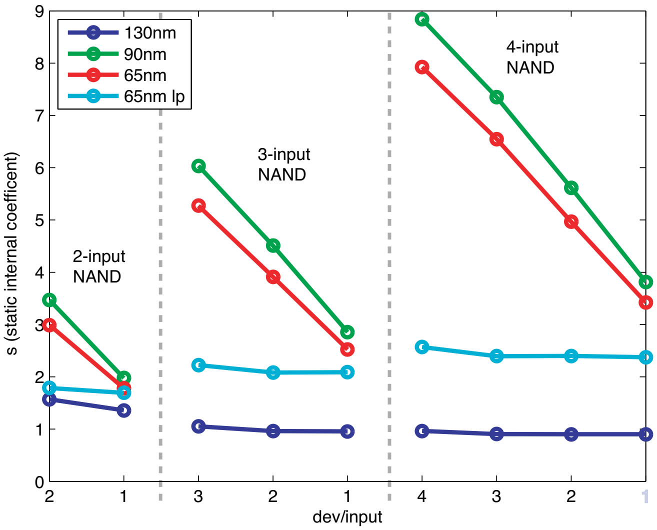


Figure 22. Leakage current for a 2-input, 3-input, and 4-input nand gate after a rise or fall transition. The x-axis corresponds to different sized nand gates and different inputs within the nand gates, from left to right: y=2 down to 1 for a 2-nand, y=3 down to 1 3-nand, y=4 down to 1 4-nand.

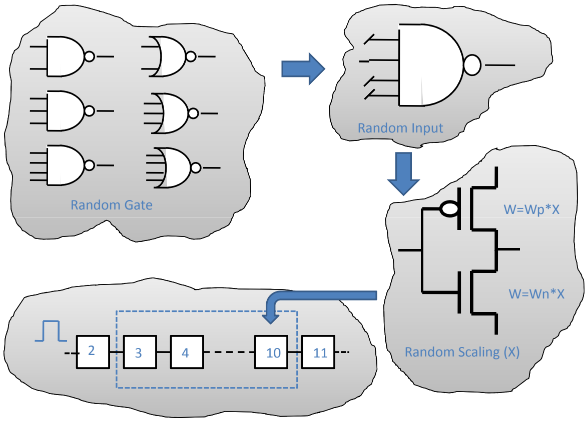
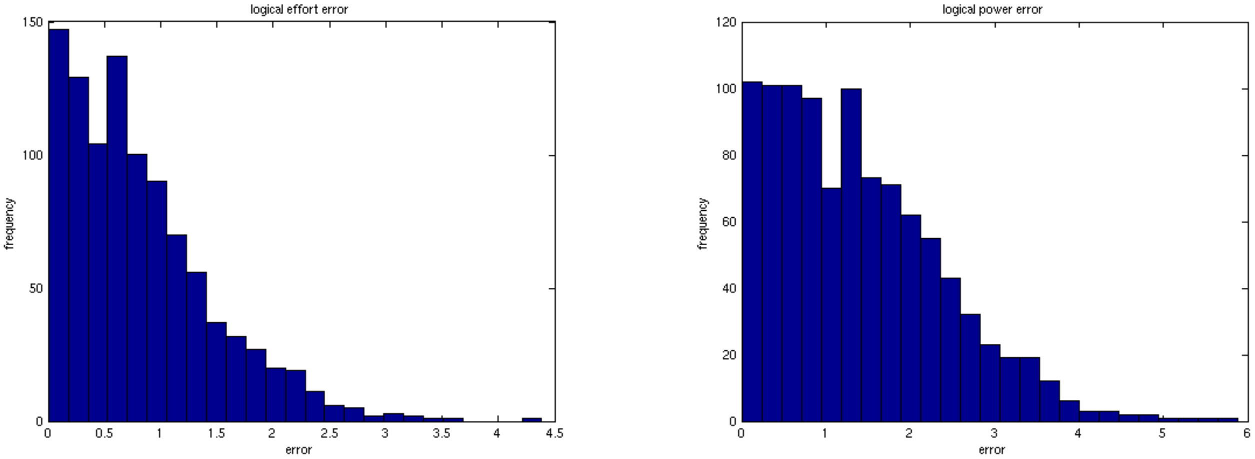


Figure 23. Randomly generated test circuit. A pool of 2 to 4 input NANDs and NORs and inverters of random sizes (1 to 8x) are chosen to comprise the 12 stage circuit. Propagation delay and energy consumption of the 3rd to 10th stage are measured in simulation and predicted with LE and LP.

4.5 Statistical Path Analysis

The accuracy of the *logical power* method was verified through a test circuit of random gate types. The test circuit was a 12 stage logic path composed of two to four input NANDs and NORs as well as inverters all scaled to random sizes between 1 and 8x minimum size, as illustrated in Figure 23. This structure of test circuit was chosen because each gate has an activity factor of $\alpha_i = 1$. The most trivial of circuits have this property, and the individual activity factors are dependent on circuit topology and input vector. This circuit was chosen to evaluate the effectiveness of *logical power* in the limit of perfect knowledge of activity factor, as the method itself is independent of whatever system one uses to predict an activity factor.

Propagation delay and total power was measured from the 3rd through the 10th stage, as well as predicted by LP and LE and the results were compared. Out of 1000 randomly generated circuits, LE had an average error of 0.8% while *logical power* an error of 1.6% with a worst case of 5%. Histograms of these data are shown in Figures 24b and 24a.



(a)

(b)

Figure 24. a) LE Error mean 0.8%, b) Logical Power error mean 1.6%

Table 1. Statistical results for Logical Power

Tech	Mean Error (%)	Max Error (%)	Active (%)	Static (%)
130nm	0.7	4.5	99	1
90nm	5.3	10.2	84	16
65nm LP	10.4	18.4	99	1
65nm SF	6.0	9.5	77	23

Table 1 shows this same experiment for various processes from 65nm to 130nm as well as a breakdown for each process what percent of total power was *active power*. The best result was a 0.7% mean error from the 130nm process and the worst was a 10% mean error from the 65nm low power process.

4.6 Hand Optimization of Inverter Chain

Unfortunately, there appears to be no analytical solution to the non-linear programming problem for gate sizings to minimize energy as specified by the *Logical Power* Equation 11 under the constraint of meeting a target delay specified by the *Logical Effort* Equation 5. These equations can be solved numerically by a method that will be referred to as the optimal sizing method. In this section, we present an approximation to this solution for the optimization of inverter chains.

Consider an inverter chain of depth N with input driving capacitance $c_1 = c_{inv}$ and load $c_{N+1} = xc_{inv}$, in order to calcualte the Pareto points in the energy delay space, we start with the *LE* optimized delay as the minimum delay, maximum energy point (any solution with higher energy than this would use different sizings, and thus be slower), and then increase this delay with a method that decreases energy. We choose to increase the delay of the final stage by decreasing its input capacitance, while keeping all intermediate stage delays the same, which we refer to as the constant intermediate delay method. This method fixes *electrical effort* (through subseqent decreases in their input capacitances in order to maintain constant *electrical effort*) up until the first stage, where delay must decrease.

To gain insight into this method, consider the differential amount of gate stage energy saved when that stage's delay is increased by decreasing its input capacitance, which is given by

$$\frac{dE_i}{df_i} = -(b_{i+1} + a_i)g_i c_{i+1} f_i^{-2} \tag{19}$$

This is obtained by plugging the equation for *effort delay* into Equation 9, which describes energy consumed by the entire path due to the i^{th} stage's input capacitance. Only *effort delay* is considered as parasitic delay remains invariant under sizing choices. This equation assumes that *static power* is negligible, which is true only for $d << d_s$.

It is clear that if a differential amount of delay is to be introduced to any one stage, that for for the case of an optimally sized inverter chain with monotonically increasing stage capacitances (always true for $H > 1$) that the greatest amount of energy savings per incremental delay is obtained when this delay is introduced into the last stage. Keeping in mind that we desire to introduce delay only to the N^{th} stage, but a decrease in C_N will lead to a decrease in h_{N-1} which in order to return $f_i = \hat{f}$, c_{N-1} will also have to decrease. Which in order to keep f_2 through f_{N-1} all equal to $\hat{f}$, all intermediate stages will have to have their input capacitances reduced leading to even further energy savings. While this method does not exactly minimize energy as a function of delay, it does come very close and has an analytical solution for the required stage sizings.

Starting with the sizings necessary for minimum delay, we obtain an optimal stage *effort delay* of $\hat{f}$, such that all stage efforts $f_i = \hat{f}$:

$$f_i = \hat{f} = g_i \frac{b_i c_{i+1}}{c_i}$$

and

$$d_{min} = N\hat{f} + P$$

We then increase the delay of the last stage by decreasing C_N from its optimal, while all middle stages are sized such that their *effort delays* remain unchanged, this is propagated back to the initial stage whose delay must naturally increase a big as c_2 decreases without c_1 decreasing:

$$d = g_1 b_1 \frac{c_2}{c_1} + (N-2)\hat{f} + g_N b_N \frac{c_L}{c_N} + P \tag{20}$$

Which considering the target application of an inverter chain, this reduces to

$$d = \frac{c_2}{c_{inv}} + (N-2)x^{\frac{1}{N}} + \frac{x c_{inv}}{c_N} + N \tag{21}$$

c_2 and c_N are not independent, but are related through recursive application of Equation 6 to the intermediate stages:

$$c_N = c_2 \frac{\hat{f}^{N-2}}{\prod_{i=2}^{N} g_i b_i}$$

Which for an inverter chain reduces to, keeping in mind that $\hat{f}$ simplifies to $x^{1/N}$ and that all parameters like *logical effort* and *branching effort* are one in this case,

$$c_N = c_2 x^{1-\frac{2}{N}}$$

Plugging this into Equation 21 and applying the quadratic formula, c_2 can be solved for for any desired delay as:

$$\begin{aligned} c_2 &= \frac{c_{inv}}{2}\Big\{d - N - (N-2)x^{\frac{1}{N}} && (22) \\ &\pm \sqrt{[(N-2)x^{\frac{1}{N}} - d + N]^2 - 4x^{\frac{2}{N}}}\Big\} && (23) \end{aligned}$$

This method is applied to a four stage buffer chain of inverters driving a fixed load with fixed 1st stage size. Figure 25a shows stage *effort delays* versus target delay for sizings picked by Equation 22 , the constant intermediate stage delay method. The middle stages stay at a fixed *effort delay* of $\hat{f}$ until sizings hit minimum sized. Figure 25b shows the same buf for sizings picked by the optimal sizing method. Figure 25c shows the energy versus delay curves of the two different methods. Each point on the curve represents a different cycle energy and cycle delay time obtained different gate sizings. The constant intermediate delay method comes very close to the performance obtained by the optimal method, the error in energy between the two methods is shown in Figure 25d , and in this scenario it is shown that it produces energies around 1% of optimal. The difference is due to the incremental savings in capacitance when delay is allowed to increase. Since energy savings decrease with $1/f^2$, the optimal method distributes marginal portions of the increase in delay to many stages and is able to save a little more energy than the constant intermediate delay method.

The optimal sizing method is applied to various stage length inverter chains driving a fixed load with $F = 250$, and the energy delay curves are shown in Figure 26 . Stage lengths of 2, 3 and 4 are shown. For this particular *path effort* an appropriately sized $N = 4$ chain minimizes delay, with more or less stages leading to an increase in minimum obtainable speed. Interestingly at speeds less than minimum, depending

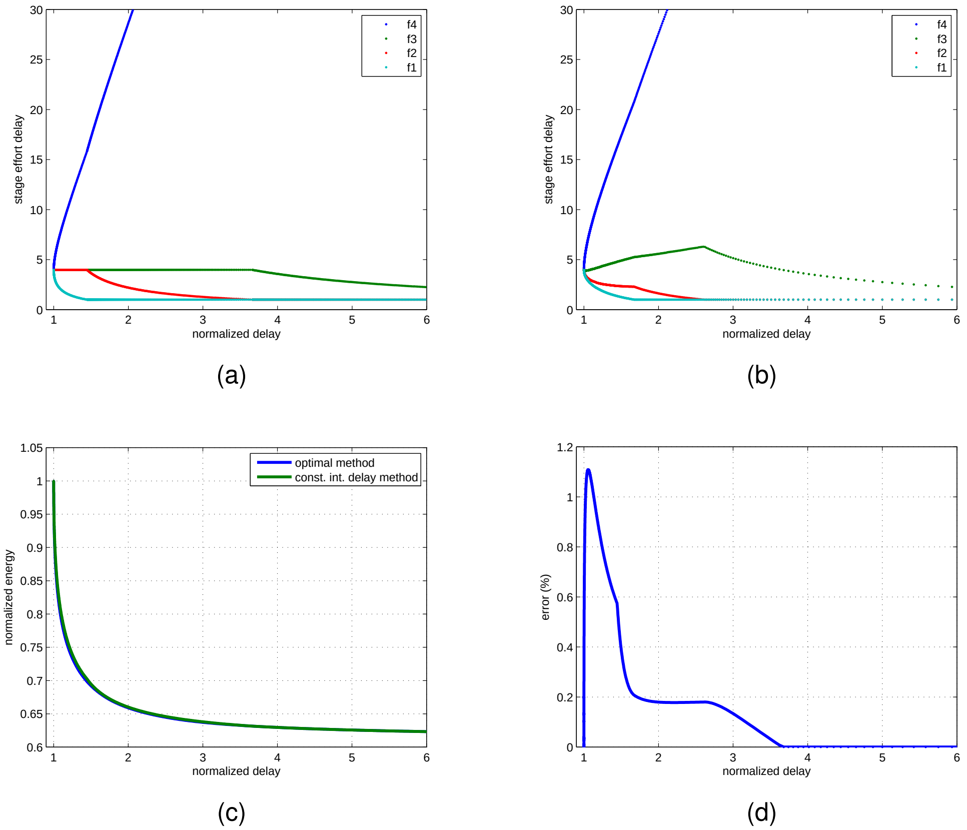


Figure 25. Stage *effort delays* of a four stage inverter chain ($F = 250$) versus target delay. a) Sizings picked via Equation 22 . The middle stages stay at fixed *effort delay* of $\hat{f}$ until sizings hit minimum sized. b) Sizings picked by numerically solving Equation 5 and Equation 11 c) the energy-delay curves obtained by both methods, and d) the error in the approximation method (a) from ideal (b).

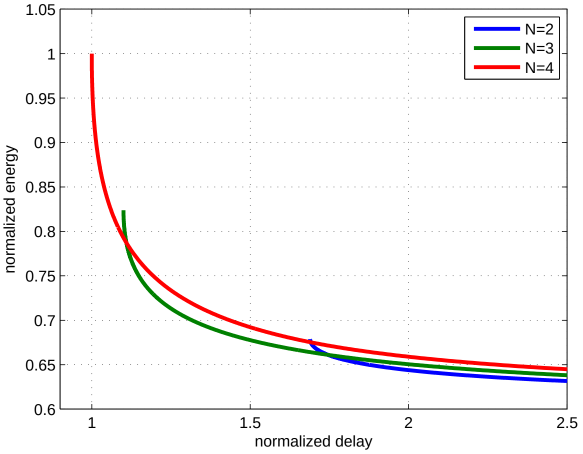


Figure 26. Energy-delay points obtained by different gate stage sizing choices of various inverter chains of different depths driving a fixed load with path effort F=250. All points shown are pareto-points in the sizing space in terms of energy and delay for their respective chains. N=4 produces the fasted possible speed for this particular path effort, but at speeds less than minimum, depending on the actual target delay, energy is minimized by choosing not only the right sizing options but the right number of stages. In particular, there are some scenarios in which a higher number of stages can minmize energy for a target delay obtainable with less stages.

on the actual target delay, energy is minimized by choosing not only the right sizing options but the right number of stages. In particular, there are some scenarios in which a higher number of stages can minmize energy for a target delay obtainable with fewer stages.

4.7 Conclusion

The method of *Logical Effort* is extended to allow for energy delay tradeoffs by *Logical Power*. This new method requires two new gate dependent parameters, *active internal* and *static internal* , and uses gate sizing as the only variable, ignoring all sorts of higher order effects including input slew rate, and is intended for hand analysis for gate circuit topologies and logic paths. Nevertheless, it is hown to be accurate to within 10% error when compared to detailed SPICE simulations across many technologies, and is usually much more accurate than that.

A hand analysis method for approximating one of the new parameters, *active internal* , is presented using many of the same approximations that *Logical Effort* uses to estimate its gate dependent parameters. This method ends up being about as accurate as approximation methods for estimating *logical effort* and is significantly better than those used to approximate *parasitic delay*. Approximating the *static internal* coefficient is difficult as different leakage mechanisms dominate in different technologies.

A method is suggested that approximates the optimal sizing problem for minimizing energy under constrained delay. This method solves analytically for the stage sizes and produces energy delay tradeoffs within 1.2% of optimal in the test case it was applied to.

CHAPTER 5

MEASURING SHORT-CIRCUIT POWER

In this chapter, A new method for accurately measuring short-circuit power dissipation of simulated CMOS digital gates is presented. Previous methods have had difficulty in differentiating between dynamic and short-circuit currents in the “turning off” network during an active transition. This new method naturally converges on zero short-circuit energy for infinitely fast switching inputs and outputs and never produces negative answers for short-circuit energy- things previous methods have had difficulty with.

5.1 Components of Power

Power dissipated in a logic gate is a complicated function of the gate’s implementation and its environment. The logic family, transistor topology, transistor sizings, and rail voltages that all define a gate are only part of the picture. The load that the gate drives, and transient inputs all factor into the power a gate will dissipate over time.

The power dissipated by the gate can be broken down into three well defined components: *dynamic power, static power,* and *short-circuit power.* Dynamic power is simply defined as the amount of power consumed by charging and discharging capacitances seen by the logic gate and can be expressed as:

$$P_{dyn} = \alpha f V_{dd}^2 C_{tot} \tag{24}$$

Where $\alpha$is the activity-factor defining the fraction of the cycles that the logic gate makes an output transition, f the cycle frequency, $V_{dd}$the rail-to-rail voltage, and $C_{tot}$the total amount of capacitance seen by the driving gate.

Whereas *dynamic power* is dissipated during an output transition, *static power* is the power consumed when the gate is not making a transition and is a simple function of the static current draw I_{stat}:

$$P_{stat} = V_{dd} I_{stat} \tag{25}$$

The last portion of power, *short-circuit* power*,* is defined as the portion of power consumed during an output transition that did not go to charging capacitances. This portion of power is quite difficult to express compactly, and for even the most trivial of logic gates, an inverter, has no closed-form analytical solution. Most approximations are extremely cumbersome. *Short-circuit power*, is a function of input slew rate, output slew rate, and gate topology.

5.2 Measuring Active Power

In order to extract the parameters of *Logical Power,* we need a method to accurately measure the components of power for a given gate. To do so we observe various currents associated with a gate, illustrated in figure 27:

$$I_{GND} = I_b + I_{pd} + I_{bd} \tag{26}$$

and

$$I_{VDD} = I_w + I_{pu} + I_{bu} \tag{27}$$

where I_{GND} is the total amount of current supplied by the GND rail, whose components are I_b, I_{pd}, I_{bd} being the currents leaving the bulk, pull down network, and any logic level zero bias applied to the circuit respectively. I_{VDD}being the V_{dd} rail equivalent.

The integrals of current are used to evaluate energy in:

$$E = Vq = V \int idt \tag{28}$$

where the integral is taken over a time period that returns the gate to its starting state. This ensures that all flux that enters through the positive terminal of the voltage leaves

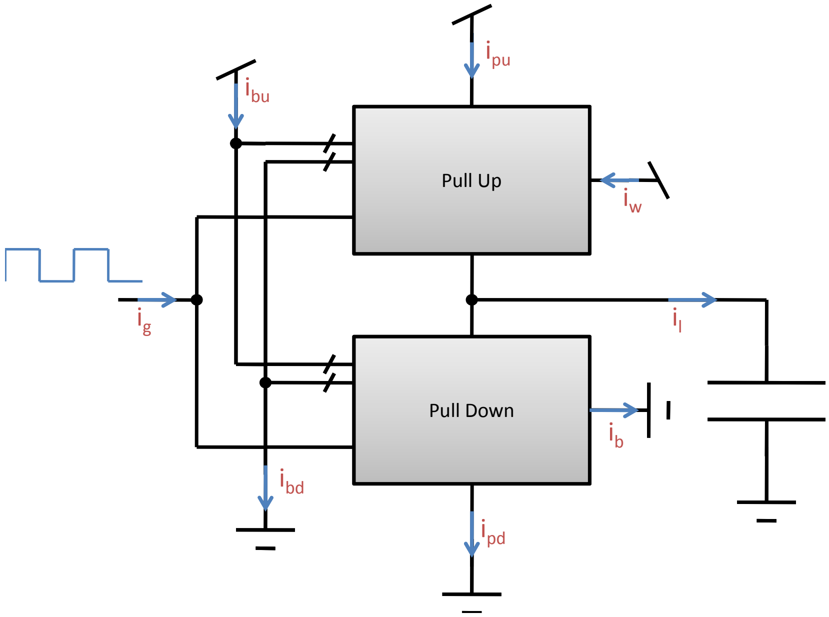


Figure 27. An arbitrary CMOS logic gate with pull up and pull down networks and corresponding currents.

through the negative terminal with no net flux accumulating in the circuit. That is, that all charge entering the circuit through V_{dd} leaves through GND. All power measurements are done through explicit integration of saved values of currents post simulation, as opposed to using power-meter sub circuits like the one proposed in [12] and used in [13].

Integrals of I_{VDD} and I_{GND} for a rising then falling transition for an arbitrary inverter in Figure 28. In this plot there are four distinct regions of interest. In Region I there is a significant amount of charge leaving through V_{DD} corresponding to a rising transition at the output of the gate. This current goes to charging capacitances and thus does not immediately show up through GND network of the gate. Region II is after the rising transition has settled and the gate has entered a static region, with Region III being the falling transition and Region IV the post falling static mode. One can see that after the gate has been restored to its original state does all charge that has entered the circuit

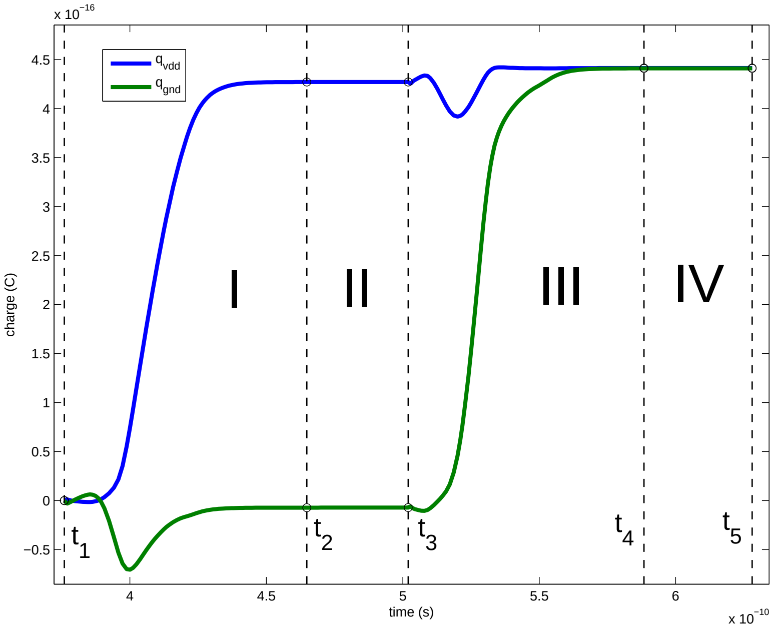


Figure 28. Integrals of current through the V_{dd} and GND rails

leaves the circuit.

Active energy is then extracted by integrating this current (either I_{VDD}or I_{GND}, both being equivalent) over Region I and Region III:

$$E_{act} = V_{DD}\left[\int_{t1}^{t2} I_{VDD}dt + \int_{t3}^{t4} I_{GND}dt\right] \tag{29}$$

In order to break active power down into its components, *short-circuit power*, and *dynamic power*, we need to make some choices on how to handle an issue known as voltage overshoot.

5.3 Voltage Overshoot and Miller Effect

The transient output voltage of a CMOS gate can go above the V_{DD} and below the GND for that particular gate during transitions as can be seen in Figure 29. This phenomena, called voltage overshoot, is due to capacitive coupling of a gate's input to its output by the gate to drain capacitances of the transistors.

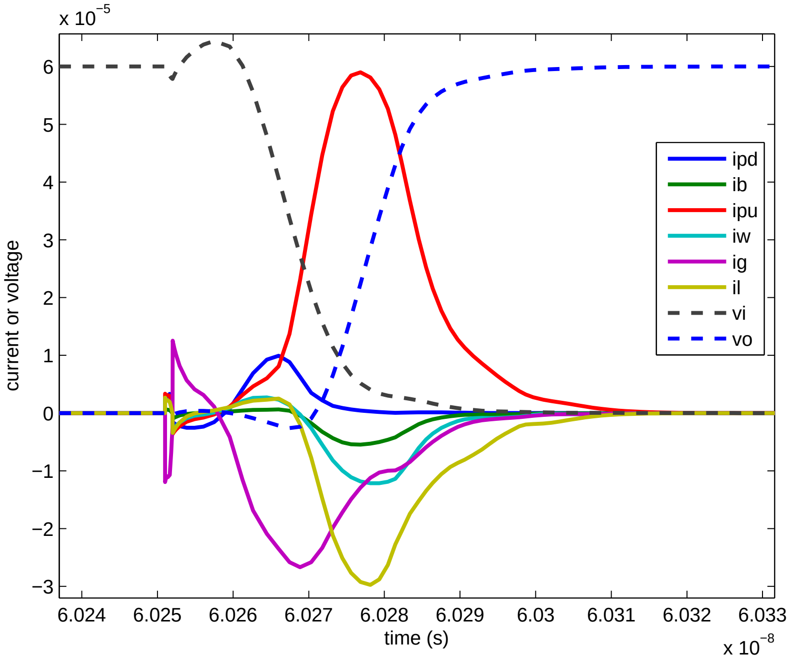


Figure 29. Transient voltages and currents during a rising output transition showing voltage undershoot at the output as well as the subsequent negative current flowing into GND.

Consider a falling output transition for a logic gate with the input voltage initially at GND. The output voltage is initially settled to V_{DD} when the input voltage starts to rise. If the capacitive coupling between the input and output is called C_M and the total capacitance seen at the output is C_L, then for instantaneous changes in the input voltage equal to $\triangle V_{in}$ the output voltage will go to

$$V_{out} = \triangle V_{in} \frac{C_M}{C_L} + V_{DD} \tag{30}$$

The magnitude of the voltage overshoot depends on the input slew rate and the speed by which the output can discharge. During the voltage overshoot the normal polarity for V_{DS} of the pull up network reverses and the output is helped to discharge through the pull up network with a positive current flowing into V_{DD}. Power is dissipated in the pull up network due to this that is not short-circuit power, in fact, this phenomena seems to reduce short-circuit power. Total power consumption is not increased by this effect, it just moves a fraction of the driving gate's dynamic power to be dissipated in the load gate.

To see this, consider the analongous circuit in Figure 30. The values of R_{U1} , R_{U2}, C_m's, V_{DD1} and V_{DD2} from both circuits are equivalent to each other. In the first circuit R_{U1} models the pull up or pull down network resistance of the driving gate, with the switching between the two networks being modeled by the voltage source V_{DD1} pulsing between V_{DD} and $0V$. The capacitance C_m models the input capacitance to the load gate with $C_m + C_L$ being the total capacitance seen on the output of the load gate. The output voltage of the load gate is V_{o1}, this is where the voltage overshoot will occur, and R_1 models an arbitrary discharging path for this overshoot (after a rising or falling input, the output is always discharged to ground, which is not the case for CMOS gates, nonetheless this circuit is sufficient to show the trends on how output voltage swing can affect power and delay). The second circuit models this effect for when overshoot is reduced to insignificance by either C_L or R_1 being sufficiently large or small with respect to the other circuit elements. The transient voltage behavior of these circuits is shown in Figure 31 showing both the voltage overshoot above V_{DD} during the rising input transition and below GND during the falling input.

For the second circuit under consideration, inspection of energy suggests that when V_{DD2} increases to V_{DD}, the voltage V_{i2} eventually charges to V_{DD} storing an amount of energy on C_m equal to $\frac{1}{2}C_m V_{DD}^2$ with an equivalent amount of energy having been dissipated in R_{U2}. When V_{DD2} is brought back down to $0V$, V_{i2} eventually reduces to $0V$ with all of the potential energy stored in C_m discharged through R_{U2} . Total energy dissipation of both transitions being $C_m V_{DD}^2$ and all of it dissipated in R_{U2}.

While the first circuit is much harder to solve out exactly, one can see that the final amount of energy stored in the circuit upon settling after V_{DD1} goes to V_{DD} is the same as in the second circuit, $\frac{1}{2}C_m V_{DD}^2$. It turns out that exactly this amount of energy is dissipated in the circuit's resistors during charging and dissipated again during discharging.

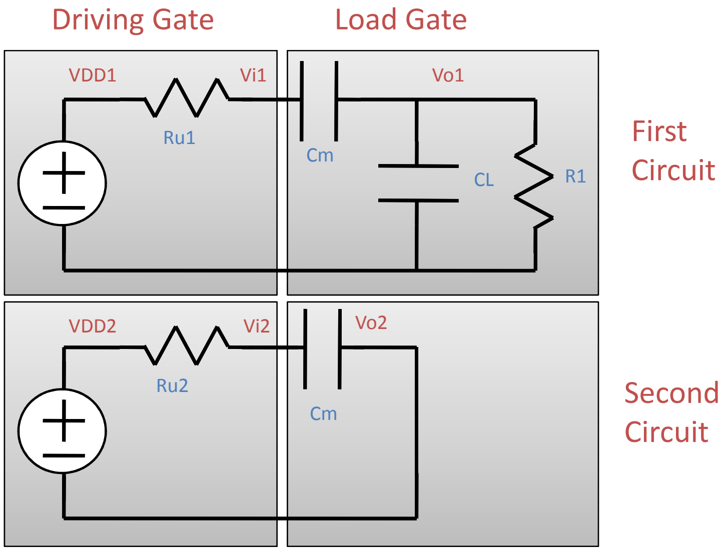


Figure 30. Example circuits for demonstrating the principles of voltage overshoot in CMOS gates. In the second circuit, V_{i2} corresponds to the input voltage to a load gate with no capacitive coupling to the load gate's output, whereas in the first circuit the input voltage V_{i1} is coupled to the load gate's output voltage V_{o1}.

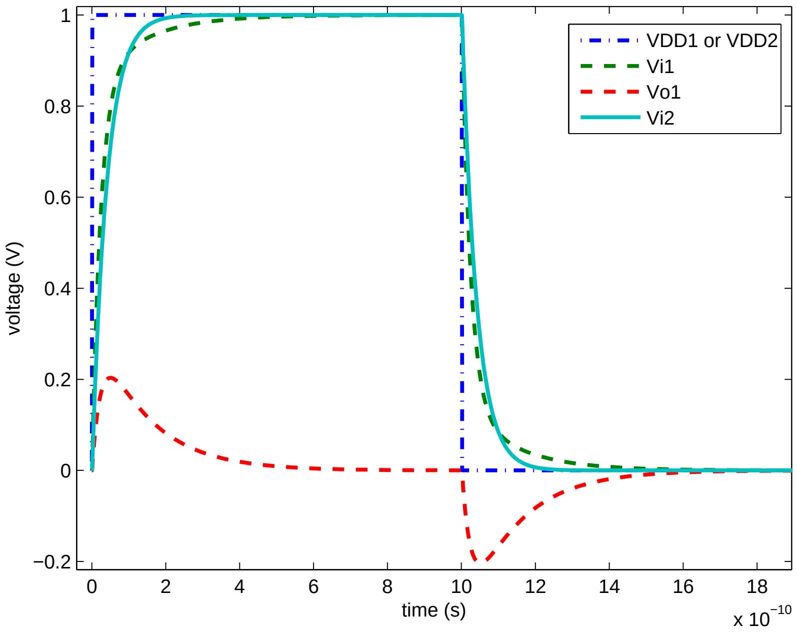


Figure 31. Transient voltages for both circuits under consideration.

That is, total energy consumption of both transitions is the same as the second circuit, but now the energy is dissipated across two resistors, one being the analog of the load gate. The total energy dissipation of both circuits as well as the dissipation in each resistor is shown in Figure 32.

The analogy then suggests that voltage overshoot does not increase dynamic power dissipation above what is already incurred due to load capacitance, as in both cases the energy dissipation was defined by C_m only. The phenomena only moves a fraction of the energy dissipation into the resistances of the load gate. The magnitude of the fraction being related to the magnitude of the voltage overshoot.

Even though this produces no effect on the total dynamic power, it can have a significant effect on delay. To observe this, notice that the delay to 50% rail-to-rail of V_{i1} is less than that of V_{i2} as can be seen in Figure 33 . This is because in the first circuit, over this range of observation, the output voltage tracks the input voltage, effectively reducing the voltage across the input capacitance C_m and reducing the amount of charge

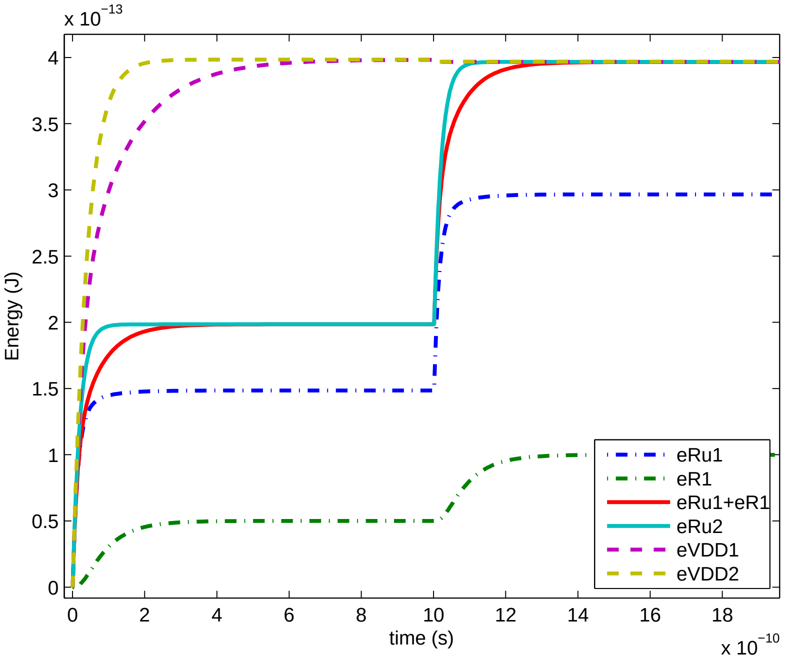


Figure 32. Energy dissipated by resistances and sourced by voltage sources as a function of time for the two different circuits for a rise and a fall. As time tends towards infinity, the energy dissipated in both circuits is identical.

necessary for a given change in input voltage. Whereas, in the second circuit the output voltage is fixed. The net effect being that the delay over this range is less for the first circuit.

This effect on delay can be explained by the Miller Effect on the effective input capacitance, $C_{effective}$, looking into the load:

$$C_{effective} = C_m(1 - A_v) \tag{31}$$

where C_m is the capacitance coupling the input to the output, and A_v is the gain from the input to the output. In the example of circuit one, the gain is positive and less than one over this range, and thus the effective capacitance is less than the case of the second circuit with zero gain. For times larger than this range, where the output voltage stops tracking the input voltage, the effective gain becomes negative, and an increase in effective capacitance is observed. Therefore, the effect of the Miller Effect

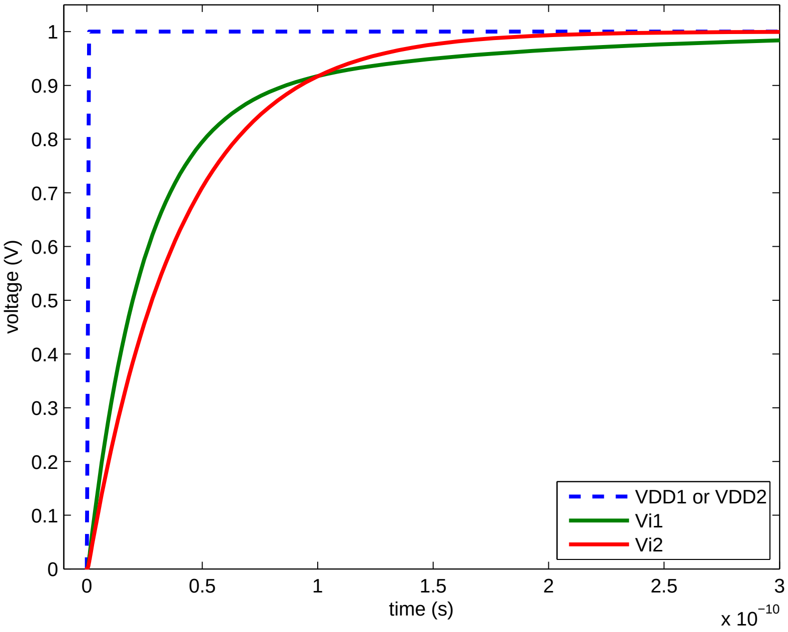


Figure 33. The Miller Effect on delay.

on the delay of the driving gate depends highly on the output slew rate of the load gate, and therefore on the load seen by the load gate.

For standard CMOS circuits, depending on the load gate's load size, the Miller Effect can either increase or decrease propagation delay. For reasonable load sizes, like shown in Figure 29, the net effect is trivial. However, if the load gate was not loaded, the output voltage would swing much faster, and could cause a significant overlap of V_{out} and V_{in} which would increase the gain over this region and thus the effective input capacitance, which in turn would increase delay.

This means that in the characterization of a logic gate for power, one can safely ignore the effects of the Miller Effect and voltage overshoot, whereas this is not quite the case for delay characterization.

5.4 Measuring Short-Circuit Power

Short-Circuit power contributes an additional energy cost to active gate transitions. It arises from increased conductivity between V_{DD} and GND during an output transition due to both pull up and pull down networks being in between their high and low impedance states. Accurately measuring short-circuit power dissipation in simulation is complicated by capacitive currents flowing in the turning-off network during transition.

Most methods for measuring short-circuit power, the methods used in [11, 14, 13] for instance, involve integrating the supply current flowing through the turning-off network during a transition and assuming that all of this charge contributes to short-circuit power dissipation. The total short-circuit energy is then

$$E_{sc} = V_{dd}\left[\int_{t1}^{t2} i_{gnd}dt + \int_{t3}^{t4} i_{vdd}dt\right] \tag{32}$$

where the first integral is over a rising output transition, and the second integral over a falling.

The problem with this assumption is that that simply is not the case. One of the contributions of this thesis is to show that a rather significant portion of this current is due to capacitive charging and discharging in the turning-off network that should be attributed to dynamic power and not short-circuit power, as well as to propose and analyze a method of measuring short-circuit power that fixes this problem.

Figure 34 shows the major components of a MOSFET as modeled by the EKV v2.6 device model [15]. For this section we are only concerned with I_{DS}, the DC channel current, and the total effective terminal-to-terminal capacitances C_{GS}, C_{GD}, C_{SB}, C_{DB} and we ignore the gate to bulk capacitance and various diodes and other current sources. Figure 35 shows an inverter with these MOSFET capacitances explicitly drawn.

Consider a rising output transition. Prior to the transition the pull down network provided a low impedance path to GND, the pull up network a high impedance path to

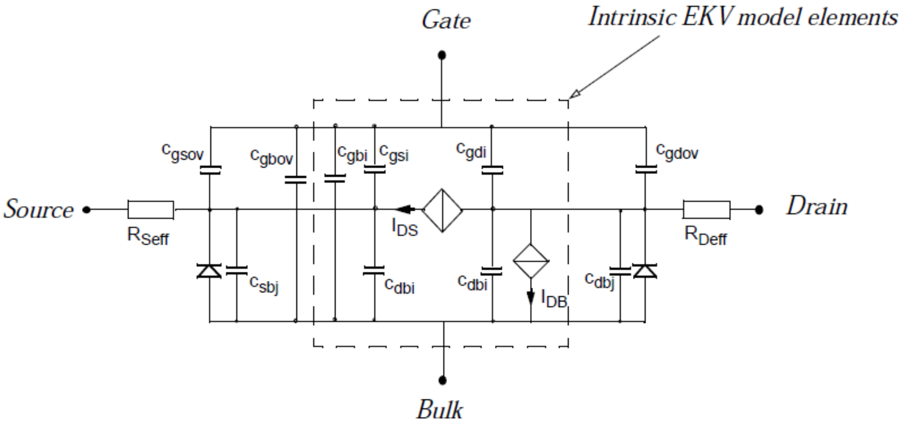


Figure 34. The intrinsic and extrinsic elements of a MOSFET modeled in the EKV v2.6 device model. It is the portion of the Source and Drain terminal currents not coming from capacitances that should be used for calculating short-circuit power.

V_{DD}. After the transition these paths are swapped and a strong path to V_{DD} is connected to the output while the path to GND is weakened. During the transition, however, both networks enter intermediate states that can allow for relatively strong connections from V_{DD} and GND to be connected to the output, providing a low impedance path between both voltage rails. That causes current to flow through both networks creating short-circuit power dissipation.

Let i_{pd} be the current going into GND from the pull down network minus the bulk current, i_b. Define q_{pd} and q_b to be the integrals of current with respect to time over the rising transition, and q_{pu} and q_w to be integrated over the falling transition. If $q_{sc-rise}$ and $q_{sc-fall}$ are the total amounts of charge flowing through the gate due to short circuit during a rise and fall transition respectively, then the total energy dissipation of a rise and fall transition due to short-circuit power is

$$E_{sc} = V_{DD}(q_{sc-rise} + q_{sc-fall}) \tag{33}$$

Considering the rising output transition still, the problem is to relate $q_{sc-rise}$ to q_{pd} and q_b. However, both of these quantities contain charge that flowed onto and off of transistor capacitances. For instance, a large portion of q_{pd} is due to the negative

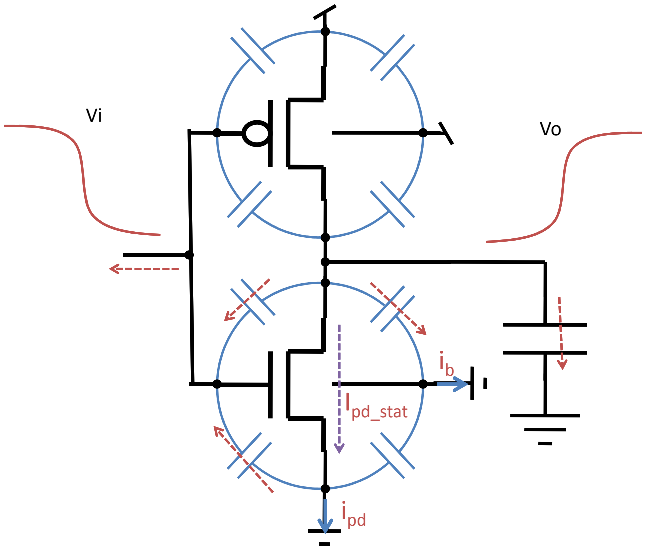


Figure 35. Differentiating between capacitive and static currents during a rising output transition.

current flowing out of GND from the discharging of the gate to source capacitance of the nFET by the input transitioning low, as can be seen in Figure 35 . Capacitive feed through from the input to the output causes the output to go below GND for a small portion of time creating a negative V_{DS} on the nFET and causing current to flow from GND to the output node. Neither of these types of contributions to the current should be considered as short-circuit power as they do not fit the definition of conduction from one rail to the other.

In [16] the authors attempt to pull these currents out of the problem by integrating the total positive current flowing into GND for a step input. This value is

$$E_0 = V_{dd}\left[\int_{t1}^{t2} f(i_{gnd})dt + \int_{t3}^{t4} f(i_{vdd})dt\right] \tag{34}$$

where $f(i)$ is the unit ramp function

$$f(i) = \begin{cases} i & i > 0 \\ 0 & i \leq 0 \end{cases} \tag{35}$$

E_0 is then subtracted out of out of subsequent measurements of E_{SC}

$$E_{sc} = V_{dd}\left[\int_{t1}^{t2} f(i_{gnd})dt + \int_{t3}^{t4} f(i_{vdd})dt\right] - E_0 \tag{36}$$

This method forces the measurement of $E_{sc}$to converge on zero for infinitely fast input transitions.

This work proposes a more direct measurement of short-circuit power. Considering the MOSFET model shown in Figure 34 there is a total amount of current flowing into the source or drain that is due to MOSFET capacitances and a portion due to the DC channel current i_{DS}. By integrating the positive portion of this i_{DS} of the switching transistor in the turning off network we directly measure total short-circuit charge without having to manually deal with dynamic and feed through charge effects. Measurements done in this manner inherently converge on zero (neglecting subthreshold leakage currents, etc.) for increasingly fast input transitions.

$$E_{sc} = V_{dd}\left[\int_{t1}^{t2} f(i_{pd-stat})dt + \int_{t3}^{t4} f(i_{pu-stat})dt\right] \tag{37}$$

Where $i_{pd-stat}$ is the static current coming out of the source of the switching transistor in the pull down network, $i_{pu-stat}$ the static current entering the source of the switching transistor in the pull up network, and $f()$ is the same unit ramp function as before.

Being able to differentiate between capacitive and static currents depends on whether or not the simulator provides access and whether or not the model keeps track. For instance, in the SPECTRE circuit simulator, the static and capacitive currents going into the source of a transistor, $M0$, can be accessed by using:

```
save M0:s:static M0:s:displacement
```

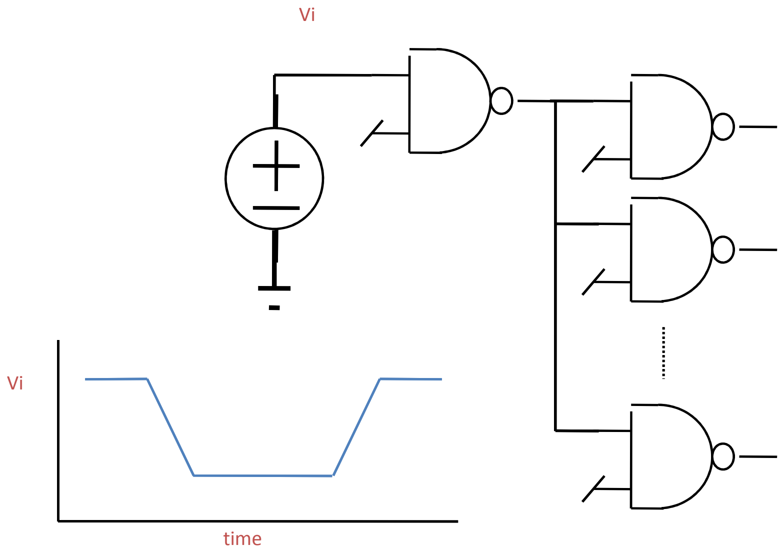


Figure 36. Test circuit with variable input slew rate and *electrical effort*.

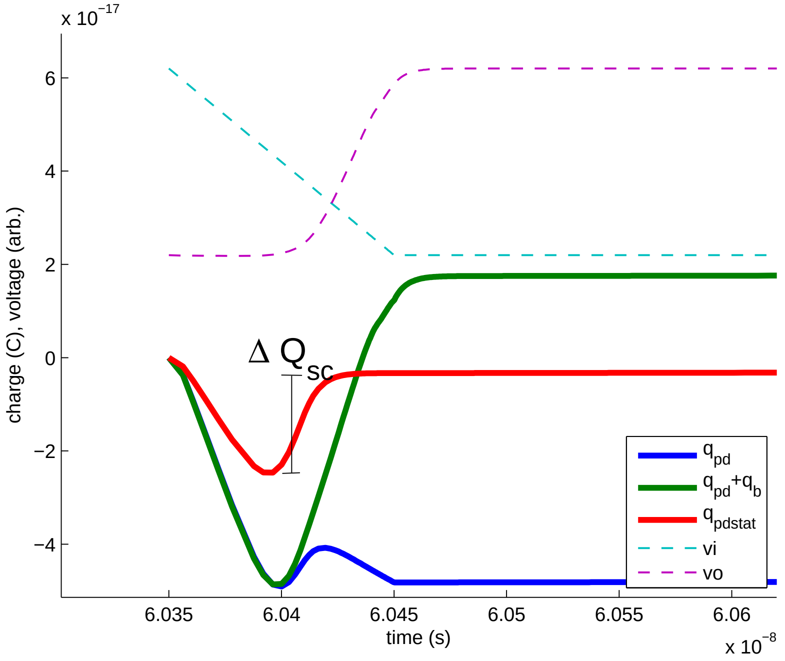


Figure 37. Integrals of current in the pull down network during a rising output transition.

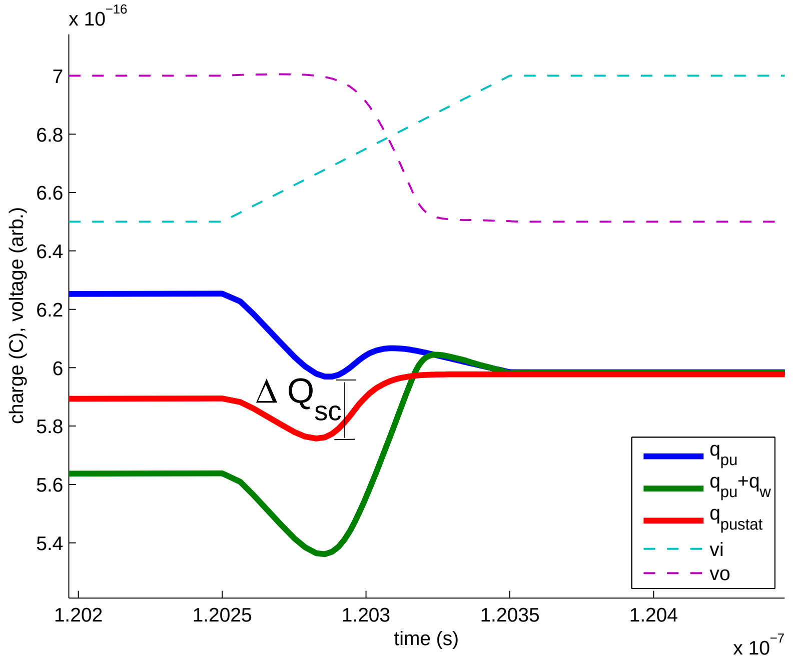


Figure 38. Integrals of current in the pull up network during a falling output transition.

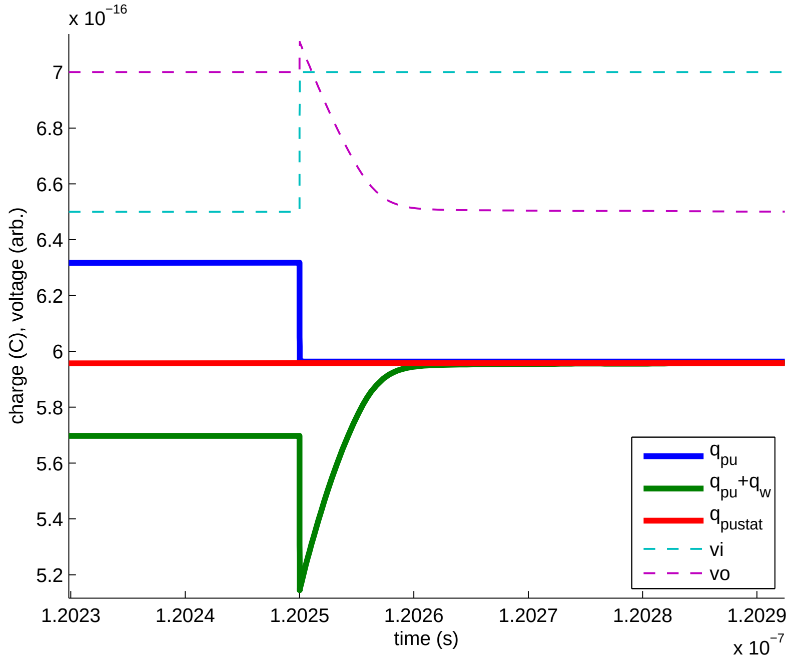


Figure 39. Integrals of current in the pull up network during a very fast rising input.

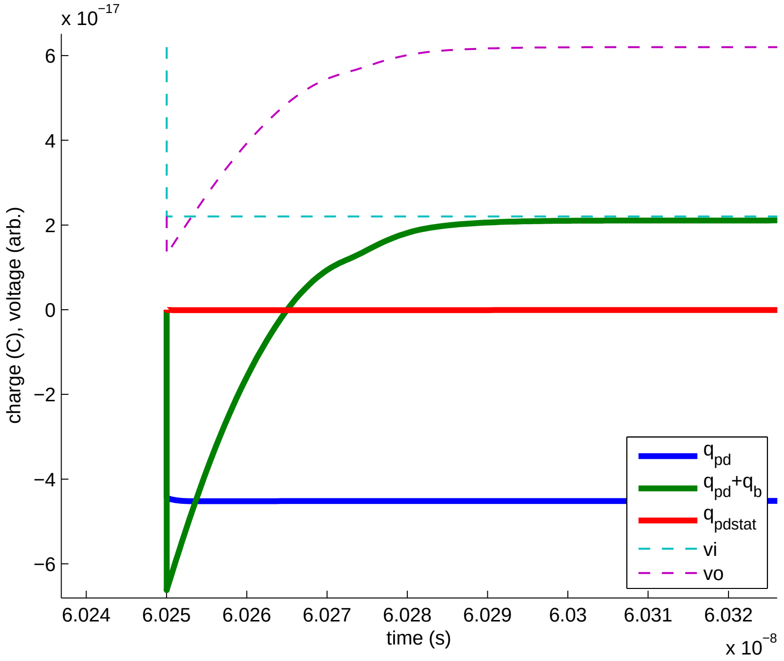


Figure 40. Integrals of current in the pull down network during a very fast falling input.

Figures 37, 38, 39, and 40 show that charge measured by integrating the static current through these devices produces something reasonable for real input transitions, and something negligibly small for very fast input transitions.

A comparison of the short-circuit measurement methods is shown in Figure 41. An inverter was simulated with increasing input slew rates for varying amounts of fanout. The black line in the figure shows input slew rate divided by average propagation delay (generated by that input slew rate) for a FO4 inverter. In each simulation, measurements of short-circuit energy were computed using the three different methods described in this section. Method 1 being the integral of GND current during a pull up transition plus the integral of V_{DD} current during a pull down as described by Equation 32 . Method 2 being the integrals of only the positive portions of these currents minus the value obtained for a step input, Equation 36. And Method 3, the one proposed in this thesis, where only the positive static currents in the off-switching transistors are integrated as described by Equation 37.

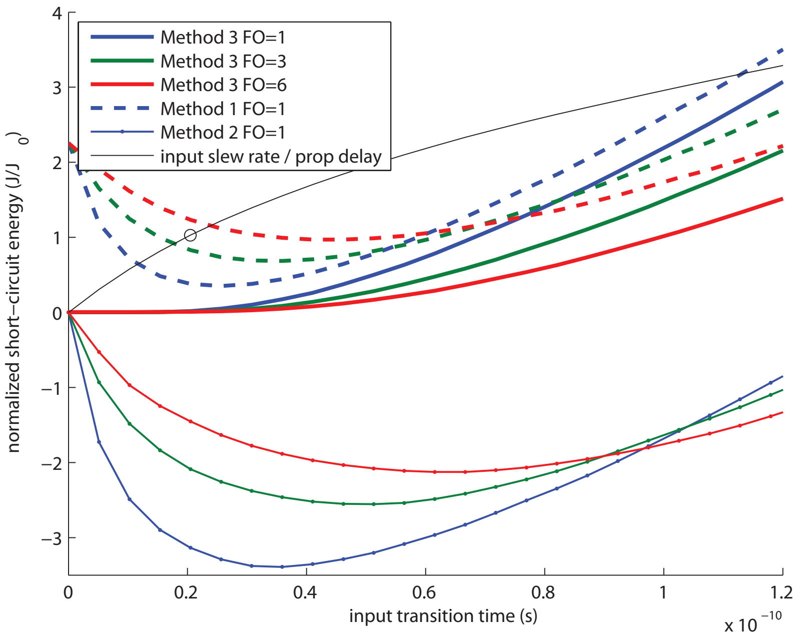


Figure 41. Comparison of short-circuit energy measurement techniques for an inverter for varying input transition times and various fanouts.

Method 1 does not trend to zero for increasingly fast inputs. Method 2 does but only by manually setting this energy to be zero for step inputs. Method 3 obtains zero naturally. Methods 1 and 2 both have negative trends initially for short circuit energy with increasing slew rates. This produces negative answers with Method 2 for a large range of slew rates. Intuitively, short-circuit energy should monotonically increases for slower input slew rates, converging on zero for increasingly fast slew rates, and monotonically decreases with output load size (higher load capacitance leads to slower output transitions and thus less overlap between input voltage and output voltage). Only Method 3 satisfies all of these trend expectations. Figure 42 shows the various methods applied in the exact same fashion (only the FO4 case is shown) to a NAND2 gate's b-input (the input connected to the gate of the nFET farthest from the output). Here Method 1 gets farther off as this gate has more parasitic capacitance to charge than the inverter case, and Method 1 is incapable of differentiating the parasitic capacitance

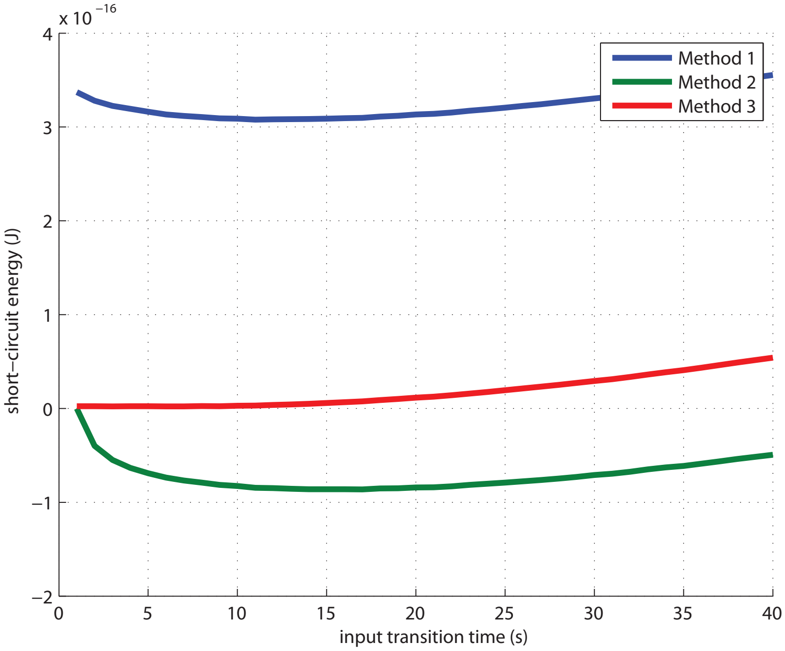


Figure 42. Comparison of short-circuit energy measurement techniques for the b-input (transistor farthest from output) of a FO4 NAND2 gate for varying input transition times.

charging from short-circuit energy. Figure 43 shows the various methods applied to an FO1 inverter. In this case, both Method 1 and Method 2 produce negative results for some range of input slew rates.

5.5 Measuring Dynamic Power

If we are confident in our measurements of *short-circuit* energy, then d*ynamic energy* is simply the difference of *active* and *short-circuit* energy:

$$E_{dyn} = E_{act} - E_{sc} \tag{38}$$

Dynamic power is not measured directly, it is simply the short-circuit power measurement subtracted out of the active power measurement (something that is assumed to be trivial to measure); thus correctness of the dynamic power extraction then depends on the quality of our measurement of short-circuit power. Fortunately, we can

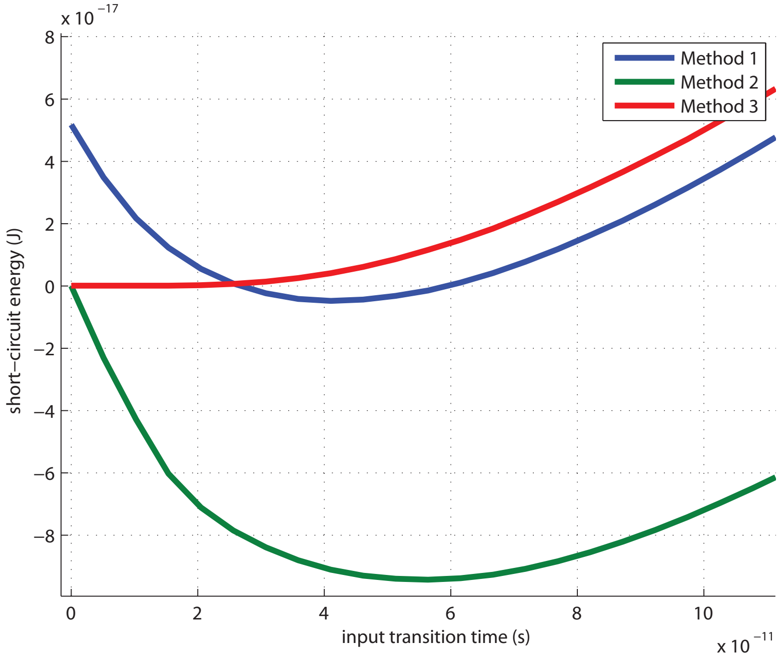


Figure 43. Comparison of short-circuit energy as measured by the different techniques for a FO1 inverter sized roughly for equal rise and fall times. Here it can be seen that Method 1 can give negative answers as well.

at least make sure that our measurements of short-circuit power are causing dynamic power to scale as expected.

We do that by observing our definition of E_{dyn} and its dependence on h, and taking the derivative of E_{dyn} with respect to h:

$$E_{dyn} = C_{tot}V_{dd}^2 = V_{dd}^2(C_{out} + C_p) = V_{dd}^2(C_{in}h + C_p) \tag{39}$$

so

$$E_{dyn,vh} = \frac{d}{dh}E_{dyn} = V_{dd}^2C_{in} \tag{40}$$

where $h = C_{out}/C_{in}$ is called the *electrical effort* of the gate and is related to the fanout. The quantity h can be easily controlled by having the gate driving multiple copies of itself, where the number of copies is equal to the *electrical effort .*

Then by taking *dynamic power* to be the difference of *active* and *short-circuit*, we arrive at a way to verify whether or not *dynamic power* is scaling linearly with output load capacitance, but not whether or not we are confusing *short-circuit power* with the portion of *dynamic power* associated with parasitic capacitance. Our error in measurement is then

$$Err = 100\left|\frac{\frac{E_{dyn,vh}}{C_{in}} - V_{dd}^2}{V_{dd}^2}\right| \tag{41}$$

Where this error metric is also sensitive to our measurements of the input capacitance to the gate:

$$C_{in} = \frac{-1}{V_{DD}}\int_{t1}^{t2} i_g dt = \frac{1}{V_{DD}}\int_{t3}^{t4} i_g dt \tag{42}$$

Where the limits of integration are defined by the $Regions$ of Figure 28 as per usual, the first integral being over $RegionI$ which is a rising output transition and therefore current is flowing off of the driving gate, and in the second integral taken over $RegionIII$ the output is undergoing a falling transition and therefore current is flowing onto the gate. This total charge flowing onto or off of the gate in either region should be equal and magnitude, it divided by V_{DD} is the effective capacitance.

Results for an inverter versus input slew rate can be seen in Figure 44 for the various different *short-circuit* measurement methods as described in Section 5.4 . In general this will also depend on how well SPICE does at conserving charge and is thus dependent on quite a few SPICE parameters. For all simulations reported in this document, the following parameters were the same, are listed in Table 2.

It can be seen that Method 1, though incapable of distinguishing the parasitic portion of *dynamic power* from *short-circuit* gets the scaling of *dynamic* with load capacitance (in this case) better than the other two methods. Though, none of the methods have a particularly high error in this metric over the entire range. As a useful point of reference,

Table 2. Simulation parameters and tolerances

name	value
iabstol	1e-14
vabstol	1e-6
reltol	1e-4
gmin	1e-12
method	gear2only
simulator	SPECTRE MMSIM70
model	BSIM 4v4

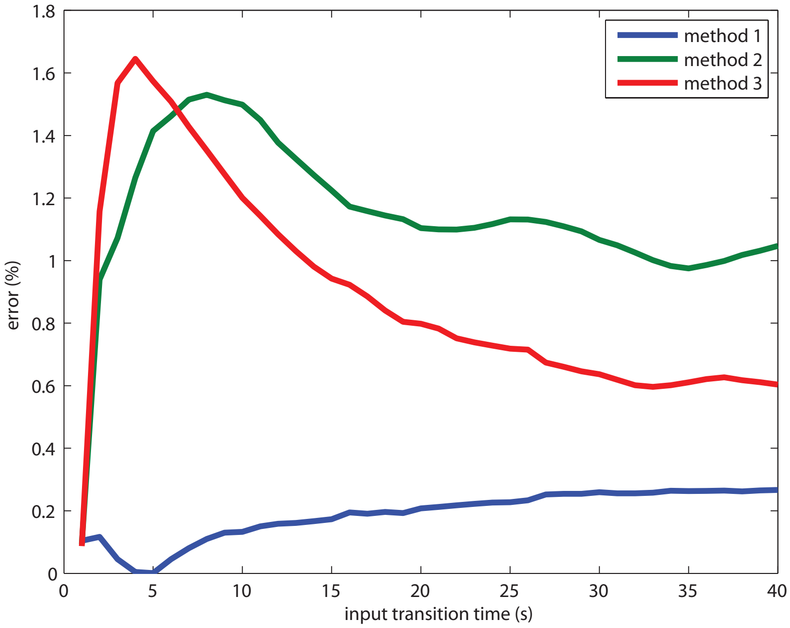


Figure 44. Error in expected *dynamic power* scaling (Equation 41) as extracted from *active power* by varying *short-circuit* measurement methods. All simulation points are of an inverter of $h = 1$ versus input slew rate.

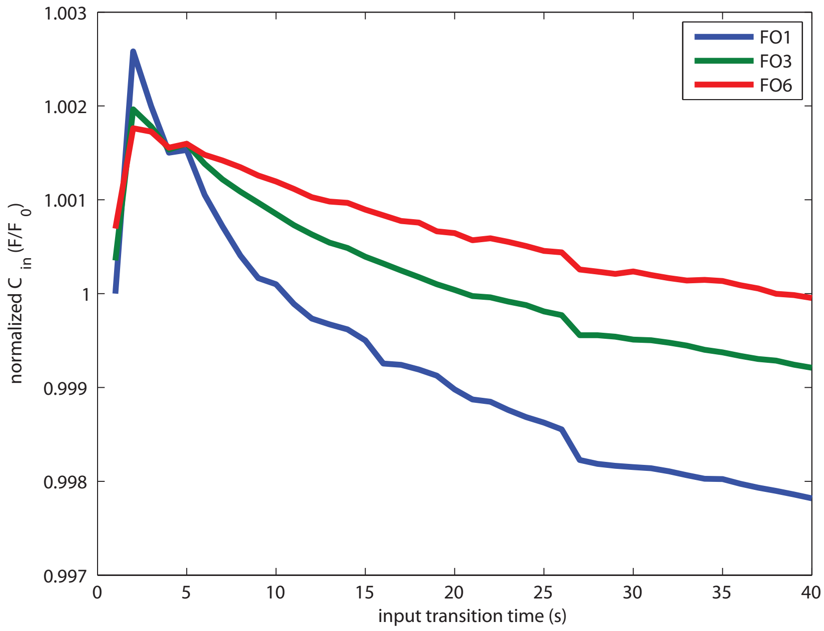


Figure 45. Measured effective input capacitance, C_{in} (Equation 42). All simulation points are of an inverter of $h = 1$ versus input slew rate.

Figure 45 shows normalized measurements of input gate capacitance, C_{in}, versus slew rate for various fanouts (in this case the fanout is equal to the *electrical effort*). It is expected that this quantity be constant, but instead changed with the same relative magnitude by which Methods 2 and 3 are erroneous in the *dynamic* scaling metric.

Figures 46 and 47 show the same simulations but this time for the b-input (the input connected to the gate of the nFET farthest from the output) of a 2-input NAND gate. For this more complicated gate, it can be seen that the measured C_{in} is much more consistent and that Method 3 produces the least error.

5.6 Measuring Static Power

Assuming insignificant amounts of gate and load current in Regions II and IV, *static power* is given by

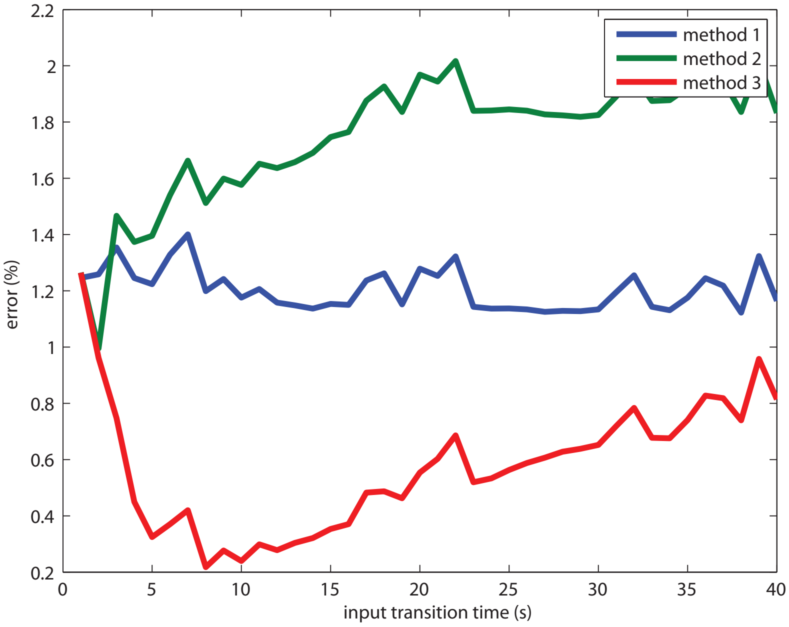


Figure 46. Error in expected *dynamic power* scaling (Equation 41) as extracted from *active power* by varying *short-circuit* measurement methods. All simulation points are of the b-input (input farthest from output) of a 2-input NAND gate with $h = 1$ versus input slew rate.

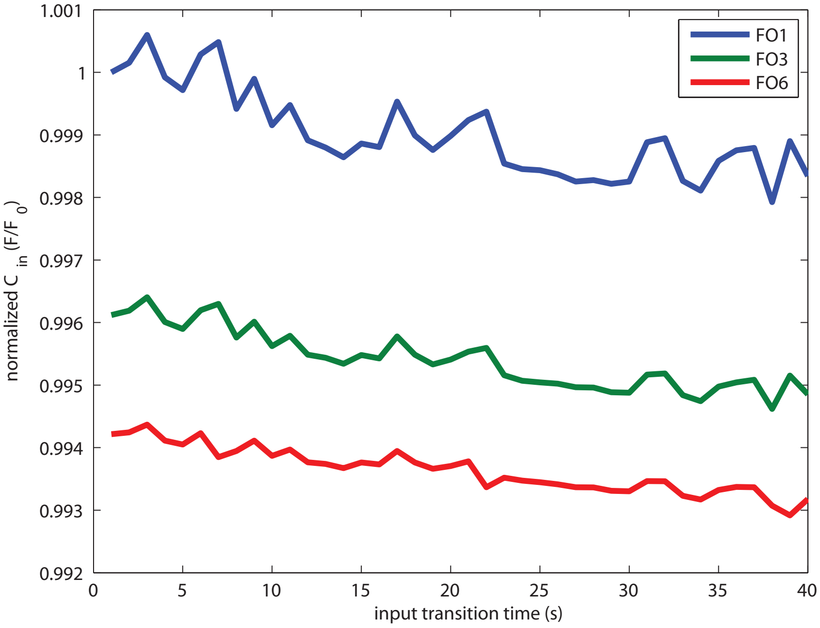


Figure 47. Measured effective input capacitance, C_{in} (Equation 42). All simulation points are of the b-input (input farthest from output) of a 2-input NAND gate with $h = 1$ versus input slew rate.

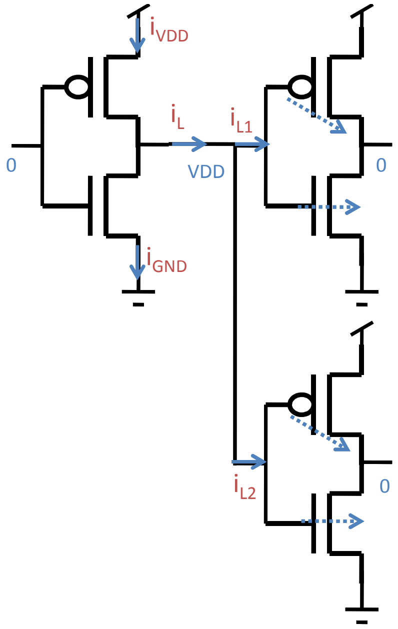


Figure 48. Gate leakage current paths for a FO2 inverter after a rise transition.

$$P_{stat,avg} = \frac{V_{dd}}{2}[i_{vdd}(t \in RegionII) + i_{vdd}(t \in RegionIV)] \quad (43)$$

where $i_{vdd}(t \in RegionII)$ means the current flowing out of the the V_{DD} rail during some time, t, in $RegionII$, etc. With $RegionII$ and $RegionIV$ being the static regions of operation after a pull up and a pull down transition as defined in Figure 28.

However, in modern processes, static gate current is a significant source of of leakage power and needs to be included in P_{stat} measurements for accuracy. This would at first appear to cause a significant dependence on load size for leakage current. Larger load sizes, when comprised of CMOS gates, mean a larger amount of gate leakage current in the load gates that has to be sourced (after a rise) and sinked (after a fall) by the driving gate.

Consider the FO2 inverter of Figure 48 . The driving gate has finished making a rise transition, and its output has charged to V_{DD}. Static currents flow through all terminals

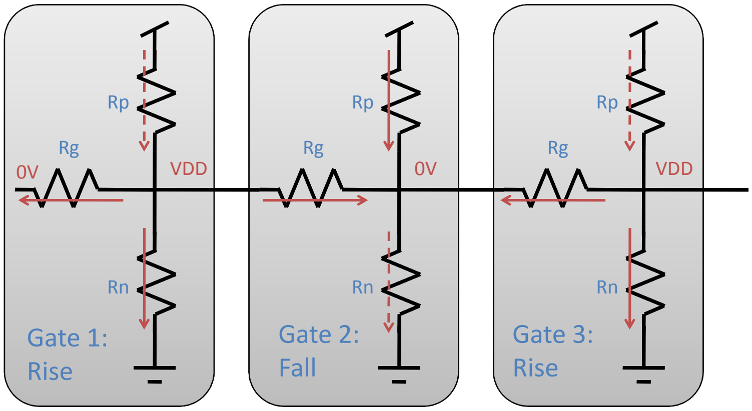


Figure 49. Static currents in a 3-stage path. In each gate the pull down network, pull up network, and gate network are approximated as R_p , R_n , and R_g respectively. Solid red arrows show currents dropping a voltage of VDD, whereas dashed red arrows represent currents flowing through no appreciable voltage drop. Static power then is measured as $P_{stat} = -V_{dd}[i_{gnd} + i_g]$ for a gate that has risen and $P_{stat} = V_{dd}[i_{vdd} + i_g]$for a gate that has fallen. With all currents referenced going into the logic gate.

of the gate: subthreshold currents in the off transistors in the pull down network, gate currents flowing out of the logic gate, and load current flowing into the gates of the load logic gates. This current flowing into the load depends on the load topology, and scales linearly with fanout.

In this case Equation 43 would overestimate the amount of static power consumed by this gate, as it simply measured the amount of current leaving the V_{DD} rail and assumes that this current passes to GND in this gate. Which is not the case, as the current flowing into the load is just a source for the gate leakage power dissipation of the load gates and does not get dissipated in the driving gate. To properly assign leakage power dissipation to gates in a topology like this, it becomes much easier to make approximations about the voltage drops across the pull up and pull down networks.

Assuming that after a rise, that the output is charged near enough to V_{DD} that the voltage drop across the pull up network is negligible, then we can approximate that no leakage power is dissipated in the pull up network. And after a fall approximate that no leakage power is dissipated in the pull down network. These assumptions about voltages and static current paths are shown in Figure 49 .Leakage power then becomes:

$$P_{stat,rise} = -V_{DD}[i_{gnd} + i_g] \tag{44}$$

$$P_{stat,fall} = V_{DD}[i_{vdd} + i_g] \tag{45}$$

Where $P_{stat,rise}$ is the static power dissipation after a rise ($RegionII$) and $P_{stat,fall}$ is the static power dissipation after a fall ($RegionIV$). Average static power dissipation of a gate driving multiple gates can then be expressed as

$$P_{stat,avg} = V_{DD}[(i_{vdd} + i_g)|_{t \in RegionIV} - (i_{gnd} + i_g)|_{t \in RegionII}] \tag{46}$$

Figure 50 shows various ways to measure leakage currents in a gate with fanout. From this figure it can be seen that measuring leakage power by Equation 46 has the desired effect of being independent of fanout.

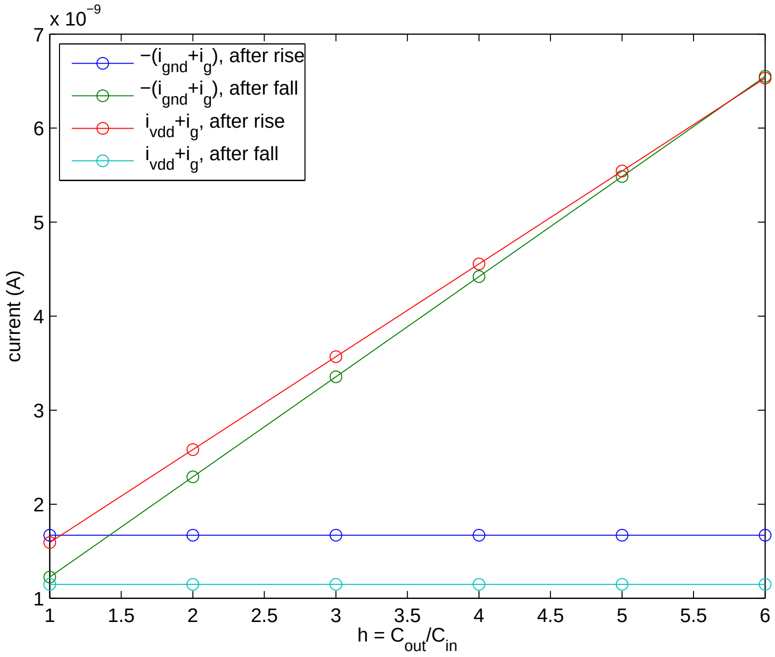


Figure 50. Leakage currents versus h for an inverter with $W_p/W_n = 3$ in a 65nm process. Shown are $-[i_{gnd} + i_g]$ and $[i_{vdd} + i_g]$ after a rise and a fall. Notice that the on network sources more current as h increases.

CHAPTER 6

A FLOATING GATE BASED FIELD PROGRAMMABLE MIXED-SIGNAL ARRAY

We present the Field Programmable Array of Analog and Digital Devices (FPAADD) as a novel implementation of a Field Programmable Mixed-signal Array (FPMA). The FPAADD is a hybrid combination of a Field Programmable Analog Array (FPAA) and a Field Programmable Gate Array (FPGA). Unlike other FPMAs where the FPGA and FPAA portions are kept separate, this architecture closely integrates the two in a fine-grained interleaved array. Instead of using hard-coded data converters, the FPAADD synthesizes data converters out of its reconfigurable fabric. The analog and digital portions share a common global interconnect. Floating Gate (FG) transistors are used as the switch and memory elements of the chip, providing better switch performance and power over traditional SRAM based approaches. The precise programmability of the FG switches also allow for computation to take place in the interconnect. These key differences make the FPAADD much more general purpose than previous FPMA architectures. The FPAADD consists of 27 x 8 array of 108 digital and 108 analog tiles and peripheral circuitry on 5 x 5 mm^2 die fabricated in a 0.35-μm CMOS process, and contains more than 130,000 FG transistors.

6.1 Introduction

Reconfigurable systems exist as an attractive alternative to custom ASIC design when the monetary cost of fabrication, or the manufacturing time is too high. Digital systems cater particularly well to reconfigurability in that any digitally solvable problem can be implemented with a very small number of building blocks that are functionally insensitive to fan-out and fan-in. FPGAs use look-up tables (LUTs) and flip flops to implement arbitrary and small number-of-input Boolean equations and state machines. High level functions are built up from large numbers of these blocks being connected together by a programmable interconnection network (the interconnect). With digital design's ability to be abstracted to very high level programming languages, systems can be rapidly prototyped and implemented on FPGAs. Of course this flexibility does not come without an associated cost increase to area, power, and degradation of system speed.

While all solvable problems can be solved in the digital domain, some problems map more efficiently to other domains. Problems like integer factorization, searching unsorted lists, and simulating quantum many-body systems, for instance, have solutions implementable on quantum computers that are algorithmically more efficient than the best known solutions on probabilistic Turing machines (classical digital computers). The filtering, smoothing, or modulation of sensor signals are efficiently solved in the domain of analog signal processing or analog computation. For a digital computer to even begin to work on real world data, some sort of analog processing must take place to convert it into a compatible format.

Analog solutions, when implemented in silicon, incur the same costs of fabrication and design time (if not significantly more design time) iteration that makes reconfigurable solutions attractive for many applications. The FPAA is the analog equivalent of the FPGA. In essence it is a set of low level analog computational elements in a reconfigurable interconnect. Unlike FPGAs, however, the choice of computational elements tends to vary quite a bit, and thus FPAAs come in many different flavors: some use discrete-time, switched-capacitors, some are based on operational amplifiers and Gm-C circuits, some use translinear elements as the building blocks, and some everything in between [17, 18, 19, 20, 21, 22].

Prototyping and implementing mixed-signal systems, where both analog and digital computation is taking place, could be done with a discrete FPGA, FPAA and data converters [23]. But for single chip, system on a chip (SoC) solutions, one needs an FPMA, a reconfigurable system containing both low level digital and analog computational elements. Such systems could also be used to tackle problems by partitioning them into parts suited to being solved in the analog domain and parts in the digital domain.

A straightforward FPMA implementation is one that contains separate FPGA and FPAA arrays, and data converters in between, such as the MADAR chip [24]. This approach, while a single chip solution, offers no new functionality over discrete FPAAs, FPGAs, ADCs and DACs. Another common approach is to also include a hard-coded microprocessor [25][26][27]. These implementations generally have a very skewed percentage of analog or digital components, with one or the other being highly specialized. Communication between the analog and digital portions is almost always through hard coded dedicated data converters. We propose a new FPMA system, the Field Programmable Array of Analog and Digital Devices (FPAADD), as a more general reconfigurable mixed-signal alternative.

Instead of partitioning and compartmentalizing the analog and digital portions of the reconfigurable system, the FPAADD tightly interleaves the digital and analog computational elements in a heterogeneous Manhattan style interconnect scheme. A key difference with previous approaches are data converters being synthesized from the FPAADD fabric as needed. This allows the system to be more flexible in the quantity and type of data converters. A synthesized 8-bit data converter, while taking an efficiency hit from the parasitics introduced by the interconnect, may still be more energy efficient than using a hardcoded 16-bit data converter, when only 8-bits of accuracy are needed. A second important difference is the global routing interconnect that shows up as a shared resource between the analog and digital portions of the array. The interleaving scheme of the FPAADD along with the above key differences allows for a heavily skewed application to spend more resources on doing computation in one domain or the other. The digital elements are LUTs and flip flops, while the analog elements range in generality from discrete transistors and capacitors to operational amplifiers and multiple input translinear elements (MITEs). Unlike other approaches that are SRAM based for their configuration, the FPAADD's reconfigurable interconnect is completely implemented by floating-gate (FG) transistors. FG transistors have many benefits: on resistance comparable to transmission gates while maintaining the parasitic capacitance of a single transistor, significantly reduced subthreshold leakage current for the off devices, non-volatility, and precise programmability [28]. This last part allows the interconnect itself to perform useful functions from low level analog bias currents to analog vector matrix multiplication (VMM) [29].

The generality and flexibility of the FPAADD enable it to implement a vastly larger application space over previous FPMA designs. Examples include, but are not limited too: built-in self test, digitally assisted analog computation, industrial control, machine learning, mixed-signal processing, digitally tunable analog circuits, to biologically inspired neuromorphic circuits.

The rest of the chapter is organized as follows. Section 6.2 describes the detailed architecture and the building of the FPAADD, Section 6.3 introduces the software stack used to place-and-route and program netlisted circuits onto the FPAADD. Section 6.4 has measurements of low-level computational elements, parasitic delay from switches, and general system verification. Section 6.5 shows a subset of potential applications and their mapping to the FPAADD are described as well as measured system results for two applications: a VCO based ADC and a 2^{nd} order sigma-delta modulator are presented. Finally, Section 6.6 concludes the chapter.

6.2 FPAADD Architecture

The computational blocks are clusters of computational elements and an interconnect network called the local interconnect. Analog components are clustered together

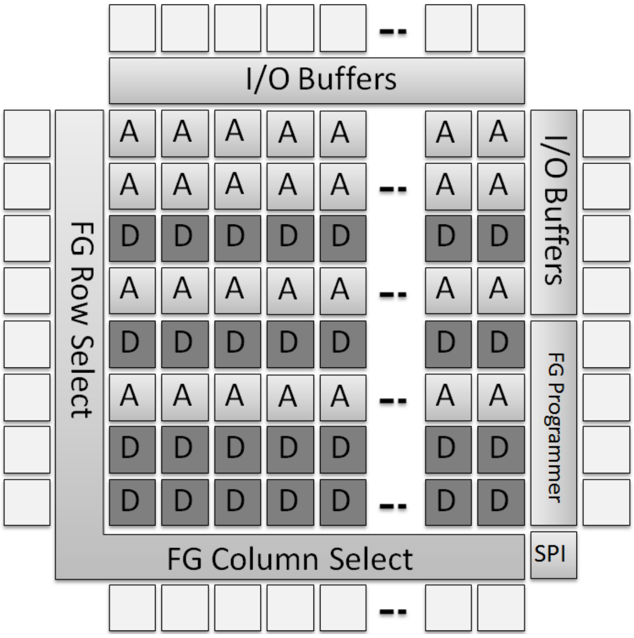


Figure 51. The general architecture of the FPAADD array.

to form the Computational Analog Blocks (CABs) while digital components are clustered into Combinational Logic Blocks (CLBs).

In the FPAADD, the choice of analog devices range in complexity from discrete transistors and capacitors to FG input operational transconductance amplifiers (FG-OTAs). Other devices include: transmission gates, and multiple-input translinear elements (MITEs) [30]. The choice of digital devices are LUTs and flip flops.

CLBs and CABs are arranged in a tileable Manhattan style global interconnection scheme (Figure 53). An analog tile comprises a CAB, two connection blocks (C-Blocks), and a switch block (S-Blocks). The C-Blocks allow inputs and outputs from the CAB to connect to the global routing tracks, while the S-Block routes nets on global

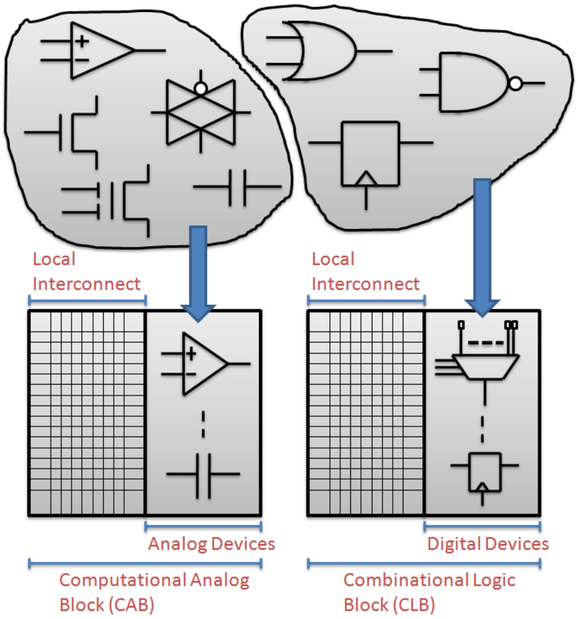


Figure 52. Left, analog devices (MOSFETs, capacitors, etc.) are grouped together with local interconnect, a sea of reconfigurable switches for connecting the devices together, to form Computational Analog Blocks (CAB). Right, digital devices (Flip-Flops and look-up tables) are grouped together with local interconnect to make Combinational Logic Blocks (CLB)

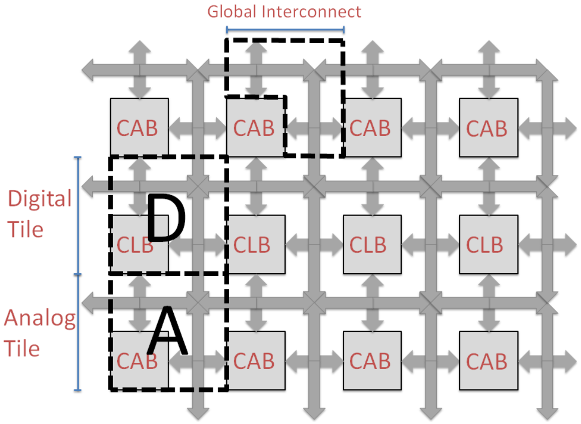


Figure 53. Interchangeable digital and analog tiles are built from either a CLB or a CAB with reconfigurable routing that allows signals to propagate between tiles (global interconnect)

tracks through the chip. The digital tile is the exact same but with a CLB. The two different tiles are completely pin compatible.

FG transistors are used for the switches and state storing elements on the chip. The dynamic range of the FG switches allow for ON performance comparable to transmission gates with parasitic capacitance of a single FET, with leakage currents an order of magnitude less than standard SRAM based alternatives [31]. The non-volatile nature of the floating-gates means the chip does not have to be reprogrammed on power up. The continuum between the on and off states allow the routing infrastructure to perform tasks other than just connecting nets: tunable delays, current biases, and vector matrix multipliers (VMM), for instance, are all easily implementable by the interconnect. Figure 54 shows a small array of FG transistors.

The core of the FPAADD is an array of these tiles. The tiles are interleaved on a row by row basis with a higher density of digital rows on the bottom and analog rows on the top. The rest of the chip is floating-gate selection and programming infrastructure (controlled by an SPI bus), and buffered and non-buffered I/O. The top level arrangement of the chip is shown in Figure 51.

The Manhattan-style routing architecture chosen for the FPAADD is the parameterizable one as understood by the VTR software. Things like the number of global tracks, track lengths, number of inputs and outputs from cells, etc. are all variables. In general, arbitrarily cranking up these variables usually leads to an increase in routing options. This can increase the chance that the place and route heuristics successfully find a routing solution to any given target circuit, and or make impossible to route circuits routable. The biggest trade off in doing so, is that an increase in routing options comes from an increase in the number of switches on any given net, and thus increases the parasitic delay of any routed signal, the dynamic power consumed on transitions, increases static power, and reduces the fraction of the silicon devoted to the actual computational elements.

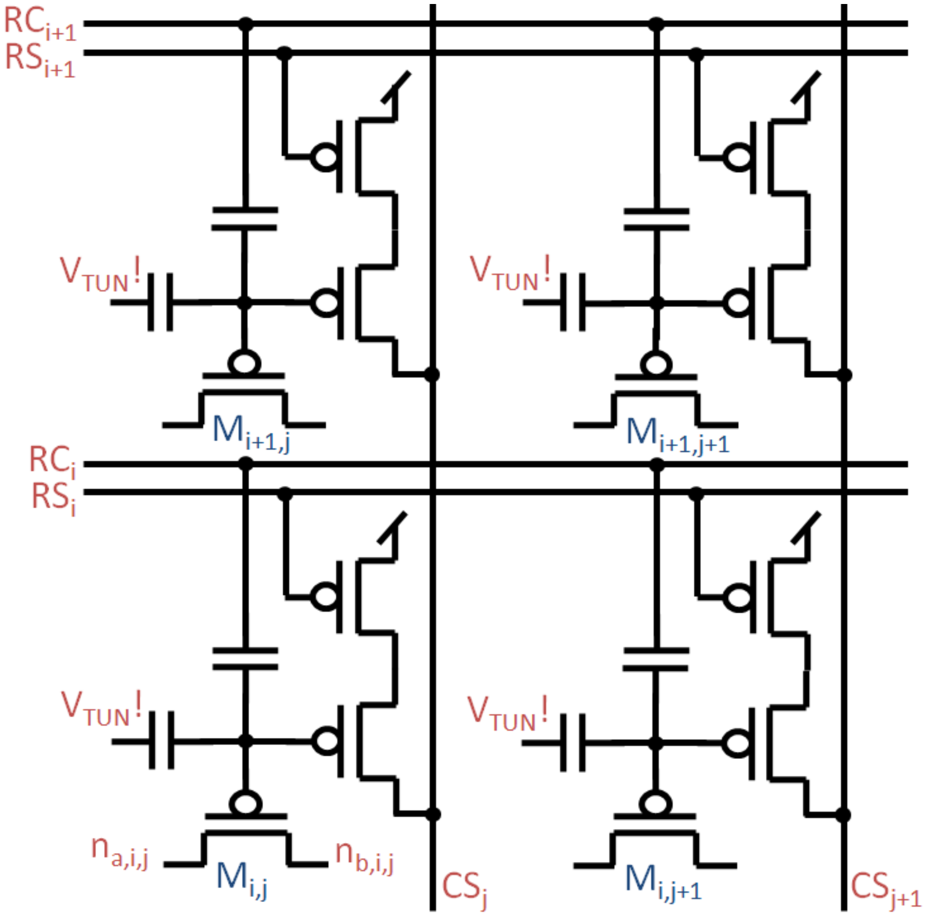


Figure 54. Programming is achieved by globally removing charge from the floating-gate nodes through C_{TUN} via Fowler-Nordheim tunneling, and then selectively adding charge through $M_{i,j}$ with impact carrier hot channel electron injection. Injection of charge per row is controlled by the selection lines CS_i, and per column by the drain lines CS_j.

The FPAADD routing architecture will be presented parameterizable, and with the specific values of any variable. The choices for said variables was, to a certain extent, a bit arbitrary. Though certain performance goals, i.e.. a minimum desired routed device to device bandwidth did set upper bounds on the number of allowable connections on certain nets.

6.2.1 Floating-Gate Switch

The most basic and ubiquitous component of any highly reconfigurable architecture is the switch and the switch's state storage. In the majority of modern FPGAs, this is

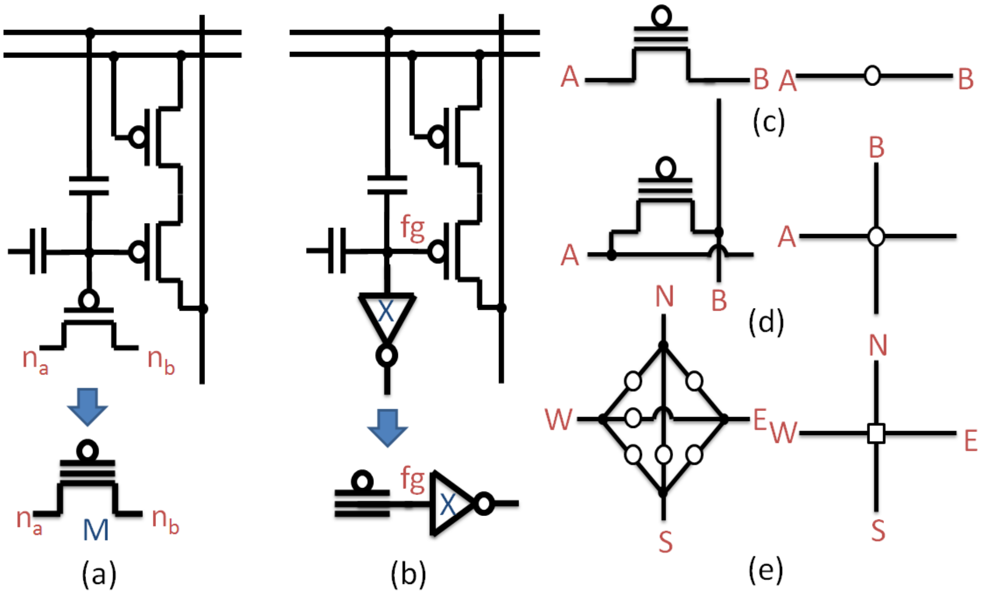


Figure 55. a) pFET switch with floating-gate memory and circuit symbol, b) Circuit symbol for a floating-gate memory element setting the gate input voltage of an inverter, c) a pFET floating-gate switch connecting two abutting nets, d) a pFET floating-gate switch connecting two crossing nets, e) six pFET floating-gate switches implementing an s-switch.

implemented by a single nFET whose gate is driven by SRAM. The FPAADD, instead, uses floating-gate transistors as the switch and memory.

Building up the local interconnect and high level portions of the chip is greatly facilitated by defining some circuit symbols: Figure 55a shows the symbol used for a floating-gate pFET switch, and Figure 55b the symbol for when a floating-gate is used as the gate input to a larger circuit like an inverter.

An open circle, as shown in Figure 55c, denotes when a switch is used to connect two abutting net lines. When a switch is used to allow connectivity between two crossing net lines an open circle is drawn over the crossing of the two nets (Figure 55d). Figure 55e shows the symbol for an s-switch connection topology, an open square. The s-switch allows a signal entering from any side to propagate across, make a turn, split in two directions, allows two nets to cross each other, or turn away from each other.

6.2.2 Combinational Logic Block

The Basic Logic Element (BLE) is the building block of the digital circuits. The standard BLE is a k-input look-up table whose output is either registered or not by a flip-flop. Shown in Figure 56 is the BLE implementation used in the FPAADD. Instead of using a standard flip-flop, a JK-FF is used that can be configured as a T-FF or a D-FF. The clock can be routed from the local interconnect, the BLE's look-up-table, or come from a global signal. These choices were made to allow of high density synthesis of asynchronous counters.

Figure 57 shows that the CLB is comprised of NO number of BLEs and a sea of local interconnect. The inputs to each BLE come from either any of the NI primary inputs to the CLB or from the outputs of any BLE in the CLB. The NO outputs of the CLB are hardwired to the outputs of the BLEs in a one-to-one fashion. The configuration of the local interconnect allows for a deterministic and guaranteed routing solution for any clustering of any NI inputs and NO BLEs. Where $NO = 4$ and $NI = 8$.

6.2.3 Computational Analog Block

The CAB (Figure 58) is the analog equivalent of the CLB. It is a cluster of analog devices and local interconnect, however, instead of a homogeneous set of devices, the CAB in the FPAADD contains: floating-gate based operational transconductance amplifiers (OTAs), switched capacitor optimized transmission gates, MOSFETs (either common centroid pFETs or nFETs), capacitors, and multiple-input translinear elements (MITEs: floating-gate pFETs with multiple input control gates). This set of devices was chosen to make the FPAADD CABs compatible with the generic CABs from the previous FPAA of [20]. Inputs to the devices come from the NI primary inputs, the two hardwired V_{DD} and gnd signals, or the outputs of any device in the CAB. The NO outputs of the CAB are multiplexed from the set of CAB device outputs. This was chosen because the number of devices in the CAB exceeded that in the BLE and it was desired to keep the same number of I/O in the CAB as in the CLB: $NO = 4$ and

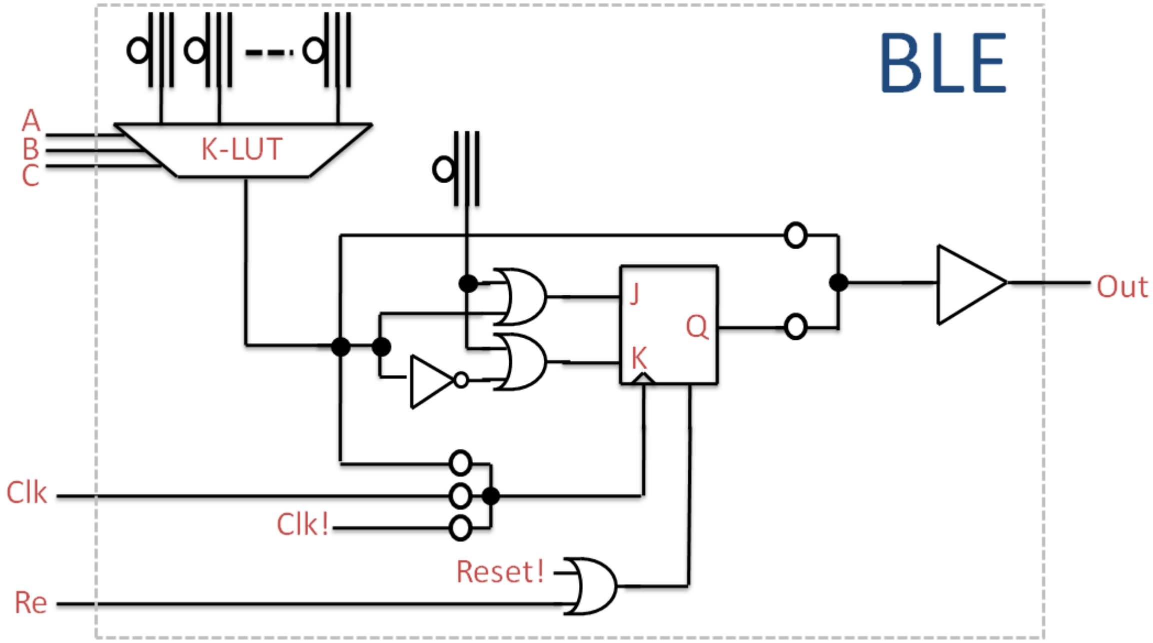


Figure 56. The BLE is a 3-input LUT whose output can be registered with a FF. The register is implemented as a JK-FF. It cab be configured as a standard FF or a T-FF, with the clock originating from the local interconnect, the output of the LUT, or a global line.

$NI = 8$.

While the routability of the CLB was complete, this is not the case for the CAB. The existence a completely deterministic and guaranteed routing solution for all combinations of NI inputs, NO outputs and CAB devices depends on whether the clustering can be partitioned such that the implied input/output relationship of the devices is preserved: an output can go to multiple inputs, but multiple outputs can not go to a single input. In the CLB, the inputs and outputs are well defined, as is the case with CMOS digital gates. While an OTA may have well defined inputs and outputs, and the gate of a MOSFET is easily classified as an input, classifying the sources or drains of a MOSFET, for instance, as either inputs or outputs is rather arbitrary. If partitioning of the circuit to be clustered in the CAB preserves these mappings then the cluster is guaranteed to route in a deterministic manner. Since many analog circuits do not partition this way, this does not automatically mean they will not route, the output multiplexor allows for limited support of shorting of outputs. If outputs are to be shorted, and if the output

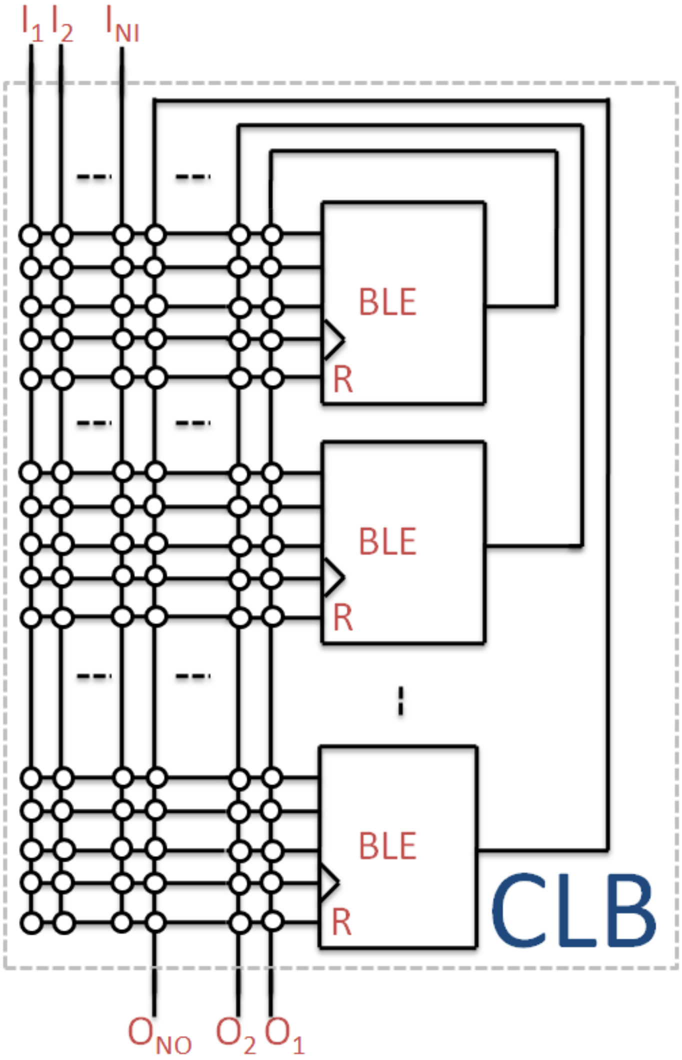


Figure 57. The CLB comprises multiple BLE devices and a sea of local interconnect. The outputs from the NO number of BLEs are the primary outputs from the CLB, and the inputs to the BLEs come from the NI number of primary CLB inputs and the NO BLE outputs. $NO = 4$ and $NI = 8$ for the FPAADD.

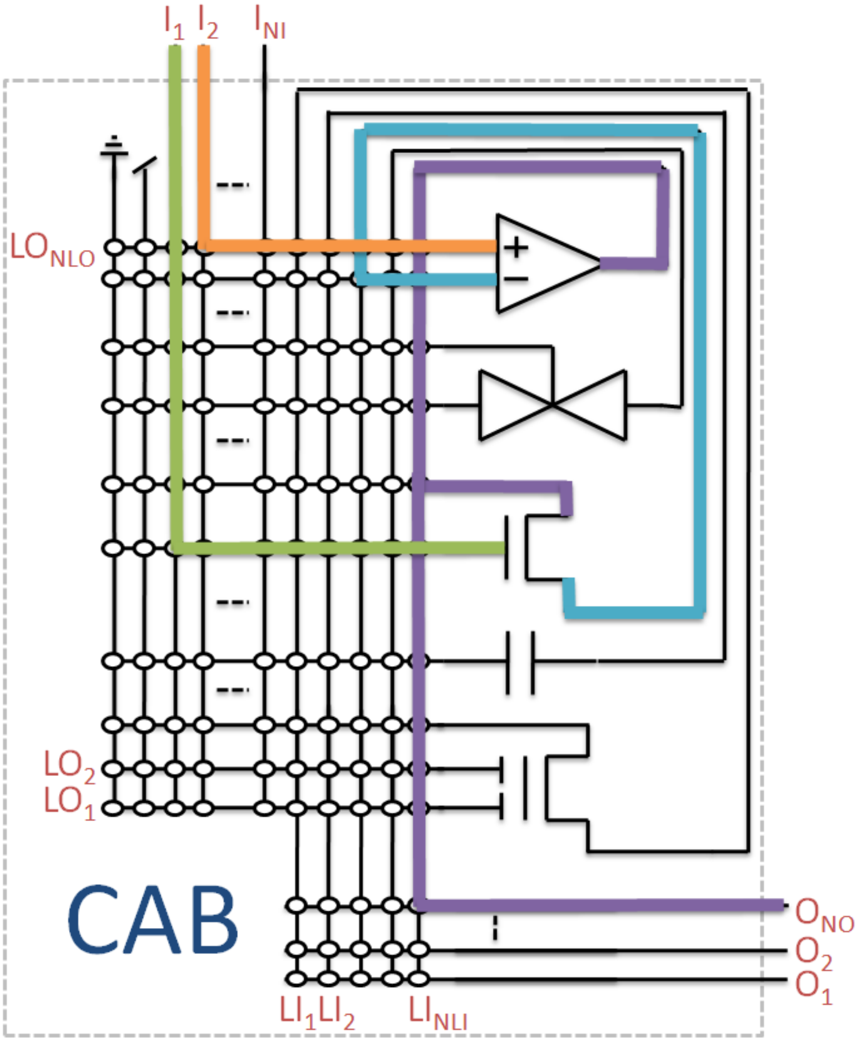


Figure 58. CAB architecture showing devices and local interconnect. Inputs to the local interconnect are vertical lines and outputs from the local interconnect are horizontal. I and LI are the primary inputs to the CAB and the outputs from the CAB devices respectively. O and LO are the primary outputs from the CAB and inputs to the CAB devices respectively. The example wiring shows a configured logarithmic amplifier circuit.

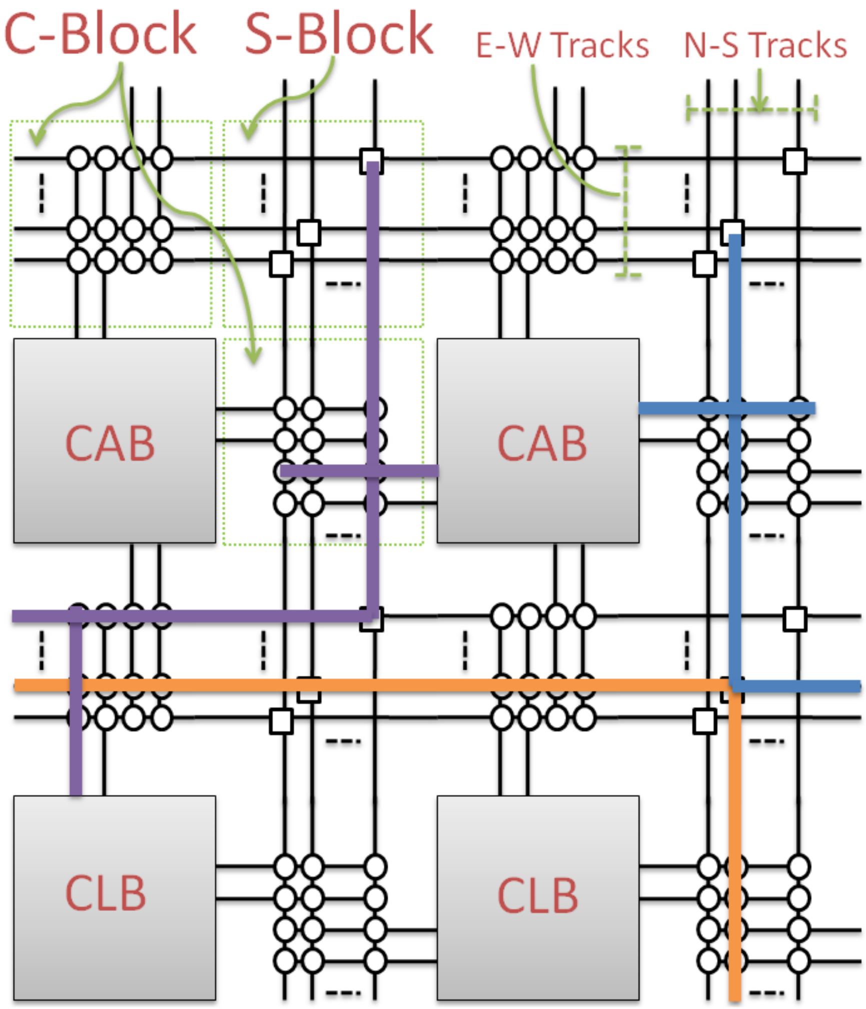
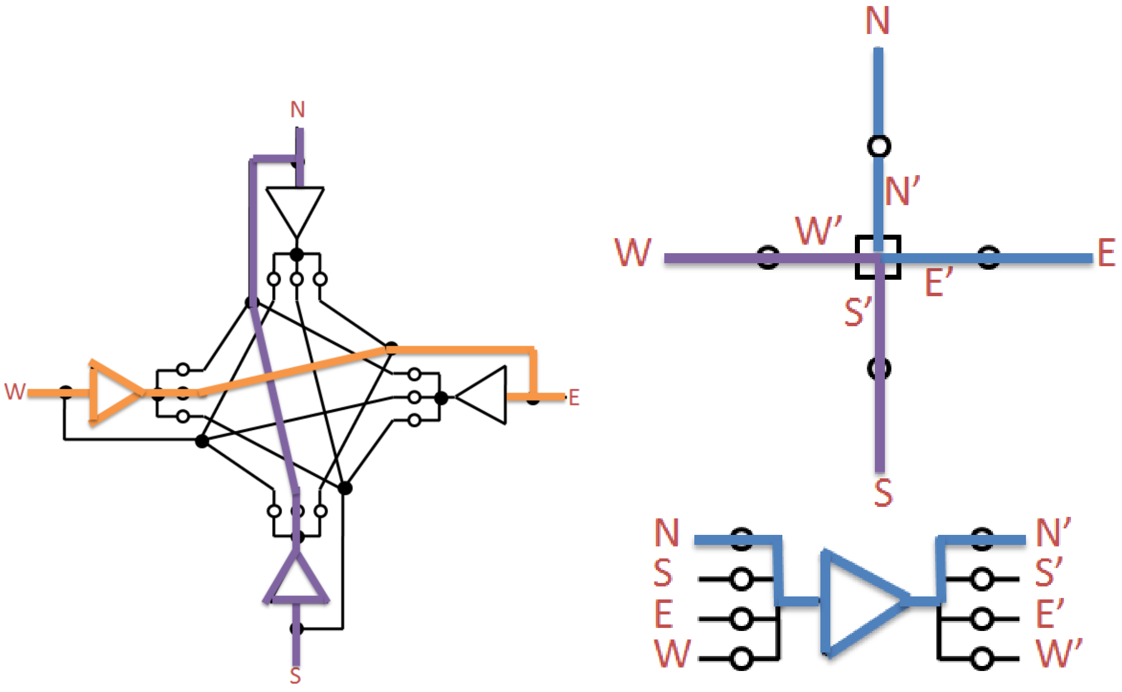


Figure 59. The global interconnect comprises vertical and horizontal track segments isolated by S-Blocks. The S-Blocks allow signals on tracks to propagate to neighbor tracks or to change directions. The C-Blocks provide connectivity from the global tracks to the primary inputs and outputs of the CLBs and CABs. Examples of allowable routings shown highlighted.

is also a primary output, then the output multiplexor can handle this. The only time output shorting will fail is if two devices are to short their outputs, and this net does not propagate out of the CAB or to the input of any device in the CAB, and all CAB output lines are already occupied with other nets.



(a) (b)

Figure 60. a) The s-switch topology used in the digitally buffered s-switches. b) The analog buffered s-switch topology. Some examples of allowable routings shown highlighted.

6.2.4 Global Interconnect

The global interconnect follows a standard track based scheme with C-Blocks getting inputs and outputs out of the CABs and CLBs and onto the tracks. S-Blocks allow track segments to be connected across, or to make turns. Figure 59 shows a two by two array of tiles where each tile contains either a CAB or CLB and global interconnect: two C-Blocks and an S-Block. There are 11 tracks in the north-south direction, and 11 in the east-west direction.

The C-Blocks in the FPAADD are implemented as a completely populated floating-gate matrix. The C-Blocks are not fractional, all inputs and outputs from the computational blocks have access to every track, and all track segments span one tile length.

The S-Blocks are a diagonal arrangement of s-switches (one buffered, ten passive) that allow signals to propagate across or to change directions into neighboring tiles, but the diagonal nature keeps the signals on the same track number as they started. The standard s-switch is implemented as shown in Figure 55e, which passively passes both analog and digital signals. Every s-switch is of this passive type except for the bottom left ones on the first track.

These s-switches on the bottom track are buffered. Each digital tile's S-Block has a single digital buffered s-switch and each analog's has a single analog buffered s-switch. Two different buffered s-switch topologies can be seen in Figures 60a and 60b. Both circuits are bi-directional, and allow for the same direction choices of signal propagation as the passive s-switch. The first circuit uses significantly less switches, has less internal parasitics per track, forces all entering signals to leave buffered, and requires four buffers. The second circuit is basically a passive s-switch with the ability to insert a single buffer on the input from one of the directions. In general, the first topology will be faster, but larger than the second topology. Because the analog buffers (a 9T floating-gate programmable OTA based unity-gain buffers) are much bigger than the digital ones (two-stage inverter chain) we chose the second topology for the analog buffered s-switches and the first topology for the digital buffered s-switches.

Table 3 contains a list of specific parameter values used in the FPAADD.

6.2.5 Interconnect Comparison

The CAB devices, floating-gate design, and floating-gate programming infrastructure were all derived from the RASP 2.9a chip, a next generation FPAA from the line developed by Hasler *et. al.* [32, 31, 20]. Significant differences from the RASP 2.9a include the choice of a Manhattan style global routing architecture, and a feedback output local interconnect scheme. The global interconnect has significantly less parasitics over short distances than the RASP's global scheme, where global tracks span the entire length of the chip. There are buffers in the global interconnect whereas the generic RASP line contains none. The local interconnect of the FPAADD has 68% lower parasitic capacitance and 50% less parasitic resistance between routed CAB devices in

Array size	27x8: 108 digital tiles and 108 analog tiles.
Chip IO	33 generic IO pads, 11 digitally buffered bi-directional pads
Devices per CAB	2 FG-OTAs, 2 TGATEs, 2 Capacitors, 2 FETs (nFET or pFET), 2 MITEs
Devices per CLB	4 BLEs
CAB / CLB I/O	8 inputs, 4 outputs
BLE	3-input LUT, routable clock and reset, reconfigurable for asynchronous adders
C-BLOCK	11 Total tracks, all of segment length 1, fully connected connection blocks
S-BLOCK	diagonal with 1 digital or analog buffered s-switch per tile on the first track
Process	CMOS 0.35um Double-Poly, 4-M
Voltage	2.4V at runtime

Table 3. FPAADD specifications

the local interconnect (devices in the FPAADD can be connected with one switch, but in the RASP chips require two at minimum) at the cost of slightly decreased routability described earlier. This leads to a 4x improvement in bandwidth of signals routed in the local interconnect over previous RASP based FPAAs. In the RASP line, CAB devices were disconnected from the routing infrastructure during program time with large transmission gates in order to not expose the devices to injection level programming voltages, but with careful circuit consideration, these can be removed in almost all cases.

The global interconnect scheme as well as the local interconnect and CLB devices draw heavily from standard Manhattan style FPGAs [33, 34]. Ignoring semi-arbitrary design decisions regarding architectural parameters, such as number of tracks, cluster size, placement of buffers, etc. the digital tiles look very similar to previously made FPGAs. The biggest difference being replacing the switch elements and SRAM with programmable floating-gate pFET transistors[32, 31, 20].

Floating-gates are very similar in operation to non-volatile technologies such as

EPROM, EEPROM, and FLASH; various FPGAs and CPLDs have been built using these technologies [35, 36, 34]. The floating-gates transistors in the FPAADD are built in a standard CMOS process. They have a higher dynamic range of programmed voltage leading to significant performance increases in power, speed, and signal integrity at the cost of density compared to conventional EEPROM and FLASH devices.

In [34], they claim that switching from using the EPROMs as the actual switch to simply using the EPROMs to control the gate of CMOS devices, that a 10x speedup was achieved. This is similar to the problem that pass-transistor logic, often used in FPGAs, face when trying to pass a logic-level (VDD for nFETs and GND for pFETs) that causes the devices to enter subthreshold before completely passing the signal. This results in logic level high voltages of about one threshold voltage less than VDD after reasonable amounts of time, usually leading to speed degradation and an exponential increase in leakage current in gates driven by these logic levels. Because the floating gate voltage can be programmed to higher than one threshold voltage above VDD, the switches stay in above threshold while passing the whole rail-to-rail voltage. Small signal resistance sweeps in [31] show floating-gate switches being as good as transmission gates but at half or better parasitic capacitance.

6.3 CAD Software

Much work has been done in the realm of synthesis for FPGAs; many software packages are available from industry and the open-source community alike. The field of synthesis for FPAAs, however, is far from mature. While the algorithms for placement and routing certainly have application to FPAAs, what does not translate so well are the cost functions (other than trivial ones like area and routability) to evaluate the desireability of routable solutions. FPGA synthesis is largely timing driven, where propagation delay models are used to identify the worst case delay of the critical path (further effort can be spent to then reduce the amount of devices on non critical paths for power optimization). While line delays certainly have some application to analog circuits, they are by no means the appropriate metric for all circuit nets.

In [37] the authors successfully apply standard placement and routing algorithms to map analog circuits to FPAAs with global parasitic reduction being the metric of choice. Next, extraction of parasitic elements is performed and back annotated to the initial input spice netlist for simulation, with fitness evaluation and iteration up to the user. The strategy in [38, 39, 40] is to partition the mixed signal reconfigurable system into the digital and analog subcircuits at data converter interfaces and apply different cost functions to each. Models for SNR estimation are developed that start with known device SNR and its degradation by connection topology of interconnect: cascode, fan-out, fan-in, and feedback. Bandwidth is also estimated using the data converter's Nyquist criterion as the bottleneck.

None of these approaches take into account the appropriateness of applying different cost functions to different net types. For instance, an algorithm that places negative weight on average parasitic capacitance of all nets will inefficiently try to reduce parasitic capacitance on nets that are insensitive to it, or that might actually benefit from it, like the outputs of DC bias generators.

The software suite, Verilog To Routing (VTR), was extended and modified to perform placement and routing on the FPAADD. As of writing, the flow is completely area driven.

6.3.1 Verilog To Routing

VTR is an open source, academic software suite that given an input Verilog circuit description and an input FPGA architectural description, performs synthesis and place and route (Figure 61). The suite consists of the following programs: ODIN II, which performs logic synthesis to standard cells (in this case, LUTs, FFs, and macro functions) [41], ABC which performs logic optimization [42], and T-Vpack and VPR which perform packing of LUTs and FFs into CLBs and then placement and routing [43].

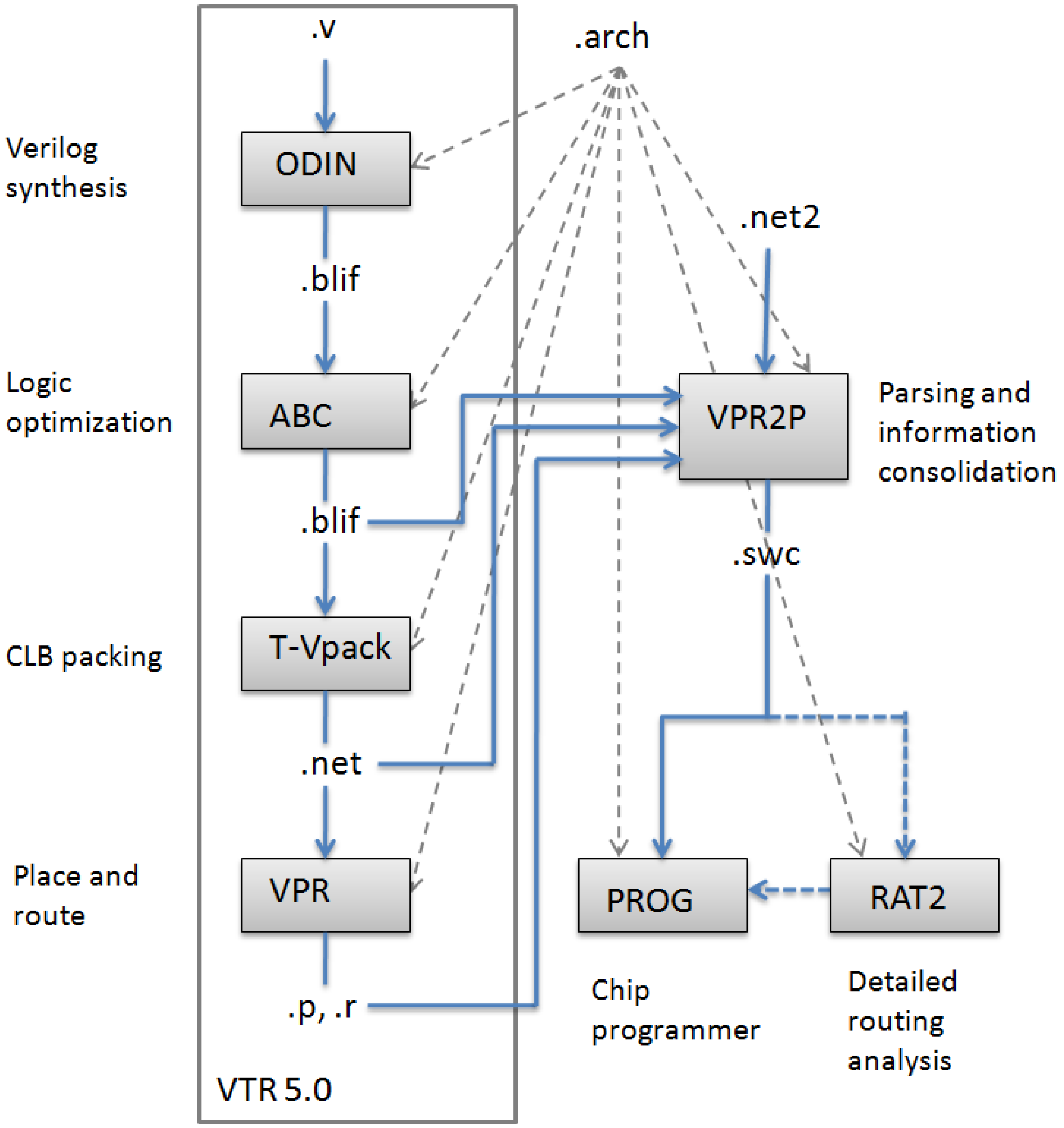


Figure 61. The software stack used for programming the FPAADD. From the VTR flow: ODIN takes an input Verilog file and performs logic synthesis targeting LUTs, FFs, and macro function blocks. ABC performs logic optimization. T-Vpack clusters LUTs and FFs into CLBs. And VPR places and routes the result. VPR2P takes an input describing the internal configuration of the CABs that are treated as black boxes in the VTR flow, and all of the intermediate outputs of the VTR flow, and creates a switch list. The switch list can be directly programmed or analyzed and modified by the detailed routing analysis tool, RAT2. All programs in the flow take various pieces of architectural descriptions of the target system.

While the flow supports synthesis of Verilog to the standard FPGA building blocks, it also supports the targeting of larger functions that may exist as dedicated hardware blocks on a heterogeneous FPGA. For instance, it is common to include hardware adders or multipliers in FPGAs as the synthesis of these rather common functions are often the bottlenecks in an FPGA implemented circuit design. In the same manner, hardcoded data converters could also be added to supplement the array for applications where synthesized converters are simply not efficient or accurate enough.

The support of black boxes made VTR a very attractive starting point in creating a software chain to provide placement and routing on the FPAADD. Digital circuitry could be synthesized all the way from Verilog while the analog circuitry could be treated as black boxes and simply placed and routed.

6.3.2 Routing on the FPAADD

While VTR will route circuits to arbitrary architecture graphs, it also supports a robust and scalable XML based architecture description language for quick graph building. Since the FPAADD was designed with a Manhattan style global and local interconnect scheme, describing the FPAADD in the VTR architecture language was relatively straight forward. Only a few minor modifications to VTR 5.0 were necessary.

The current flow starts with the circuit input as a blif and a net2 file. The blif file contains all of the digital circuitry as described as netlists of LUTs and latches with black boxes for the analog circuits. T-Vpack then packs the digital circuits into CLBs and the CABs are already prepacked in the net2 file. VPR then places and routes the CLBs and CABs.

The program VPR2P was written to take all of the intermediate file outputs of the VTR flow, consolidate the information, fill in the blackboxes with information from the net2 file, and then to translate the information into the corresponding physical switch locations on the FPAADD. The output is a row column switch list that is the input into our programming software. It also handles chip and board communication as well as the algorithms for erasing and programming the floating-gate memory elements.

Since VPR is not concerned with local interconnect, as routing at that level is deterministic, the GUI does not show the internals of the blocks. In order to analyze the detailed routing solutions (global routing and local routing) and to provide for a way to set and unset switches by hand, the RAT2 tool was created.

The RAT2 is a simple program written in MATLAB that can read in a switch list, display the routing solution, modify by means of a point and click interface the switch list, and dump out a switch list. This switch list can then be used to program the chip.

6.4 System Verification

The FPAADD as described in Section 6.2 was fabricated in a standard double-poly, single n-well, 4 metal CMOS 0.35um process. A die photo of the FPAADD is shown in Figure 62. All reported data are taken from the fabricated chip. The system is operated at 2.4V during run time, as opposed to 3.3V, to increase retention of the stored charge on all floating-gate transistors and to support legacy hardware that was designed to run at 2.4V [20].

All CAB and CLB devices, as presented in Table 3, are verified to be functional, via successful interconnect routing to I/O pads; global interconnect, local interconnect, and interconnect buffers are all working as expected. Simple circuits have been built: XOR gates and full-adders implemented in the CLB floating-gate based LUTs, asynchronous adders generated from FFs and LUTs, MOSFET threshold and characterization data extracted from transistor devices in the CABs, and ring oscillators built out of the buffered global interconnect. Verification and programming of the the floating-gate transistors were performed and found to be similar from results reported in [20]. The design and layout of components for the CAB were re-used from previously designed FPAAs and performance metrics were found to be comparable to published literature in [20].

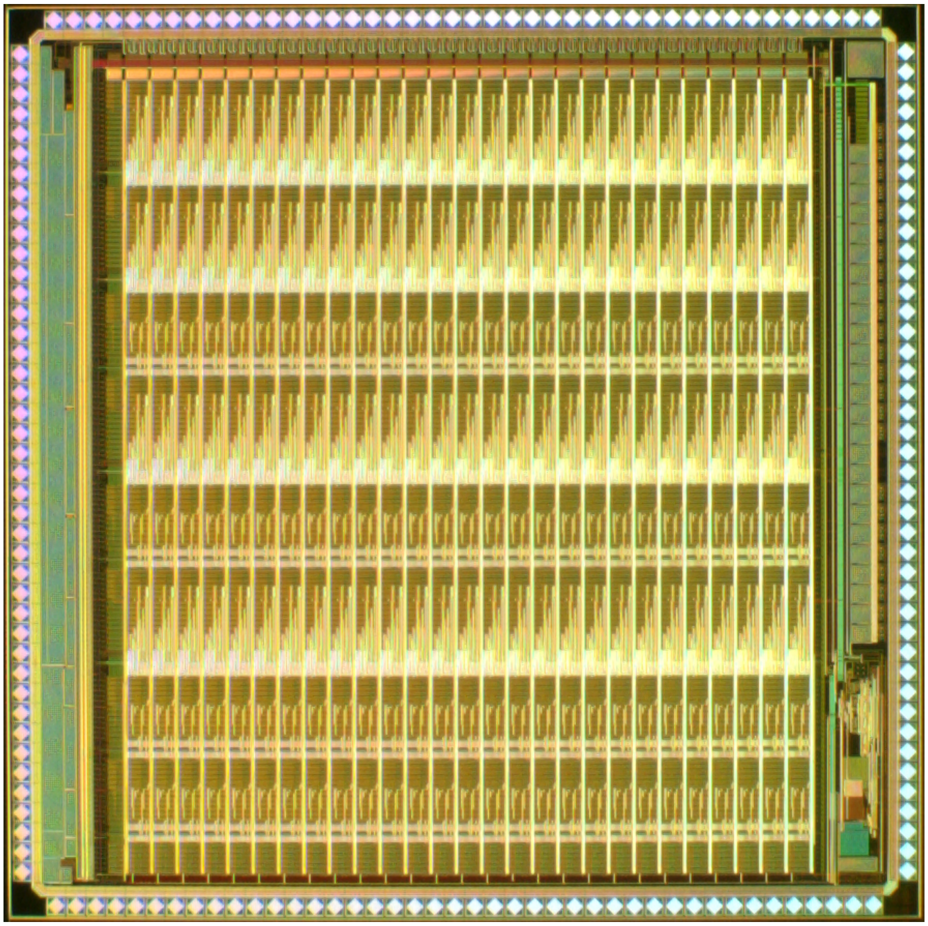


Figure 62. Die photo of the fabricated FPAADD.

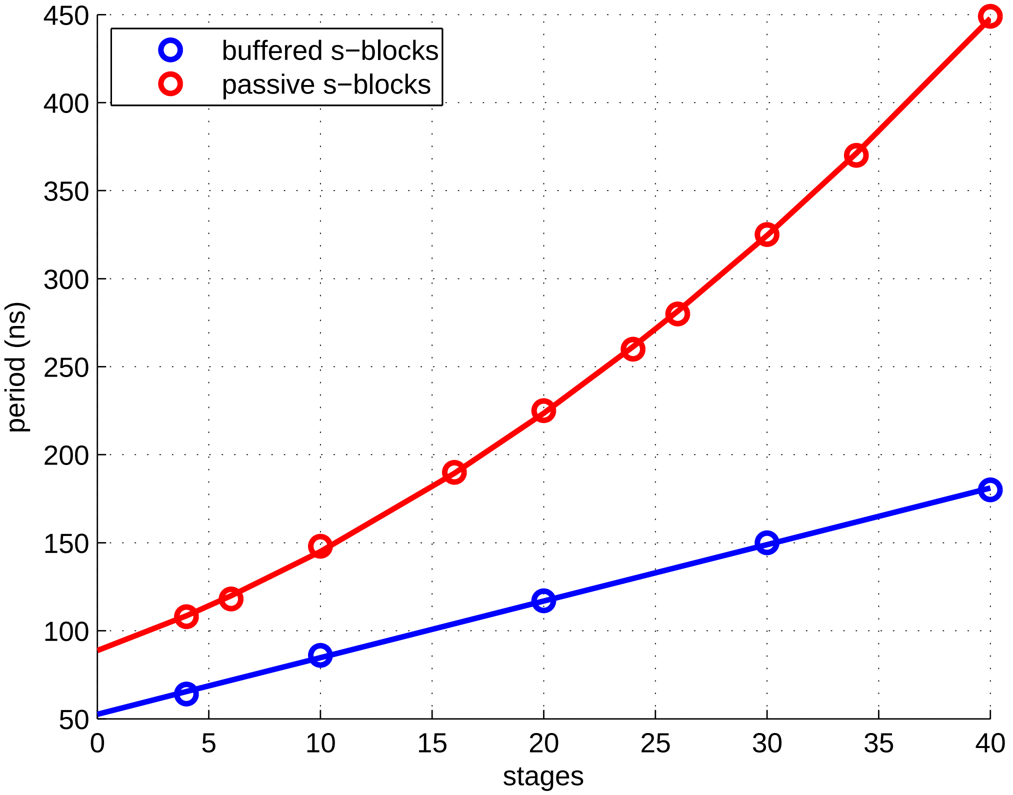


Figure 63. Ring oscillator period versus number of additional interconnect stages (s-block to s-block) for digitally buffered and passive s-blocks. The incremental delay due to a digitally buffered s-block is 1.6ns.

To evaluate the performance of the interconnect network, we measured delay as a function of routing distance (i.e. interconnect stages). Ring oscillators were implemented to perform this measurement, each interconnect stage being a C-Block and S-Block. Both buffered and unbuffered digital tracks are measured. In Figure 63, oscillator period is plotted as function of the number of stages. As expected, the delay of the oscillators using non-buffered tracks increases quadratically with the number of stages as is typical of RC ladders. The delay of the buffered tracks increase linearly. The delay of moving from one tile to the next through a digitally buffered s-switch is 1.6ns. Using a similar method, the BLE to BLE delay was measured to be less than 7ns.

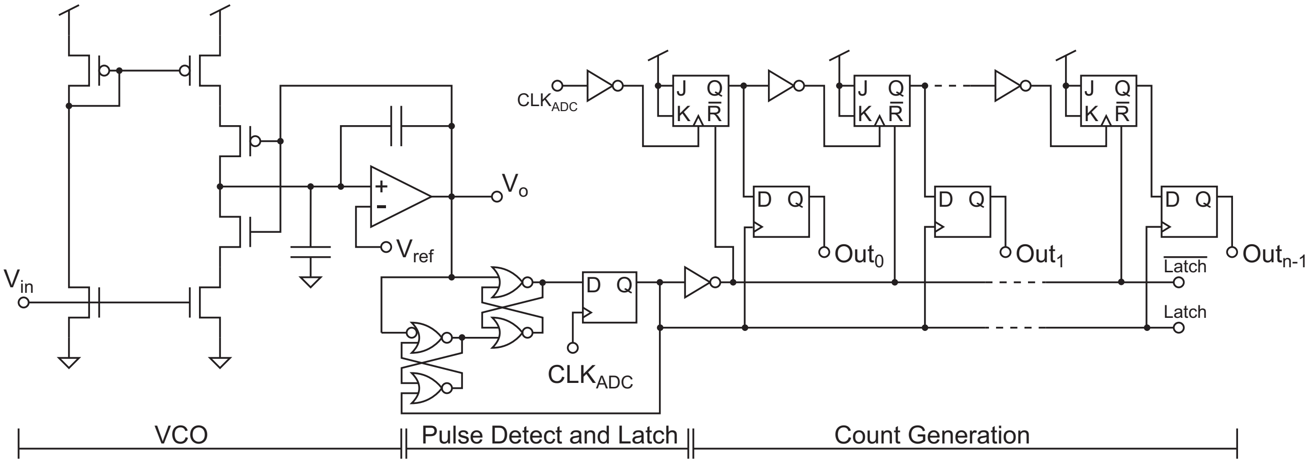


Figure 64. An 8-bit ADC built on the FPAADD. A current or Voltage Controlled Oscillator's (VCO) output period is measured by a digital backend.

6.5 System Applications & Results

Previous FPAAs have been used to build continuous time filters, vector matrix multipliers, AM receivers, analog speed processors, among others [19, 20]. The reconfigurable and mixed-signal nature of the FPAADD allows the user to address a variety of applications from pure analog to mixed-mode to pure digital including FPAA applications in previous literature. Two example system applications have been built to demonstrate the configurability and performance of the FPAADD: a VCO-based ADC and a 2^{nd} order low-pass sigma delta modulator.

6.5.1 VCO ADC

An 8-bit VCO-based ADC was built in the FPAADD as shown in Figure 64. The voltage controlled oscillator was built using discrete transistor CAB components; the asynchronous counters, state machines, and registers were built out of the CLBs. Figure 65 shows the frequency versus control voltage plot of the VCO. The linear dynamic range of the VCO was measured to be from 0.18 MHz to 7 MHz. The digital back-end was clocked externally at 2 MHz. However, the back-end was operational up to 18 MHz.

The ADC was measured to have no missing codes, and its operation can be seen in

Figure 66 for a 200.137Hz, 0.4V_{PP} input sine wave applied at V_{in}. INL and DNL data is not presented due to the non-linearity inherent in VCO based ADCs. The non-linearity of the ADC is due to the following effects: the voltage (V_{in}) to current converter is a simple nFET operated in sub-threshold, so an exponential voltage to current conversion is expected, while the rest of the VCO (a current controlled oscillator) performs a linear conversion of input current to frequency. The digital back-end counts the number of CLK_{ADC} transitions per VCO output pulse (V_O), giving a measure of the period for V_O. Using expected circuit behavior, the input is reconstructed from the output by fitting it to the following equation:

$$-ln[aT_{out} + b] = V_{out} \tag{47}$$

where T_{out} is the measured output, a and b are terms lumping subthreshold parameters of the input V-to-I input stage and the linear current controlled oscillator stage. The signal is then reconstructed from the ADC output and shows the circuit to be in excellent agreement with expected circuit behavior, as seen in Figure 66.

The VCO based ADC system consumed a total of 10 tiles (four analog and six digital) representing 4.6% of the total number of tiles in the FPAADD array. The percentage of device utilization within the six digital tiles was 88% while the utilization within the four analog tiles was 23%. Low element utilization of the CAB is due to the heterogeneous nature of the devices present within the CAB. The VCO used primarily discrete transistors found in the CAB along with an OTA and 2 capacitors leading to the low utilization value.

6.5.2 Delta-Sigma Modulator ADC

Figure 68 depicts the system diagram of a 2^{nd} order low-pass sigma delta created in the FPAADD. The low-pass filter was built using components from 2 CABs, and a single CLB is utilized for the D Flip-Flop. The poles of the loop filter are designed to be located

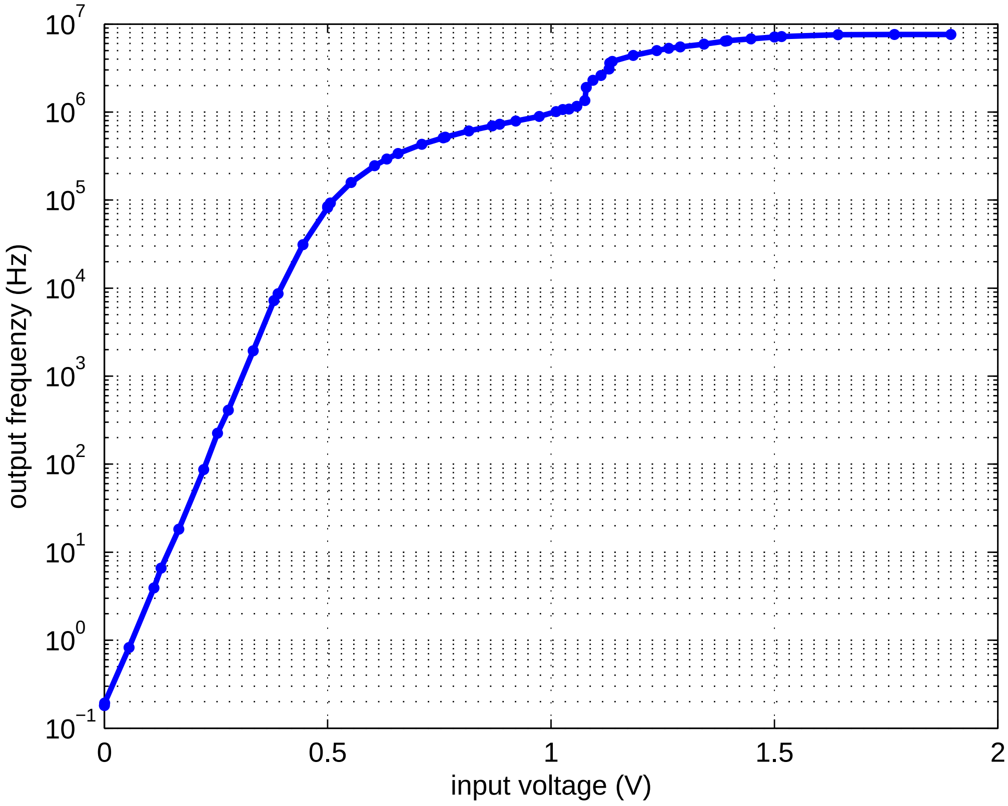


Figure 65. Measured response of the VCO over varying input voltage.

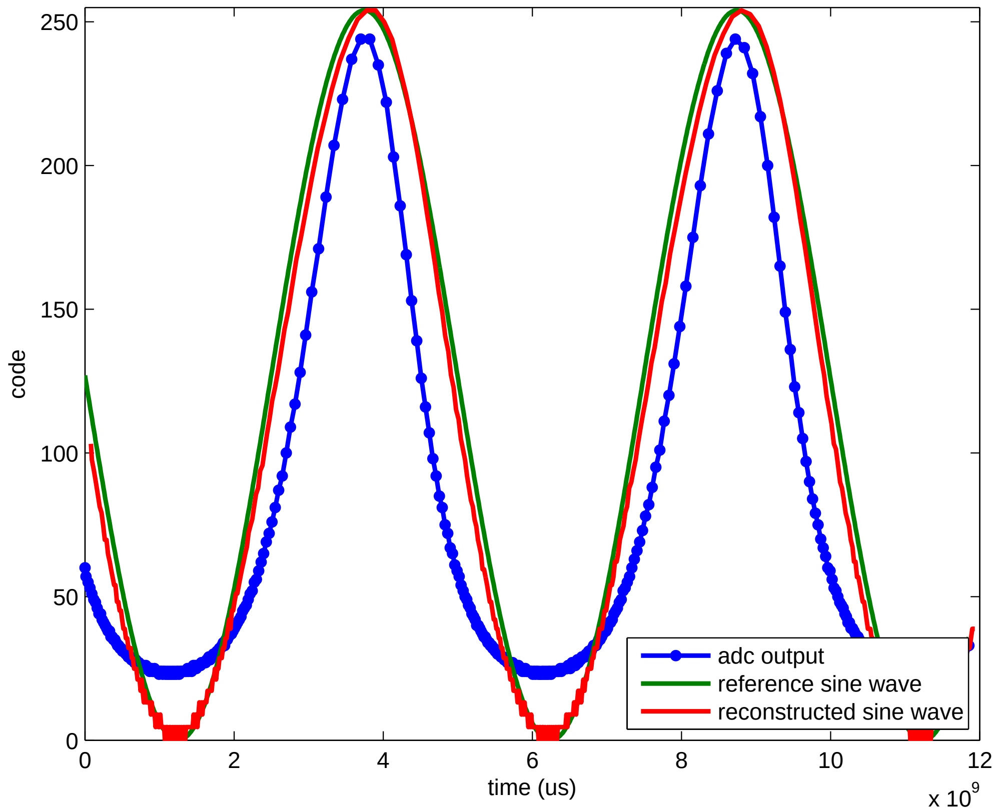


Figure 66. 8-bit VCO based ADC digital output (dotted line) for a 200.137Hz input sine wave of 0.4V_{PP} and the reconstructed input signal.

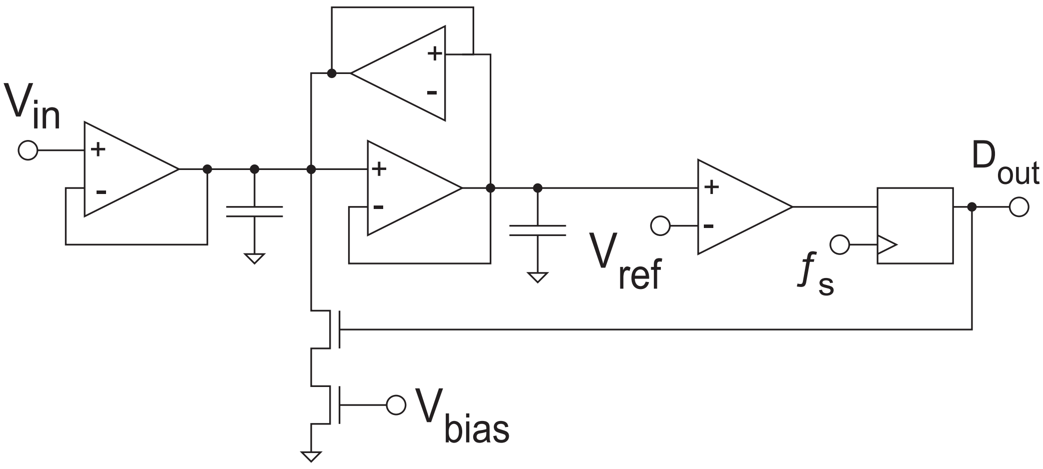


Figure 67. A 2^{nd} order sigma-delta modulator with 1-bit DAC feedback.

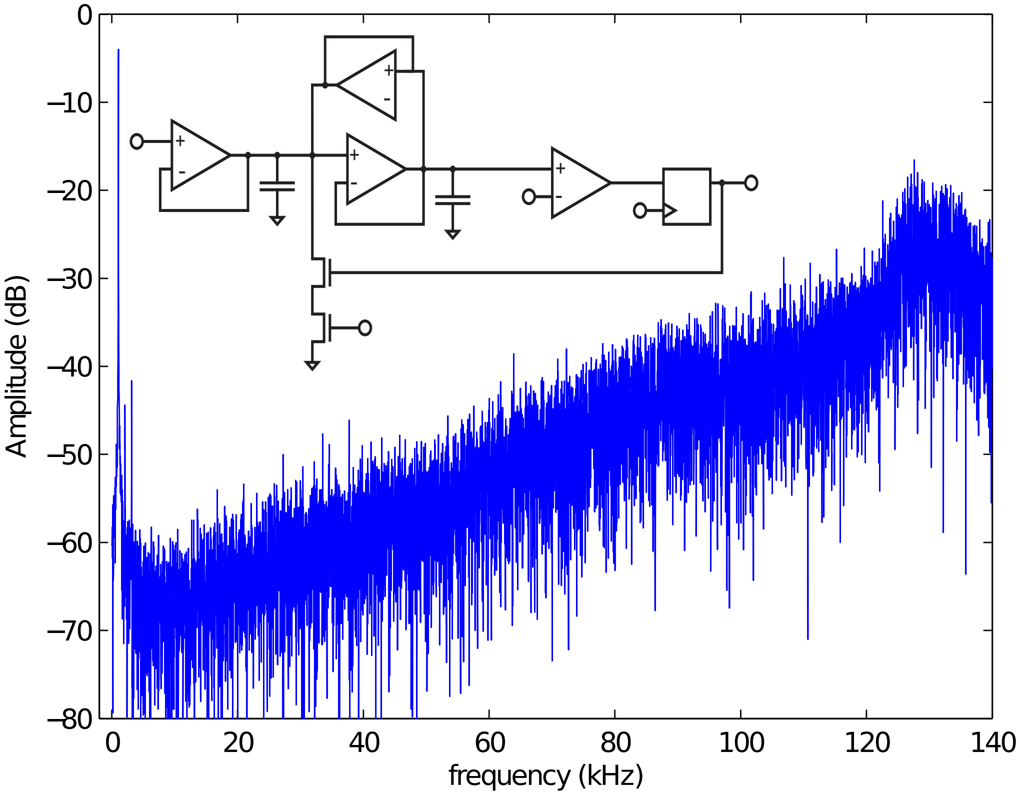


Figure 68. A 2^{nd} order sigma-delta modulator with 1-bit DAC feedback. Measured power spectrum for an input of 1.0478 kHz at 2.5 MHz oversample frequency.

at zero. The sigma-delta modulator has a measured SNR of 24.1 dB and SFDR of 39.2 dB at a bandwidth of 20 kHz and over-sampling frequency of 2.5 MHz. Figure 68 is a 32k FFT of recorded data taken from the FPAADD at the previously stated input and sampling frequencies. Insufficient gain of the loop filter is the probable reason for lower than expected SNR. Further optimization of the loop filter is required to increase the SNR.

6.6 Conclusion

A mixed-signal heterogeneous tile array (FPAADD) of CAB and CLB components has been built and presented. Verification testing of the system was performed at the component, tile, and system level. Initial results of the FPAADD display 7ns BLE to BLE performance and 1.6ns buffered tile to tile delay. Oversampling ADCs were implemented to test the functionality of the tile array and show the reconfigurable nature of the chip. The next stage of research will further characterize the FPAADD with emphasis in system scalability, power and noise analysis, and optimum partitioning of analog/digital functionality. This will allow realization of larger systems that take full advantage of all the computation properties. The goal of the FPAADD is a bridge towards embedded systems containing the reconfigurability of a FPAA and digital processors, resulting in an embedded single chip reconfigurable solution to implementing complex systems.

CHAPTER 7
UNIFIED SOC FPAA ARCHITECTURES

The FPAADD chip spent about equal parts by area on reconfigurable analog fabric as it did reconfigurable digital fabric. This 50/50 area breakdown, and for that size of a chip, resulted in having a very significant amount of analog parts to build systems with, but was rather wanting in digital devices. Potential mixed-signal applications that we wanted to explore and implement on the FPAADD were often immeditately rendered impossible by the relatively small amount of FPGA-like fabric to implement the digital hardward in. Most, if not all, modern FPGAs have significantly more reconfigurable digital fabric than the FPAADD as well as contain hard-coded macroblocks for common but cumbersome to synthesize circuits: memory arrays, multipliers, etc.

In this sense I am referring to macroblocks as devices in the system that are significantly larger, not only in physical size, but in functionality than the most basic building blocks in the system. In the FPAADD, and FPGAs in general, the most basic digital building blocks are flip-flops and look-up-tables. Even these blocks are not the most basic one could choose for a reconfigurable array. Making the basic building blocks out of pFETs and nFETs would do the job. In fact, this would go a long ways towards being able to implement any analog or digital CMOS system. Simple logic gates or amplifiers could be wired together from these devices, and then these wired together to implement larger systems. The problem, of course, being that this level of generality, while being practically as general purpose as possible in a VLSI system, would not actually be able to do anything particularly well. It would simply take too much area to really get anything done and the resulting implementations would subsequently suffer from outrageous ammounts of parasitics, causing them to be very slow, and very power inefficient, and likely completely non-functional for non-digital applications.

The opposite end of the spectrum is the completely custom ASIC, with no reconfigurability, that solves one and only one problem, but does it incredibly efficiently and well. It is, therefore, a completely inherent tradeoff of reconfigurable systems that in their design one has to choose between generality of problems it can solve versus optimality of solutions. Classic FPGAs then eschew any notion of being able to implement analog systems, and therefore make their most basic building blocks out of look-up-tables and flip-flops. Which, admitedly, are still quite a bit higher functionally than the most basic digital building blocks that could have been chosen. One could build FPGAs out of an array of two-input NAND gates and interconnect. This would still be able to implement the functionality of any CMOS digital system (if the array were large enough), while being particularly more efficient than an array of pFETs and nFETs as building blocks.

Instead, many-input look-up-tables are used as the lowest level building blocks for boolean logic, and flip-flops are used for memory. A k-way look-up-table can implement any boolean equation of k or less input variables. And a single flip-flop can implement one bit of digital storage. Higher values of k allow for smaller mappings of very high fan-in logic to these building blocks, and can be very efficient when all inputs tend to be used, but end up being inefficient when few inputs are used. Also, arbitrarily large look-up-tables implemented as single gates quickly become slower than compound gate solutions. This is because gate delays tend to increase polynomially with transistor stack heights and linearly with the number of gates. So while circuits with fewer transistors tend to be smaller and more energy efficient, it is not always the fastest implementation. Common numbers for k tend to be in the range of 4-6.

Since absolute generality tends to be poor at doing anything, and no generality has no flexibility, it is then the job of the reconfigurable system architect to figure out what subset of the entire concievable application space to be targetable by their architecture, and then to optimize for that set of benchmarks. A good rule of thumb for this problem is that if all of the applications you want to target all have some common high level functions of considerable complexity, put them in the architecture. For instance, digital multipliers, DSP slices (multipliers with accumulators), and memory arrays are often added to FGPA fabrics. Even the most general purpose FPGAs will have blocks of the later, as implementing memory arrays out of flip-flops wired together through the interconnect network tends to be so area inefficient. Moreover, because memory storage is such a common requirement of digital applications, FPGAs without embedded SRAM arrays tend to be heavily restricted in the application pace that they can target. Since many applications that we wanted to implement on the FPAADD required much more memory than could be synthesized out of the fabric, we decided that future versions of the chip would have to have some sort of embedded SRAM.

So the next generation of RASP architectures became full SoCs. They switched over to an FPAADD-like reconfigurable array, gained embedded SRAM, a hardcoded processor, and have a host of instrumentation and custom peripherals pushed on chip. This would result in one of the most advanced, complicated, and flexible mixed-signal reconfigurable SoCs ever attempted by an academic group.

7.1 RASP3.0 Architecture Family

The RASP3.0 architecture is used in a range of chips fabricated in 350 and 40nm processes. At the heart of all RASP3.0 chips is an FPAADD like reconfigurable array that is a fine-grained, heterogenous, interleaved array comprising varying complex block types. In all verions CABs and CLBs exist, but some versions of chips have more specialized complex blocks, such as the RASP3.0n (Complex Neuron Blocks (CNB)) and the RASP3.0rf (Complex RF Blocks (CRB)). The interconnect network is a two-level hierarchy comprising a very highly connected and dense local interconnect network internal to each complex block, and a light weight global interconnect for moving signals between the blocks. The general architecture for these chips is shown in Figure 69.

The name RASP3.0 refers to both a chip, the RASP3.0, and the architecture used

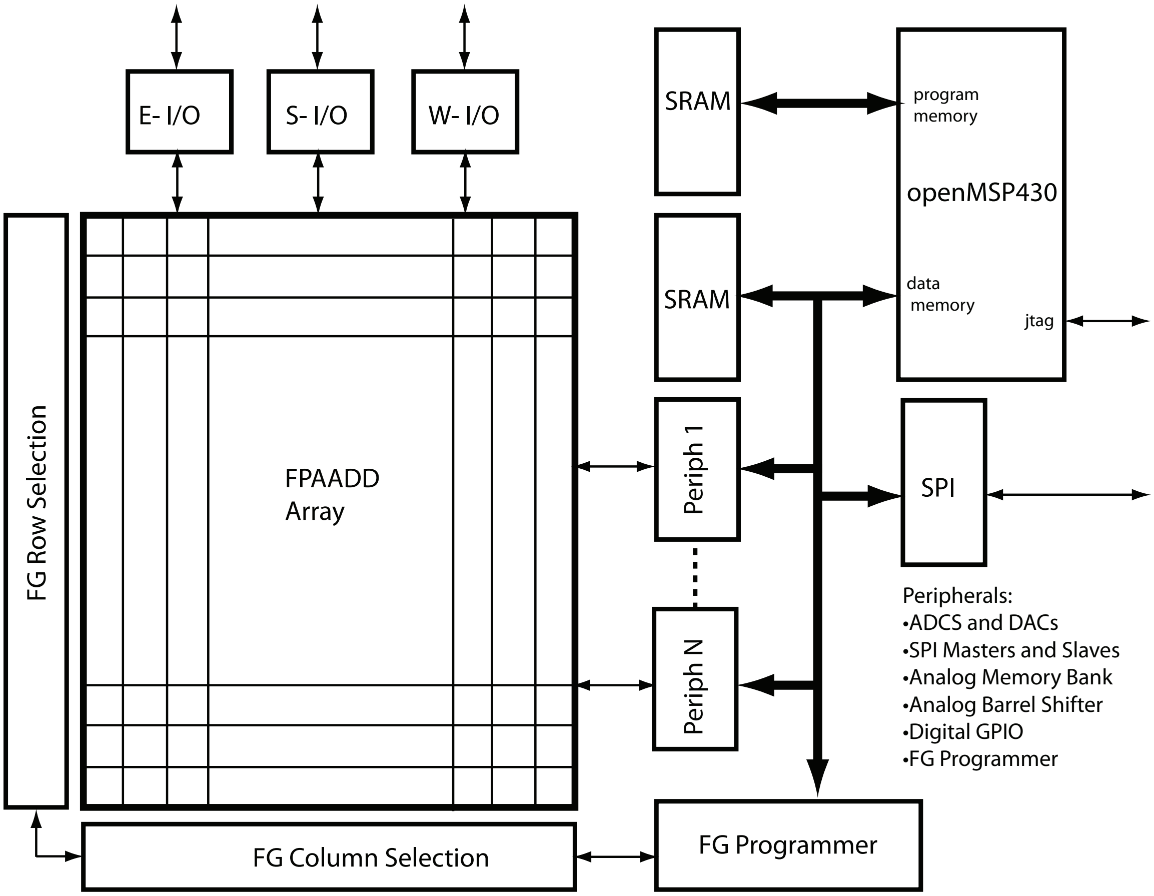


Figure 69. The general RASP3.0 architecture is an FPAADD array with integrated processor for onchip floating-gate programming control and runtime cocmputation and datapath control.

	RASP2.8	RASP2.9v	FPAADD	RASP3.0	3.0a	3.0n	3.0rf
CABs	32	78	108	98	49	84	42
CLBs			108	98	49	28	28
CNBs						63	
CRBs							21
Devices / CAB	10	10	8	13	13	13	11
Manhattan Style			Y	Y	Y	Y	Y
Buffered Interconnect			Y	Y	Y	Y	Y
Processor / SRAM				Y	Y	Y	Y
Analog Memory				Y	Y		
ADCs / DACs				Y	Y	Y	
RF Interconnect							Y
Die Size (mm^2)	9	23	23	84	54	84	6
Feature Size	350nm	350nm	350nm	350nm	350nm	350nm	40nm

Table 4. Comparison of various RASP chips.

in the following chips: RASP3.0a, RASP3.0n, and RASP3.0rf. Table 4 shows a comparison of features of the various flavors of RASP3.0 chips and some previous architectures.

These chips are all full-blown SoCs with integrated processors and SRAM. The inclusion of a processor and digital memory banks was to supplement and increase the amount of flexible digital processing the chips can implement, as well as to push the floating-gate programming algorithms and control on-chip. All chips use a modified version of the openMSP430 processor, an MSP430 instruction set compatable processor. In the 350nm versions of these chips, this processor was synthesized using a commercially available digital standard cell library and uses vendor provided SRAM intellectual property (IP) blocks. Because no IP blocks were available to us for the 40nm version, a custom digital standard cell library and SRAM blocks were created.

The processor is able to send information to and from the array through memory mapped I/O special purpose peripherals. These peripherals include ADCs and DACs, allowing measurements to be performed on chip, with the data taken by and stored in the processor, and communicated off-chip through an two-wire serial interface, or a multi-channel SPI port peripheral . Having a completely self-contained system for implementing circuits and taking data is very convenient, for instance, for using these systems in educational environments. This will also allow a significant speed-up to floating-gate programming and a reduction in power for any application that puts reconfiguration in the loop or algorithm.

However, the inclusion of the processor is not merely for convenience. As mentioned it is also there to supplement the processing power of the digital portion of the system and increase overall implementation flexibility; portions of a problem can be mapped to reconfigurable analog, reconfigurable digital, or a general purpose digital processor.

7.2 The RASP3.0 and RASP3.0a

The RASP3.0 is a large reconfigurable array comprising 108 CABs and 108 CLBs in an FPAADD like fabric, with embedded processor, memories, and a plethora of peripherals. This chip was fabricated in a 350nm process, occupying 12mm by 7mm of die area. The layout of the chip can be seen in Figure 70 . The RASP3.0 and RASP3.0a chips are identical except that the reconfigurable array in the RASP3.0a chip is about half the size of that in the RASP3.0

There is a 32 by 1024 sample and hold based analog memory. The sample window is variable, allowing the memory elements to also act as accumulators. It is designed to be accessed much like traditional SRAM memory banks. A row of 32 analog values can be read out or written to simultaneously with row selection based on a digital input address. The memory can be directly controlled by the processor, or custom configured

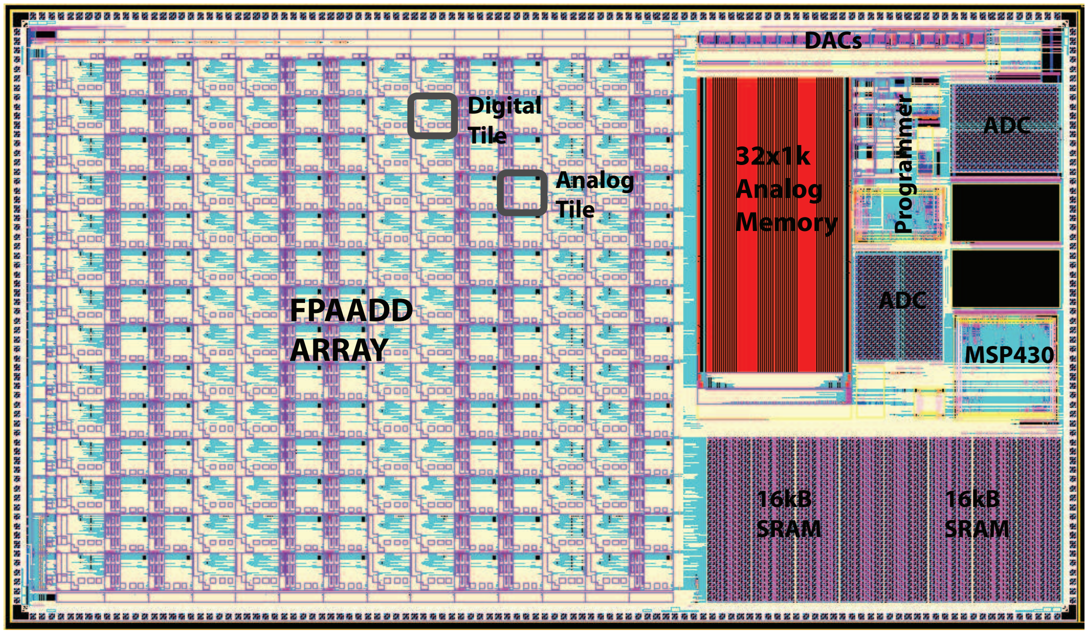


Figure 70. Layout of the RASP 3.0

digital logic from the array. The goal of this block is to be able to facilitate algorithmic analog computation without having to constantly go through data converters to use digital storage media. There is also a complementary 32 wide barrel shifter peripheral that can be used in conjuction with this peripheral to further alter access patterns. This analog memory and reconfigurable fabric can be used to explore a Von Neumann machine with a completely analog datapath and memory with digital dataflow control. Leveraging this idea and floating-gate based VMMs built from the fabric, a very novel computer can be built to tackle problems like image convolution.

While we have shown that data converters can readily be synthesized in the FPAAD like fabric- flexible in number, size, and topology- several dedicated data converters exist as peripherals on this chip. There are 16 8bit DACs, and two 50MSPs 8Bit ADCs. These can be used in the datapath for mixed-signal applications, or can simply be used as instrumentation controlled by the processor.

Target applications will initially include image transforms, synthesizable data converters, PLLs, frequency synthesizers, PWMs, and analog datapaths with digital control. A particularly lofty goal might include handling the details of robot navigation and control: high level decision making being done by programs running on the processor, using image tracking data as computed through analog image transforms built from VMMS in the routing fabric, analog data storage in the analog memory, and datapath control by state machines in the digital fabric. Based on these decisions, servos could then be driven directly by PWM generators synthesized from the array.

7.3 Direct and Indirect Switch Programming

To facilitate the creation of very accurate VMMs, some switch matrix arrays in the fabric are implemented as directly programmed floating-gates.

When synthesizing analog VMMs out of a switch matrix, the coefficients are implemented as effective threshold adjustments through accurately modifying the charge stored on the floating-gates. Because of this, mismatch between the injection transistor and the circuit transistor reduces analog VMM accuracy. Some of the CBLOCKs are implemented as directly programmed floating-gates to facilitate the synthesis of highly accurate analog VMMs out of the routing fabric. This idea was originally implemented in the RASP2.9v.

The difference between directly programmed and indirectly programmed floating-gates is whether or not current measurements are made on the circuit transistor or the injection transistor during the programming algorithm (Figure 71). In the direct case, both the circuit and injection transistor are the same transistor. But in the indirect case, they are two seperate transistors. The indirect case leads to a more efficient switch (less parasitic capacitance and resistance), but suffers from process variation mismatch causing precisely programmed switches to be less accurate. The direct switch architecture, while able to achieve higher bits of resolution in programmability, inserts

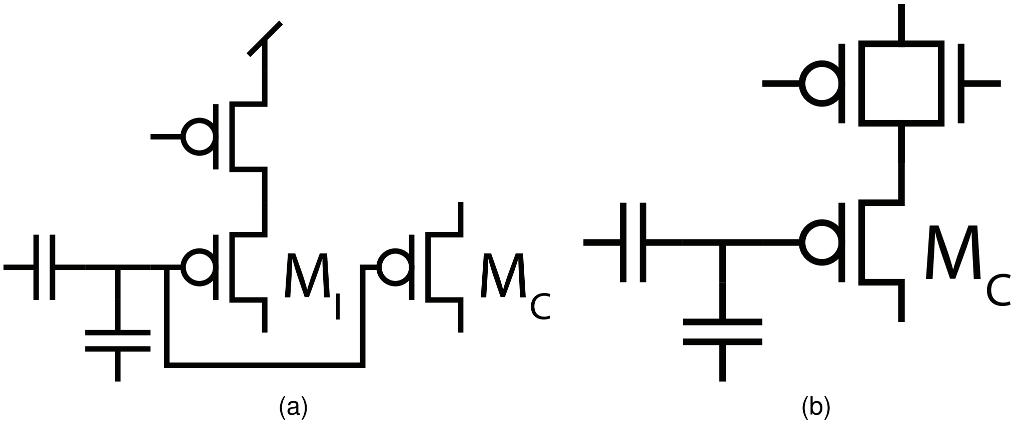


Figure 71. A) The standard indirectly programmed switch element: The in-circuit transistor, M_C, is programmed by the injection transistor, M_I. B) A direct programmed switch element. In this case the in-circuit transistor, M_C, is also the injection transistor.

a transmission gate in series with the floating-gate circuit transistor. This extra transmission gate is in the signal path during run time, tripling the parasitic capacitance, and more than doubling the effective series resistance of the switch. Making the direct switch significantly less efficient. It is for this reason that only some of the C-BLOCKs are implemented as directly programmed switches (DCBLOCKs). The top CBLOCK in every analog tile is directly programmed, all other CLBOCKs are indirect.

7.4 Volatile Switches as CAB Components

The RASP2.9v chip included some switches in the interconnect that were not controlled by floating-gates, but by flip-flop based digital shift registers. Because they are not controlled by floating-gates, we refer to these as the volatile switches. These volatile switches allow for a quick run-time reconfiguration of some of the switches in the array. This can be leveraged to build interesting devices: arbitrary waveform generators, data converters, etc, or can simply be used as a quick and simple way to access nets in a circuit for debug purposes.

In the RASP2.9v, the shift-registers were hard-wired together in chains. In the RASP 3.0 the shift registers are treated as generic devices in the CABs (Figure 72),

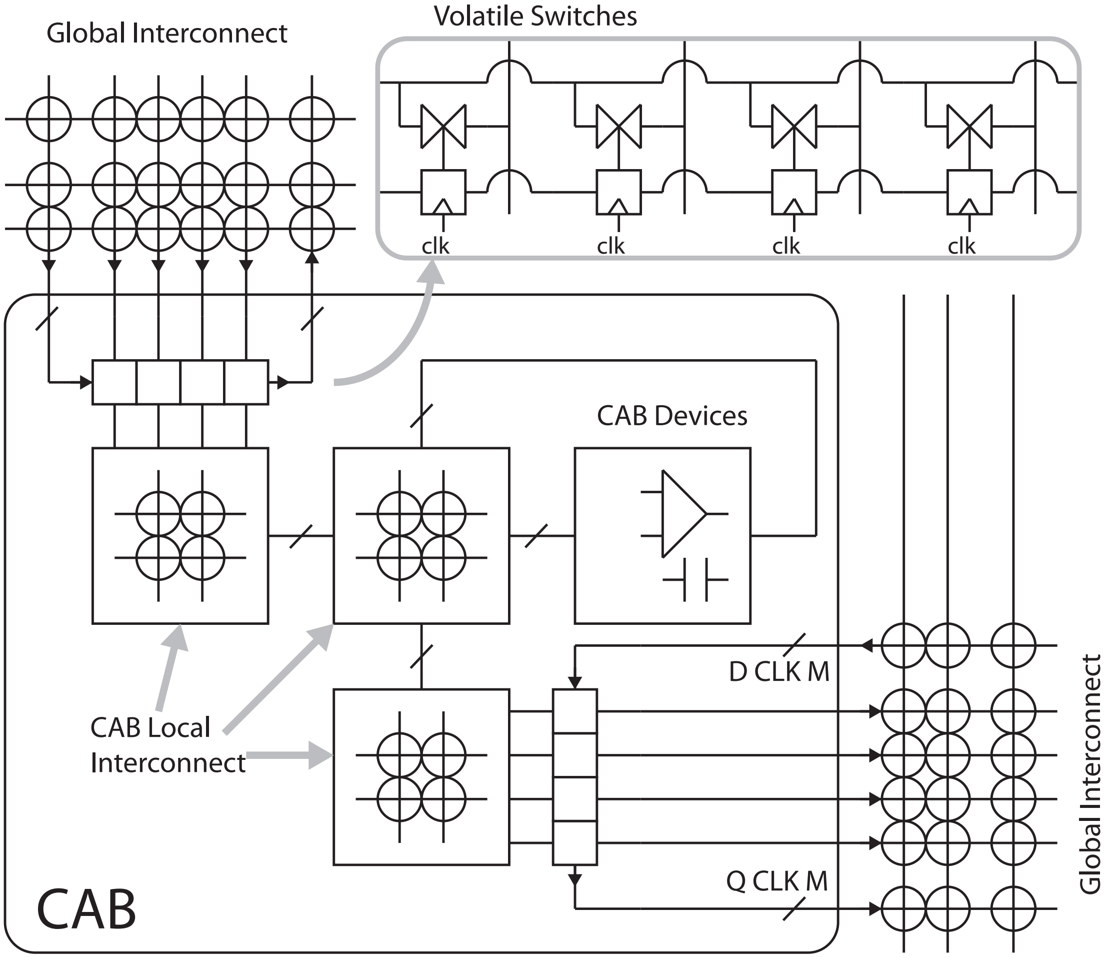


Figure 72

with all shift register control signals being locally routed inputs. This allows one to vary the number and depth of any shift register through programming, as well as imense flexibility in the detailed control of shift register operation: one register's output could clock another shift register, or clocking could be created from synthesized digital state machines, or be driven directly by SPI peripheral blocks controlled by the processor. By treating the shift registers as generic devicse in the CAB, we will be able to much more easily incorporate their use into high level synthesis tools, something sorely lacking from previous architectures. To facilitate the control of these volatile switches, a custom mult-channel SPI peripheral was created.

7.5 Floating-Gate Based Analog JTAG

Part of the floating-gate programming infrastructure is the ability to select any floating-gate in the system and measure the current through the indirect injection transistor stack. This is essential to feedback based precise floating-gate programming algorithms, floating-gate characterization, and floating-gate programming error detection. In previous architectures it was impossible to make this measurement without disturbing run-time chip operation. This has been relaxed in this chip to facilitate a new, interesting feature.

Figure 73 shows an indirect programmed switch. The transistor M_C is used in some arbitrary circuit, and M_I is the injection transistor for programming the floating gate voltage V_{FG}. Because V_1 and V_2 capacitively couple onto the floating gate, any measurement of I_M will be affected by those voltages. The floating-gate programming infrastructure allows the selection and measurement of any floating gate in the system.

Of course, comparing a runtime measurement of I_M with the program time current gives the lumped sum effect of how much V_1 and V_2 have moved from program time. This means that to infer exactly what V_1or V_2 is, another relationship needs to be found. Since most floating-gate switches are used to connect two nets together, and if the switch is doing its job well, then V_1 and V_2 should be nearly equal. One could always start at a node with a known potential, like vdd or ground, and work their way through the circuit, net by net, until reaching the desired net.

In this case, being able to measure these currents at runtime turns the programming infrastructure into sort of an analog equivalent of JTAG, allowing the user to probe any routed signal. Of course, this feature could also be leveraged to read out values from analog VMMs implemented in the indirectly programmed portions of the interconnect, making it not just a tool for debugging puposes, but also a tool for computation.

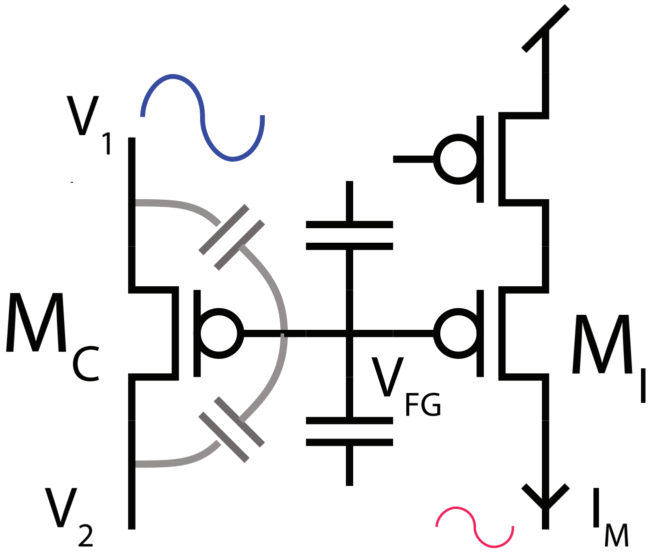


Figure 73. **Changes in V_1 and V_2 change V_{FG} through capacitive coupling. This modifies I_M. Comparing this current at runtime to program time can be leveraged to figure out what V_1 and V_2 are.**

This idea originated with a project that was building two-dimensional resistor lattices out of switch-matricies, where each resistor was implemented as a precisely programmed floating-gate transistor operating in the ohmic regime. Certain problems can be mapped to resistor lattices, where the solution to the problem is encoded in the voltages at the various grid locations in the lattice. While the actual resistor lattice is efficiently implemented in switch matricies on previous architectures, reading out the grid voltages was incredibly cumbersome. This analog JTAG would be a more efficient way to read out the grid voltages without the addition of any hardware.

7.6 The RASP3.0rf

The RASP3.0rf is a 40nm reconfigurable chip based on the RASP3.0 architecture. It was specifically designed to support high speed reconfigurable RF applications. To this end, the RASP3.0 array is extended and partitioned into two sections. One is the general purpose FPAADD like array comprising interleaved CAB and CLB based tiles for implementing baseband frequency computations. And the other portion of the array

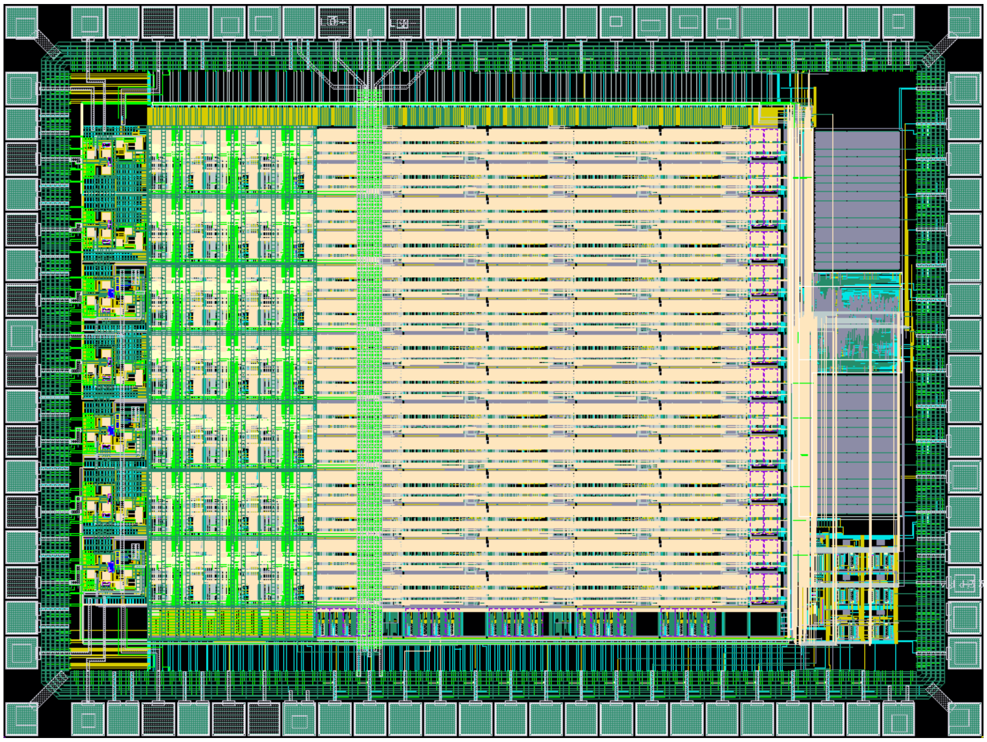


Figure 74. Layout of the RASP3.0rf.

is designed for routing high frequency RF signals and contains Complex RF BLocks (CRB). The chip also includes IO blocks containing low noise amplifiers (LNAs) for bringing in RF signals from the external world. Figure 74 shows a picture of the layout.

The CRBs contain devices useful in RF front-end applications: mixers, high speed amplifiers, capacitors, and non-overlapping clock generators. Like the regular array, the RF tiles containg both local and global interconnect arranged in a Manhattan style layout. Figure 75 shows the partitioning of the array into RF and baseband portions. The SBLOCKs are highly specialized to the task of routing reconfigurable and flexible delay lines through the RF array. In addition to the standard routing options that SBLOCK connections make, these also allow signals to jump to neighboring tracks and even return to the same CBLOCK. Inside the SBLOCKs are active inductor elements so that routed signals move along a reconfigurable unity gain delay line implemented as an L-C ladder.

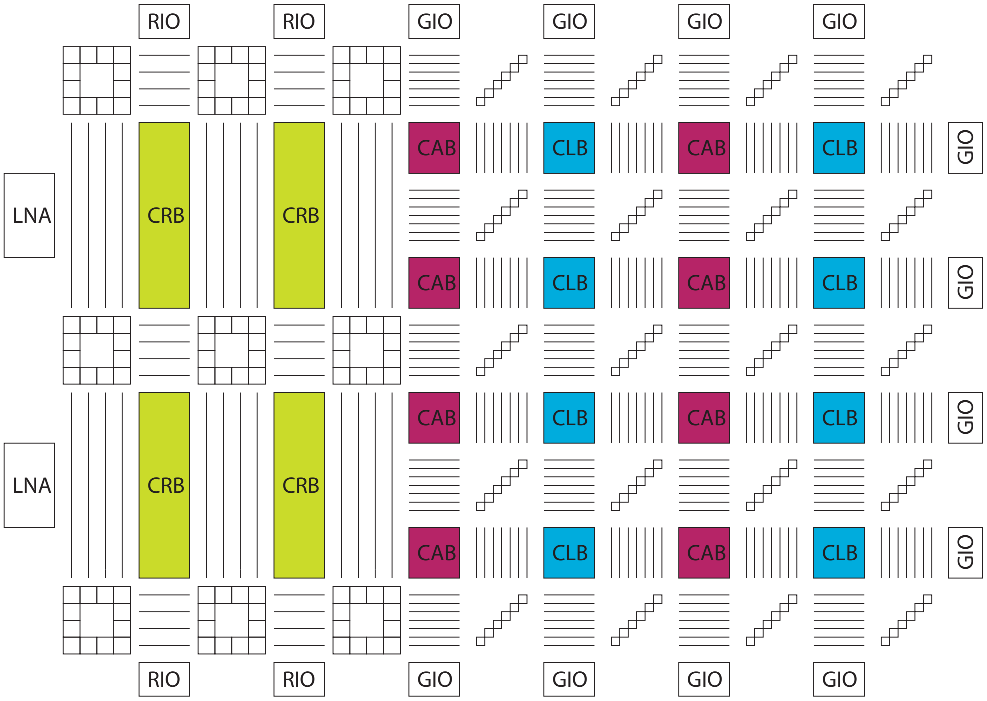


Figure 75. The RASP3.0rf array is comprised of a high speed RF section and a lower speed, general purpose section.

7.7 Reconfigurable Delay Lines

The RF array can be used to implement all sorts of reconfigurable frontend architectures for RF applications. Delay lines of arbitrary length can be built whose stages are tapped through switch-matrix based VMMs such that analog FIR filters can be built operating on the incoming RF signals. This allows some data processing to occur while the data is still in the RF domain, and before being messed with by the process of downmixing.

Figure 76 shows a portion of the RF array with some example routing options. This example shows an 12-stage delay line zigzagging through a couple of tiles in the array. Stages in the delay line are tapped through VMMs implemented from floating-gate switch matricies in the CBLOCKs. The figure shows three tappings. The first four stages in the delay line are tapped by two different VMMs, these taps are fed into separate mixers and their baseband output signals routed out towards the lower

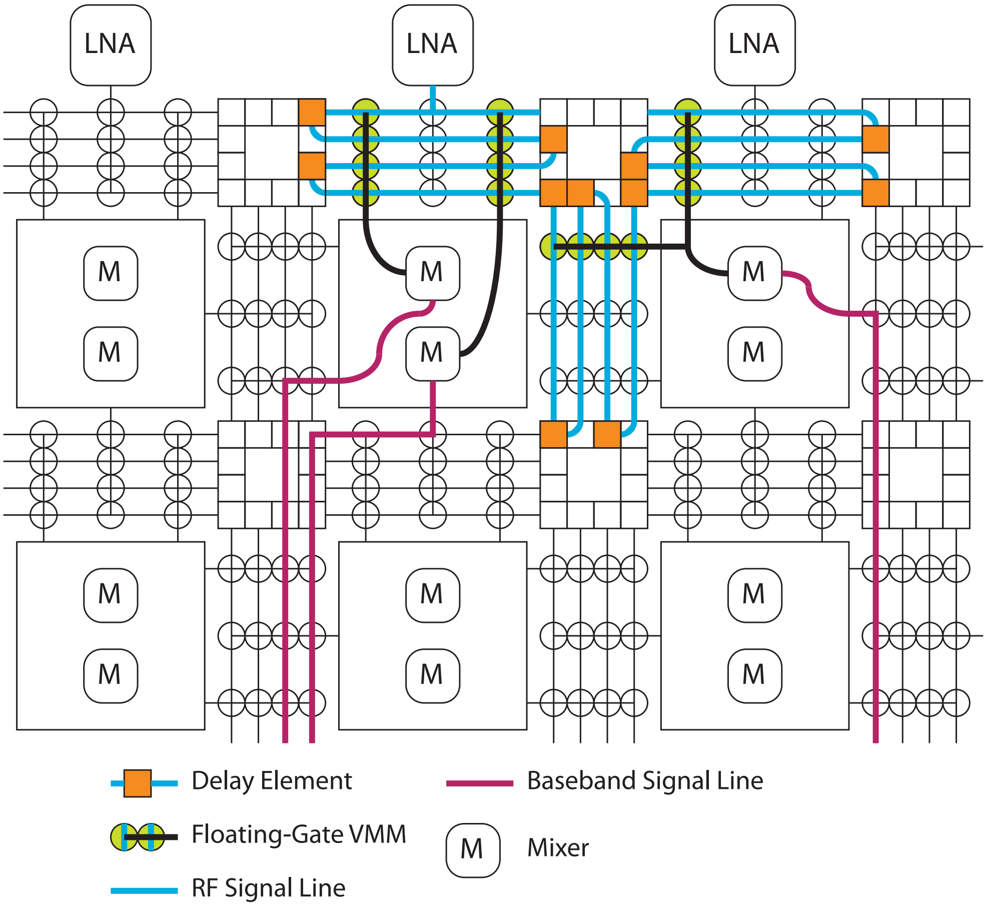


Figure 76. The RF array routing a many stage delay line whose values are tapped out through VMMs in the CBLOCKs, downmixed by mixers in the CRBs, and then the baseband signals routed out and towards the general purpose array.

speed array for further computation. The third taps the last 8 stages using multiple neighboring CBLOCKs and summing their outputs using local interconnect internal to the CNB, where the signal is downmixed and sent on its way.

Figure 77 shows the array configured to perform beam forming on multiple RF input signals. In this example a wavefrom would be sent at some angle towards three different antennas arranged in a line. These signals $x_0(t)$, $x_1(t)$, and $x_2(t)$ are sent into the array through three LNAs and each proceed along three different delay lines. Two different tapping of these delay lines are then shown approximating the equations

$$g(t) = x_0(t) + x_1(t - T) + x_2(t - 2T)$$

and

$$h(t) = x_0(t) + x_1(t - 2T) + x_2(t - 4T).$$

In this case the VMMs all have the same coefficients and act to simply sum their corresponding signals, though differing weights could of course be implemented. The amplitudes of equations $g(t)$ and $h(t)$ will vary with respect to incident angle of the wavefrom with respect to the antenna array plane. Each equation will have a maximum amplitude at distinct incident angles, effectively performing a bit of angle detection.

7.8 Custom Processor and Memory

The latest RASP chip architectures rely heavily on an onboard processor to handle the details of floating-gate programming, chip communication, and on-chip instrumentation control. The RASP3.0rf design is no different in this respect. Unfortunately, however, we were unable to obtain IP blocks for digital standard cells or SRAM, so we set out to create these ourselves.

Commercial digital standard cell libraries generally contain hundreds of standard cells. The one library we used to synthesize the openMSP430 in the 350nm RASP3.0 chips contained about 500 standard cells. This number is made up of permutations of many input gates and all of the possible logic functions they could implement. And having a large number of different drive strengths for each cell. The average number of different drive strengths is somewhere around four, with some cells having more drive options, and some less. Then having logic gates with upwards of eight inputs, its easy to see just how fast a library could grow. Some of these gates are compound gates, whose only benefit is a slightly smaller footprint than building the gate out of other cells.

A lot of work has gone into the study of library gate choices versus quality of synthesis results. In [44] they claim that going from a 200 cell standard cell library down to a 20 cell library generally produced synthesis results with trivially different worst case

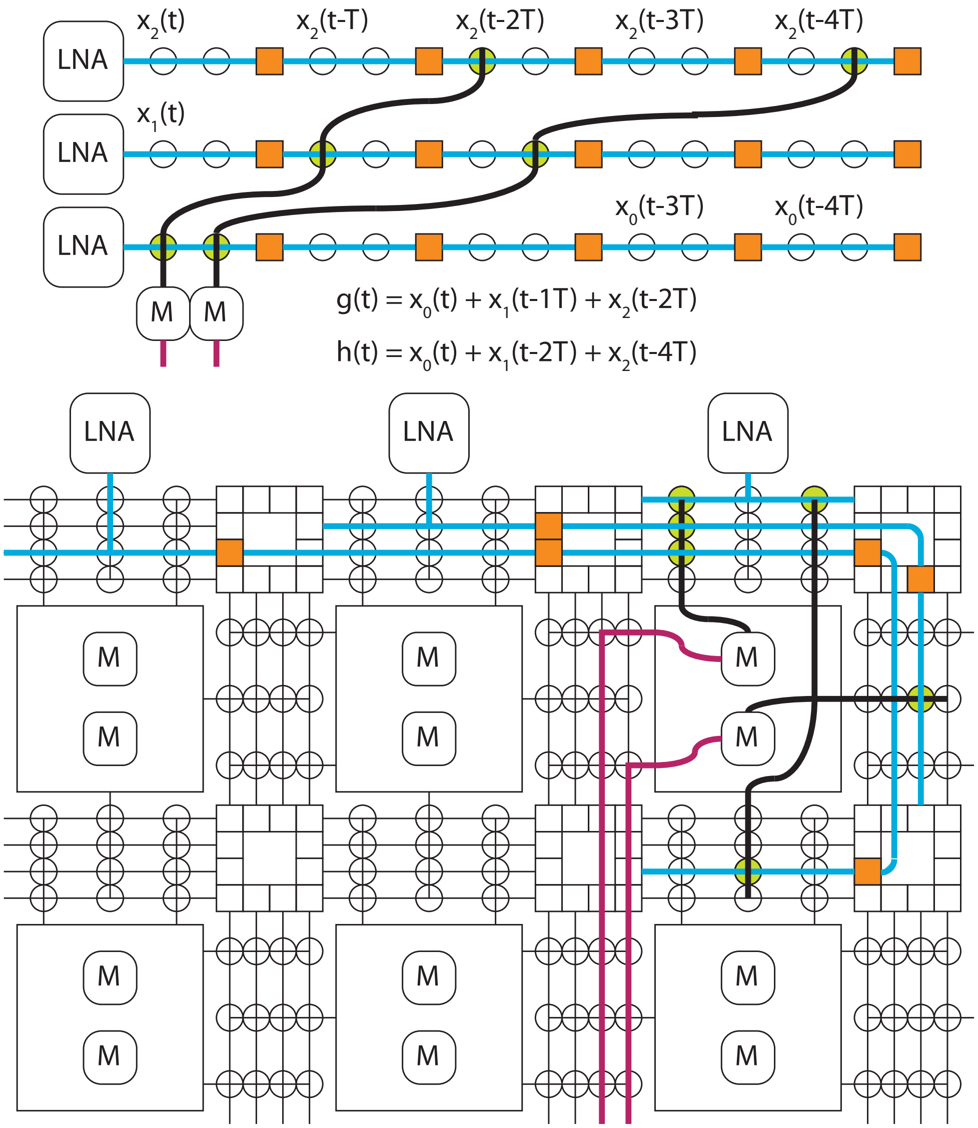


Figure 77. A beam forming circuit above and how it would be routed and implemented on the RF array.

single level	nand2, nand3, nand4, nor2, nor3, nor4
double level	aoi21, aoi22, oai21, oai22
triple level	aoi211, oai211
exclusive	xor2
inverter	inv1x, inv2x, inv4x, inv8x, inv16x
storage	dff, latch
filler	feed1, feed2

Table 5. List of gates in the cadsp_40nm digital standard cell library

critial path delays, and for synthesis results that varried between the two libraries, the increase in delay using the smaller library was generally less than 10%. Consequently, I designed a 22 cell library containing the gates listed in Table 5 . No compound gates were included, and all gates have only a single drive strength with the exception of inverters, for which five drive strengths were included. Initially only a single storage element was included, a simple D-flip-flop, however synthesis of the openMSP430 would not complete without adding a transparent latch to the library. All logic gates were sized for minimum average rising and fall propagation delays using a mix of simulation and hand analysis techniques. Then all gates were made four times the minimum width to reduce the standard deviation of fabricated delays from simulated delays due to process variations.

The library was characterized using Cadence Encounter Library Characterizer, and the openMSP430 was synthesized using Synopsys Design Compiler. Library design was iterated upon until the results of synthesis were successful and expected performace was reasonable. Then the library was laid out using Cadence Virtuoso (Figure 78) and abstracted using Cadence Abstract Generator, and the synthesized processor was placed and routed using Cadence SoC Encounter. The worst path delay through the processor was 1.2ns, and the layout was 300um by 200um.

The SRAM was built based on the design reported in [45] . The 8T SRAM cell was

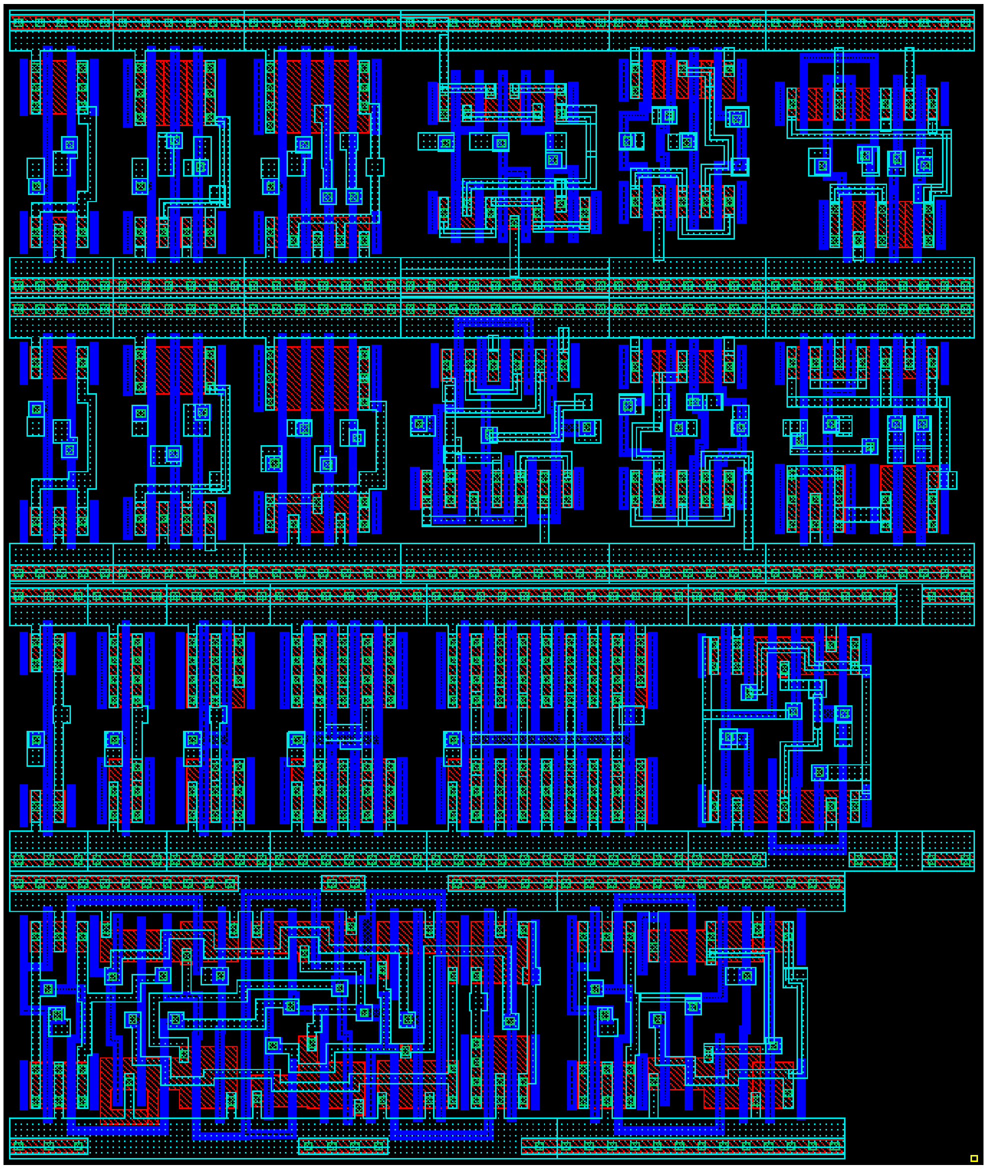


Figure 78. Layouts of all of the gates of the cadsp_40nm standard cell library

chosen over traditional 6T designs for reliability and ease of implementation. Two 8kB blocks were design and laid out to be used as the openMSP430's program and data memories. Figure 79 shows the layout of the openMSP430 with its SRAM arrays.

7.9 Floating-Gate Scaling to 40nm

There were several design challenges in migrating from a $350nm$ process to a $40nm$ one: designing analog circuits to work on a $0.9V$ rail, threshold voltage is about a third of the rail, transmission gates with poor midrail conductance, floating-gate leakage uncertainty, lack of double poly silicon, etc.

All circuits, analog and digital, were designed to operate with a $0.9V$ rail. This required significant redesign of some of out previous CAB components. Any time a transmission gate was needed, we opted to use an overdriven nFET instead. That is, an nFET whose gate is driven by a $3.3V$ rail source.

The floating-gates themselves had to be redesigned to use only moscap devices, as we traditionally use poly-poly capacitors for our control gates, and most modern processes do not have double poly. Much care was taken in the design of the floating gate layouts to produce the safest design possible, that is, the design most likely to retain charge. Test cells were created containing more aggressive and tighter layouts.

7.10 Results

The RASP3.0 was fabricated in January, 2013. It is back and currently being tested in the CADSP lab at the Georgia Institute of Technology. The processor is alive, and reliable floating-gate programming algorithms are currently being implemented on the openMSP430 processor. Unfortunately, at the time of this writing, no large systems have been implemented on the hardware yet. However, everything is looking very

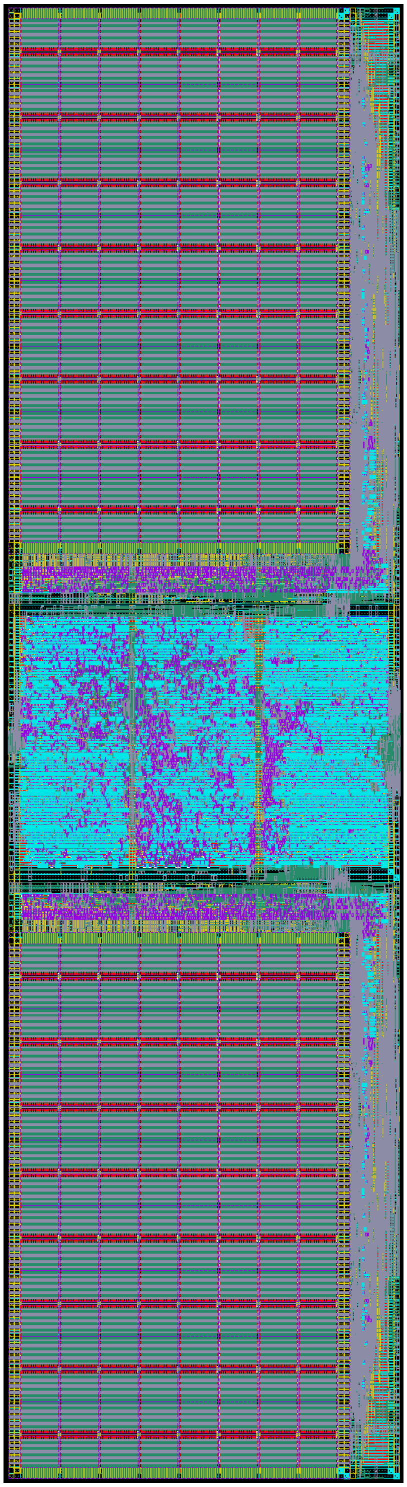


Figure 79. A custom 40nm openMSP430 and SRAM.

promising and we look forward to using this platform to explore and implement non-conventional computing solutions for years to come.

The RASP3.0rf was fabricated in November, 2013 and we have not gotten our hands on the chips to start testing yet.

7.11 RASP 3.0 CAD Tools

The new toolflow is a complete rewrite of the existing tools for the FPAADD and previous generations of RASP chips. SciCos replaces MATLAB as the graphical frontend, with sythesis, routing, and programming handled in custom Python, Assembly, and other open source tools written primarily in C and running on Linux. The new code base is faster, more flexible, more powerful, and easier to use than before, as well as relying only on open source environments.

A particular amount of effort has been put into being able to utilize routing resources as computational elements. All routing resources on the RASP chips are implemented by floating-gate transistors, and as such make very efficient on and off switches, but are also particularly well suited at implementing resistances and other non-linear two-port devices. Using routing resources as explicit devices presents a particularly difficult case to the place and route problem. As placement of devices into the interconnect changes the interconnect graph, the problem is fundamentally changed.

7.11.1 VPR

The VPR tool was designed to be a platform for simulation based FPGA place and route experiments. It was built to be an open-source academic platform for analyzing the efficacy of place and route algorithms in the mapping of benchmark circuits to FPGA architectures, the effects that varying the FPGA architecture has on the solution space, and the circuit performance of any routing solution. Being open source, algorithms, architectures, cost metrics, and benchmarks are easily swapped for large parametric and statistical experiments. It is particularlly well suited for asking questions like “If I were to increase the number of global tracks in my FPGA, how many new circuits would I be able to route versus how much slower would my results be?”, etc.

At its heart, VPR is simply a wrapper for applying off-the-shelf placement and routing algorithms to the problem of implementing a target circuit graph out of some subgraph of a target archiecture. In this sense, it can target a limitless variety of FPGA architectures. The VPR toolchain can be interrupted and any custom architecture graph could be inserted. However, the tool does implement a parametric architectural graph generator.

This generator provided a quick interface to building parametric FPGA architectures that were a subset of the architecture space it could target. That space was limited to Manhattan style architectures. The fabric was a linear array of tiles comprising complex logic blocks and global interconnect where the complex logic blocks contained the computational devices (LUTs and FFs) and some reconfigurable wiring called the local interconnect.

The local interconnect is used to wire devices together that have been clustered together into these complex blocks, and then those groupings are wired together at a higher level using the global interconnect. In this manner, some level of hierarchy is applied to the global place and route problem. First, the target circuit is partitioned into chunks that fit into complex logic blocks which, in the earlier versions of VPR were guaranteed to be deterministicly routable, and then this new partitioned circuit was placed and routed globally.

The global interconnect implements a way to route signals between complex blocks and across the chip. It consists of a set of track segments arranged much like the roads of Manhattan in a grid like fashion giving access to the buildings in between the roads. Cars or signals move through the city going straight or making turns at stop light junctions (SBLOCKs in the FPGA) or stopping and entering buildings (complex blocks) through CBLOCKs. As a city planner could change the number of lanes on the streets, the size of blocks and the number of buildings in each block, how stop lights work, etc. and it would still fit the same sort of general over all architecture, so does the architecture builder in VPR allow quickly making these trade-offs.

Since we have built a lot of variations of reconfigurable chips, it is nice to have a sort of unified code base for utilizing these chips that leverages as much code reuse as possible. So to this end, we have been restricting our architectural decisions to those that are more easily targetable by VPR's architectural building language.

7.11.2 VMM Synthesis and Fan-In Elements

VPR is far from perfect for our application. It was designed to target circuits comprising standard digital CMOS circuits implemented from look-up-tables and flip-flops. Circuits, and thus graphs, built from these devices have some patterns that can be exploited.

It is assumed that devices have ports that are either inputs and outputs, and that each device has at least one input and one output. Nets that connect these devices together then have some rules about what kind of ports can be connected to a single net: at least one input pin must be on every net, and exactly one output pin must be on every net (Figure 80). That is, it is forbidden to route circuits that have two device output pins shorted together. This of course would be an error in a CMOS digital circuit, where output pins represent strong drivers, and connecting outputs together would produce circuits that fight for control of that net, and thus be a violation of the CMOS requirement that for all possible logic combinations, only a single pull-up or pull-down network will be driving any net.

This, of course, is not a requirement of analog circuits. It is the very nature of multiple drivers fighting on a single net that allows an analog circuit to produce net voltages that are between the rails. Requireing analog circuits to be partitioned such that all device pins are labeled as inputs or outputs, and enforcing that no two output pins be connected together in a circuit is a bit unnecessarily restrictive. However, it is a restriction one must put up with if one wants to use VPR to route analog circuits. It turns out that a surprising number of analog circuits can be partitioned such that one does not violate these rules.

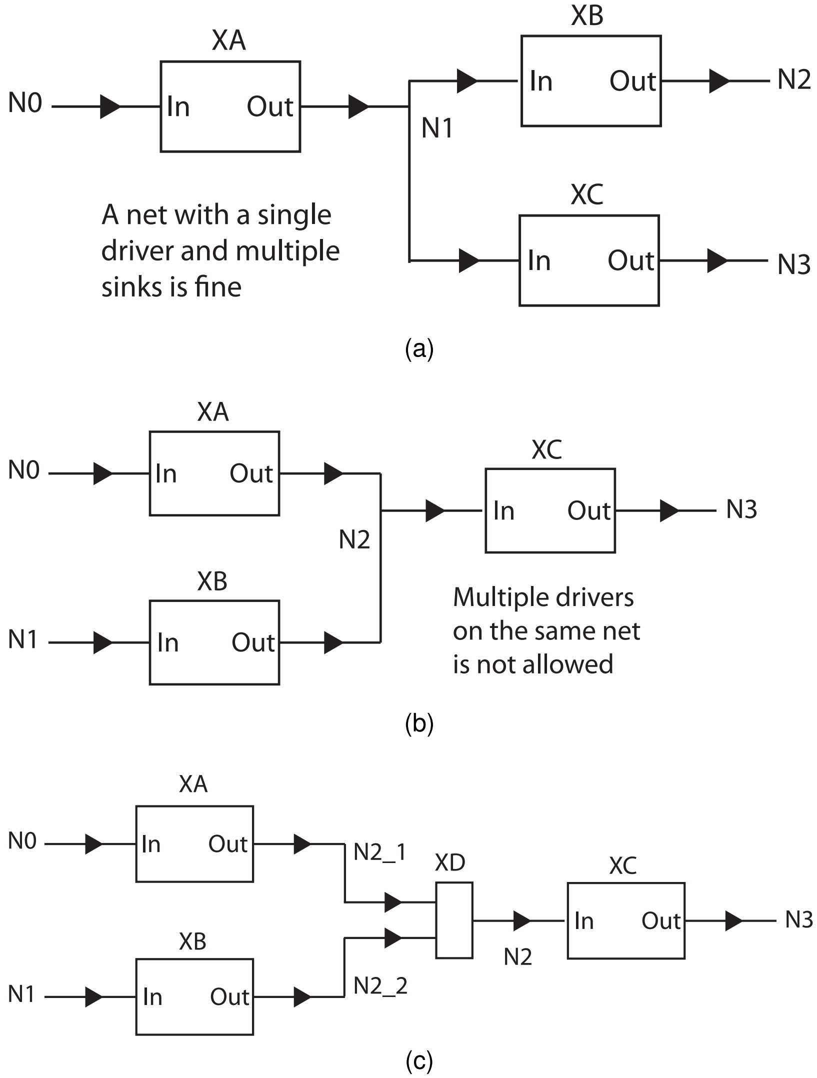


Figure 80. VPR treats all ports as either inputs or outputs and enforces some rules related to these port conventions. One particular rule is that a net is not allowed to have multiple output pins driving it. The topology in (a) is allowed whereas the topology in (b) is forbidden. The addition of a fan-in circuit (c) fixes the problem.

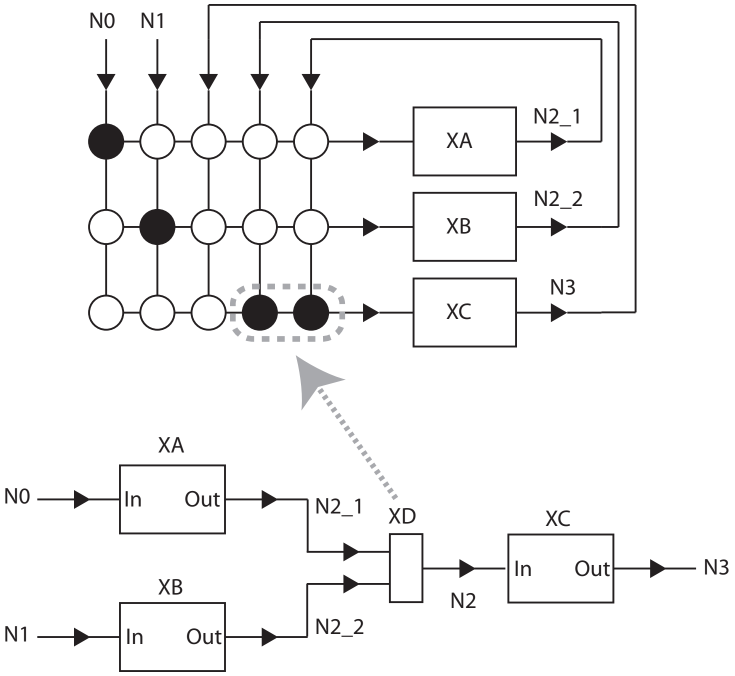


Figure 81. In this figure, open circles are floating-gates that are off, and solid circles are on. Here an extra floating-gate switch in an input row in the local interconnect is used to short two output nets together. The fan-in element is simply any row in the local interconnect. It has many inputs and only one output that is hardwired to a specific CAB device. Because of this, it can not support fan out at its output.

The multiple drivers on a single net problem is handled by explicitly inserting joining circuits into the netlist. These circuits are implemented as a single row in the local interconnect (Figure 81), and also provide a simple and easy way to insert floating gates from the interconnect as explicit computational elements into the solution.

The fan-in circuits have limitations: since these are implemented at no additional cost through targetting rows in the local interconnect as fan-in devices, this does impose some restrictions on where they are inserted into circuits and how. The biggest restriction is that the fan-in elements cannot support fanout, as their outputs are hardwired in the local interconnect to unique inputs to CAB devices.

This means for a net that has multiple drivers, but also goes to multiple inputs, a single fan-in circuit will not suffice. A fan-in circuit will have to be added for each connection from that net to the input of a device (Figure 82). For instance, a N-driver

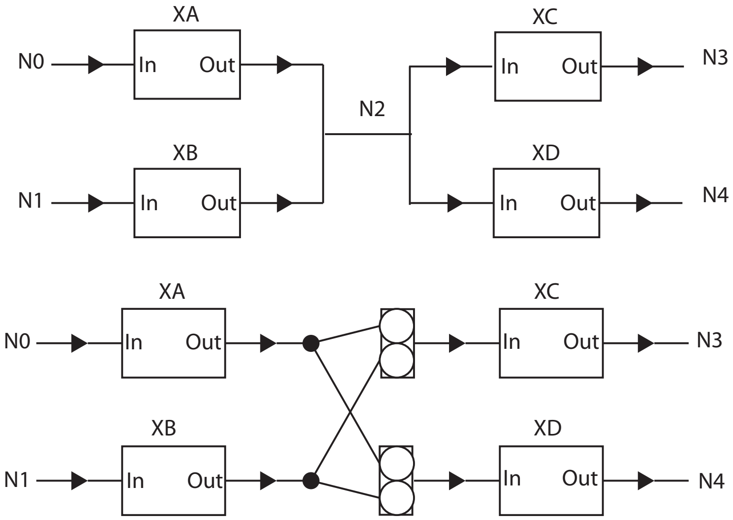


Figure 82. A net with multiple drivers and multiple sinks. Because the fan-in circuits do not support fan-out, fixing this net requires adding multiple fan-in circuits, one for each sink.

to M-sink net will require M fan-in circuits to be added, which, fortunately, does not reduce the efficiency of any routed solution, it just makes netlist generation a bit more tedious.

The number of inputs able to be consolidated by a fan-in circuit is also clearly related to the size of the local interconnect. In the RASP3.0, though the number of inputs to a CAB is 16, this number is functionally limited to 13 as some of those inputs are reserved for the flexible shift register control signals.

7.11.3 VMM with WTA circuit example

As an example I will show how to build and map a classifier circuit built from a vector matrix multiplier (VMM) and winner-take-all (WTA). This circuit will use a lot of discrete nFETs and OTAs from CAB devices, but will also heavily leverage using floating-gates from routing resources to implement the VMM itself and some feedback elements.

Figure 86 shows the whole circuit we will eventually be mapping, and we could map the whole thing at once, but since this section is supposed to be instructional, we will first map the WTA portion by itself. The WTA circuit that we want to implement is shown in Figure 83. It is a current mode circuit with $I_{IN,0}$, $I_{IN,1}$, and $I_{IN,2}$ as the input currents and $I_{OUT,0}$, $I_{OUT,1}$ and $I_{OUT,2}$ as outputs. The output currents are split from the same current source, with the largest input setting the common voltage, which tends to cutoff the currents in the other legs. The result is that the plot of output currents is sharply peaked around the largest input current. The details of this circuit's operation are not particularly relevant to this discussion. We just need to proceed to partitioning the ports on the devices as inputs and outputs.

Figure 84 shows such a labeling. In this figure a new arrow type is used to emphasize that the arrows are labeling port direction on the devices, and not a marker denoting current. Our goal here is to be able to eventually generate a netlist that is targetable to the RASP architecture, that means that all components in the netlist correspond to devices in complex blocks, and that there are no port related errors.

The nFETs in the RASP chips are implemented as three-port devices with two inputs and one output. The gate and one of the source/drain terminals are inputs, and the remaining source/drain terminal is an output. Because these source and drain terminals are interchangeable, there is some ambiguity to the possible mappings one can do. In Figure 84, the ground net is always an input to nFETs, but we could have labeled the circuit with those ports as being outputs. The reason for labeling it the way that it is was not actually arbitrary. Since the ground and vdd nets show up as inputs

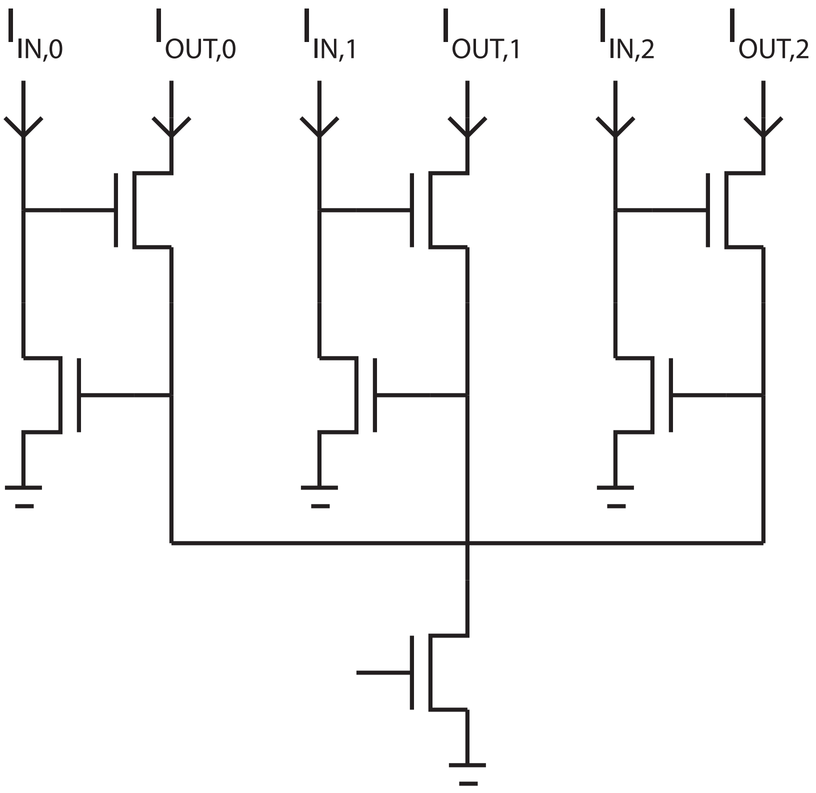


Figure 83. A simple winner-take-all circuit.

to the local interconnect, they are more immediately accessible as inputs to devices, requiring only a single switch to be turned on to tie these nets to any device input.

This labeling produces a valid set of circuit components that will map to devices in the RASP complex blocks. That is because all devices have valid port numbers and directions. If we had labeled one of the nFET gates as an output pin, then the circuit would not map, as the complex blocks do not have any nFET devices where the gate is an output pin. This labeling, however, does produce some nets with multiple drivers.

A visual inspection of the port-labled circuit shows that all input nets have this problem. That is, it can be seen that these nets have multiple arrows entering them. We will now add fan-in elements to the circuit at each of the problematic nets to fix this problem. Figure 85 shows the circuit with fan-in elements added. In each case, the offending net is broken into as many nets as there are drivers plus one with each port

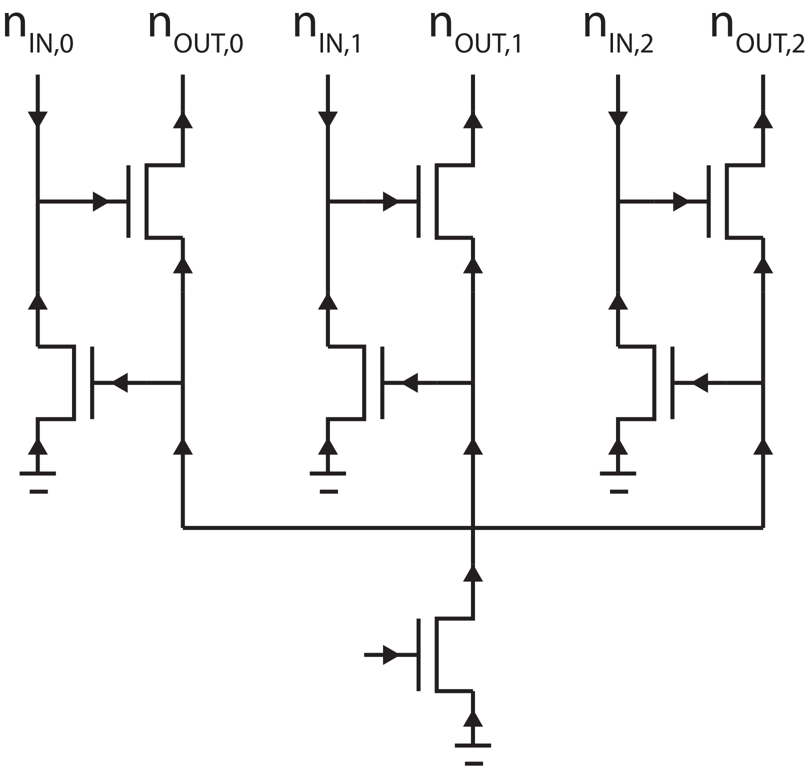


Figure 84. The WTA circuit with all ports labeled as inputs or outputs. In this partitioning, the input nets have multiple drivers on them, and circuit would be rejected by VPR.

that was attempting to drive that net now going to a unique input to a fan-in element, and the output of that fan-in element going to the single device that was using that net as an input.

In each case, a single two-input fan-in element was necessary to fix each net. Since these fan-in elements are implemented as rows in the local interconnect, and because in this case we simply want the fan-in elements to short all of these nets together, each of these fan-in elements will be implemented as two fully programmed on floating-gate switches.

This circuit is now a completely valid candidate for placement and routing on the RASP architectures using the VTR / RASP toolchain. However, since this example only shows the fan-in elements being used as band-aids, we will go ahead and add a VMM to this circuit, where the fan-in elements will be used to implement most of the

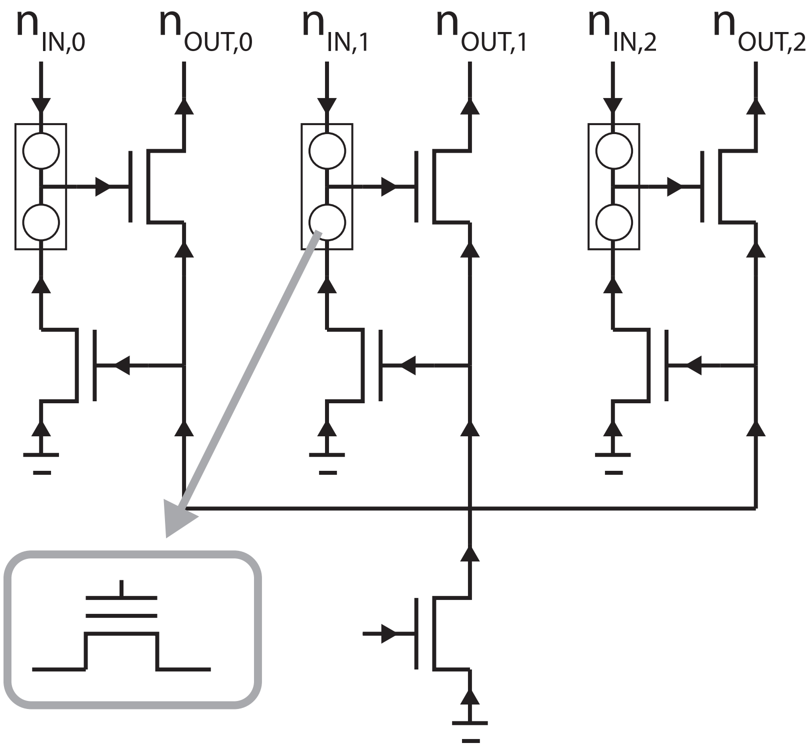


Figure 85. The nets with multiple drivers are split and combined using fan-in elements. This circuit is now a valid input to VPR.

VMM.

Figure 86 shows the circuit we will be mapping. This circuit uses a switch matrix of floating-gates to implement the VMM, with transimpedence amplifiers (with floating-gate feedback) as input stages. The goal will be to partition and label this circuit such that all of these floating-gates are implemented as fan-in elements.

The first step is to label all ports and identify the locations where fan-in elements have to be added. The second is to then absorb all floating-gates into these fan-in elements. A summary of these steps and the rules involved is as follows:

1) Produce a valid labeling of all pins on all elements in the circuit. An invalid labeling, for instance, might contain a device that had no input ports.

2) Split up all nets that have multiple drivers by inserting fan-in elements. The fan-in elements are N-input and one output devices, where in the RASP3.0 chip, N can go up

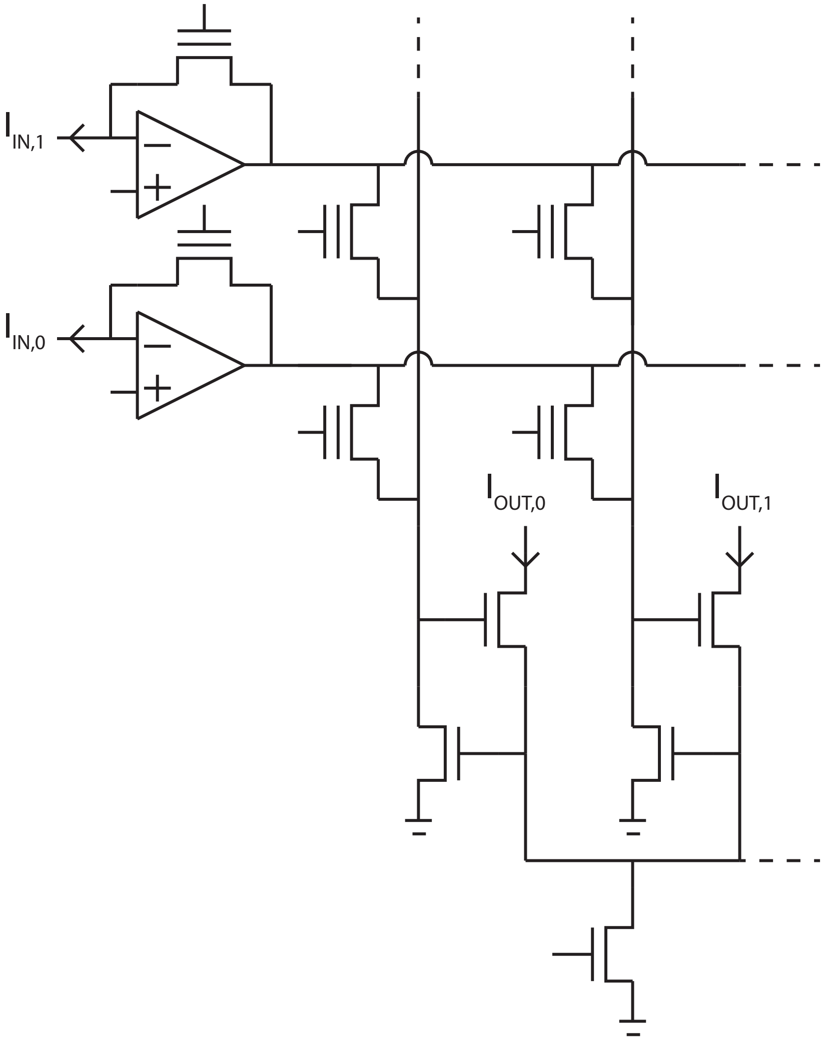


Figure 86. A VMM to WTA circuit using floating-gates for weight elements in the VMM and feedback elements for the OTA circuits.

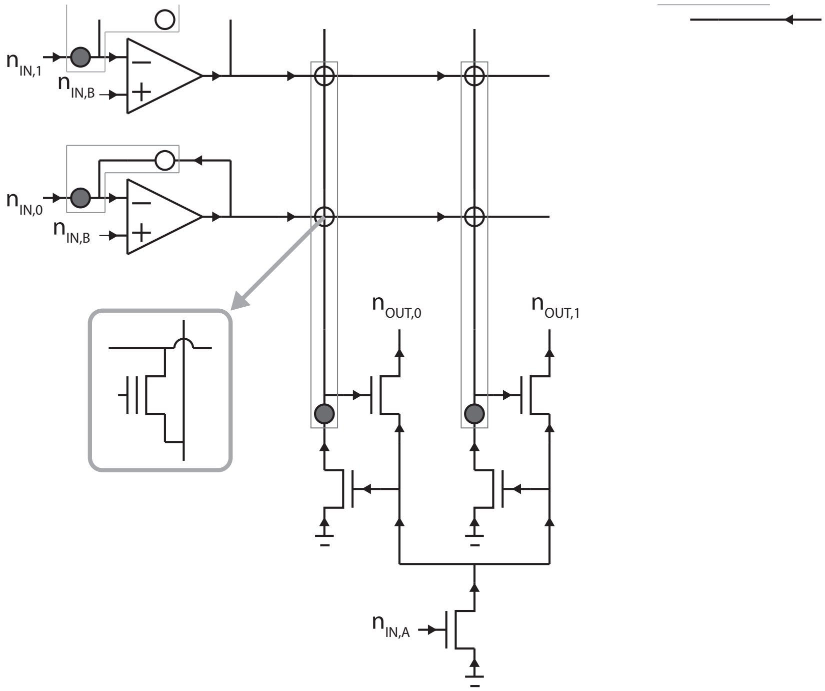


Figure 87. All device floating-gates have been lumped into fan-in elements. The fan-in elements fixing all nets with multiple drivers and implementing the explicit and precisely programmed floating-gate elements. In this figure precisely programmed floating-gate are shown as open circles, and floating-gates acting as on switches as solid circles.

to 13, and the output of each element has to be connected to one and only one input pin of a valid CAB device.

3) Absord floating-gates into these fan-in elements.

4) If the circuit is valid, stop. If not, then go back to step 1 and try another valid labeling of ports.

Figure 87 shows a completely valid paritioning. I have drawn the floating-gates in the fan-in elements that are being used to simply short nets as filled in black circles, and the floating-gates that are programmed to be computational elements as open circles. This circuit will be implemented with seven discrete CAB devices (5 nFETs and 2 OTAs), with the four fan-in elements implementing the floating-gates and being built out of rows of the local interconnect.

The following is a blif format netlist describing this circuit that is ready for placement and routing using the RASP toolchain. The extra numbers passed at the end of .subckt lines correspond to FG values for programming, with zero being as off as possible and one being as on as possible. The simple OTA devices in the CABs have FG based current sources, so they take a FG value for setting this bias current. The fan-in elements, called sm here in the .blif file, take a FG value for each input. Some of these inputs are used to simply short nets together and so recieve the value of one, while others are used to implement VMM weights, biases, resistors, etc. and therefore take values in the range of one to zero. Where these values will be mapped to drain current values, under some constant bias, of the injection transistor for that particular floating-gate address. Where a value of one will get mapped to the highest current producable by the floating-gate programmer, and a zero to the lowest, with the intermediate values corresponding to currents in-between. Where once properly characterize, specific drain current targets could be passed to the programmer instead.

```
.model vmma_wta
.inputs in0 in1 iba ibb gnd
.outputs out0 out1
#OTAs w/ FG feedback
.subckt sm in[0]=vi0 in[1]=in0 out=in0_ # 0.5 1
.subckt ota p=iba m=in0_ out=vi0 # 0.5
.subckt sm in[0]=vi1 in[1]=in1 out=in1_ # 0.5 1
.subckt ota p=iba m=in1_ out=vi1 # 0.5
#VMM from SM
.subckt sm in[0]=vi0 in[1]=vi1 inv[2]=vo0 out=vo0_ # 0.11 0.21 1.00
.subckt sm in[0]=vi0 in[1]=vi1 inv[2]=vo1 out=vo1_ # 0.12 0.22 1.00
#WTA legs
.subckt nfet s=gnd g=nx out=vo0
.subckt nfet s=nx g=vo0_ out=out0
.subckt nfet s=gnd g=nx out=vo1
.subckt nfet s=nx g=vo1_ out=out1
.subckt nfet s=gnd g=ibb out=nx
```

Figure 88 shows how this circuit would be mapped into a ficticious, RASP3.0-like CAB. In the drawing, the CAB is shown to contain exactly two OTAs, and 5 nFETs, whereas CABs in the RASP3.0 chip contain four OTAs and only two nFETs. So in the RASP3.0 this circuit would still route, but the placer portion of the toolflow would have to split the circuit up into at least three CABs to find enough discrete nFET devices to map the WTA portion.

7.12 VMM Synthesis and Macroblocks

One major new feature of the toolset is the ability to use preconfigured, black-boxed circuits. This enables the creation of highly optimized circuits that do not get touched by the synthesis and routing software, but placed anywhere in the array that they fit.

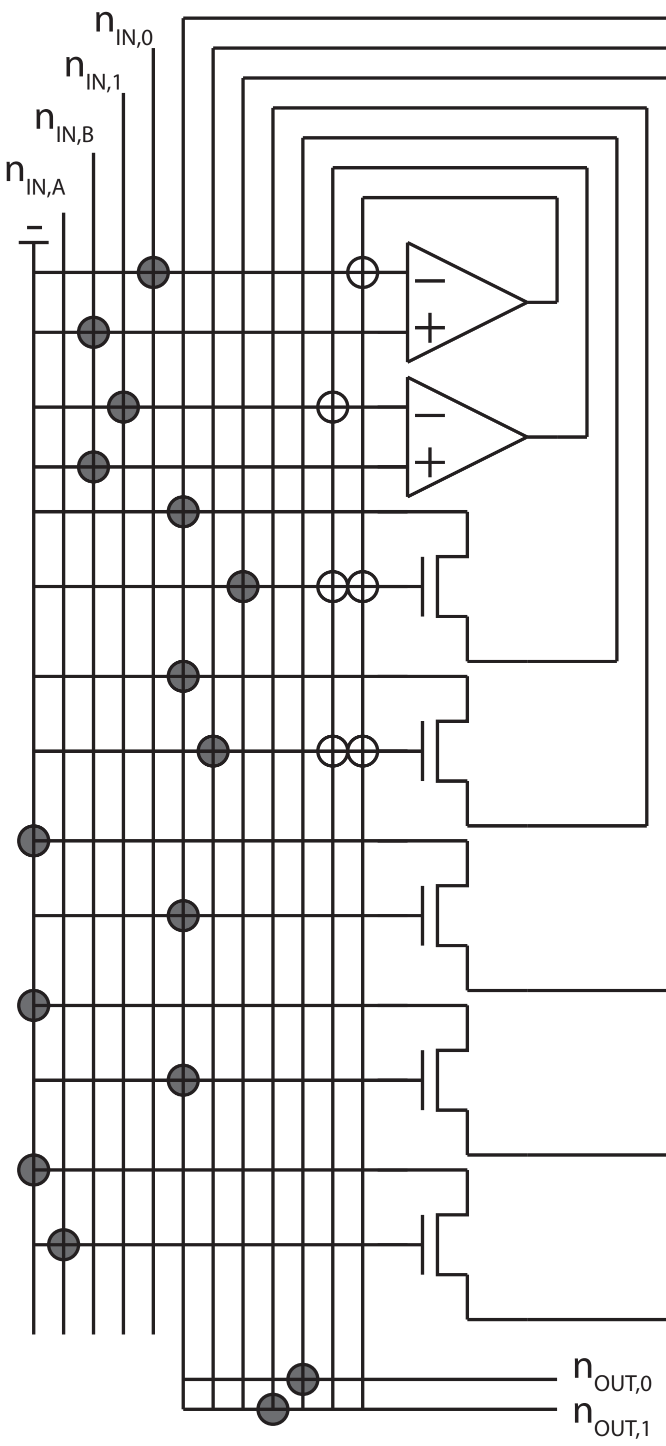


Figure 88. The VMM plus WTA circuit mapped to a ficticious CAB similar to RASP3.0 CAB. Solid circles are "ON" FGs being used simply to short nets, while the hollow circles are precisely programmed FGs implementing weights in a VMM or resistors. "OFF" FGs are not shown, but exist at all wire intersections in the local interconnect. The local interconnect rows with multiple circles are implementing the fan-in elements.

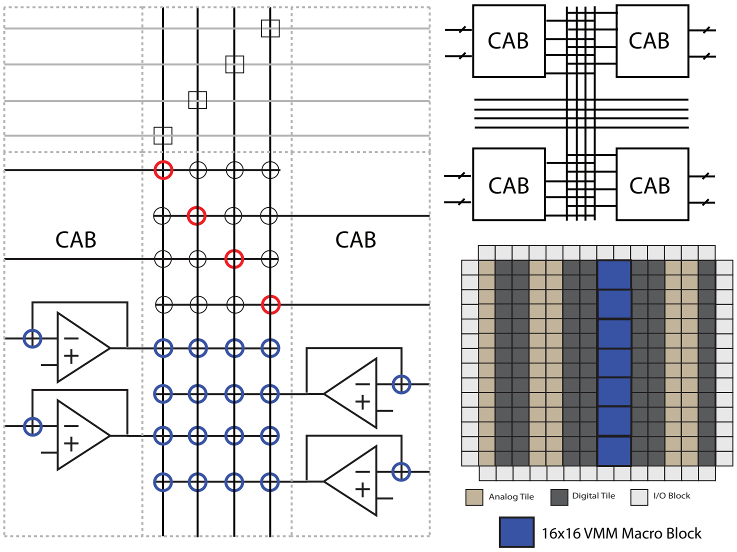


Figure 89. Preconfigured blocks implementing VMMs, lumped into larger blocks and presented as blackboxes to the routing software.

While this works particularly well for leveraging custom analog circuits that are very sensitive to the specific routing choices in their implementation, it also allows for the first time the creation of a way to synthesize VMMs out of the global interconnect of the chip.

Figure 89 shows a custom, preconfigured circuit implementing a VMM. This circuit requires the use of multiple CABs and their intermediate global interconnect. In this case, OTAs from the CAB devices and floating-gate switches from CBLOCKs are used to create the VMM.

Figure90 shows what the array looks like in VPR without this preconfigured circuit, and Figure 91 shows what the array looks like when a fraction of the CABs are lumped together and preconfigured as 16x16 VMMs.

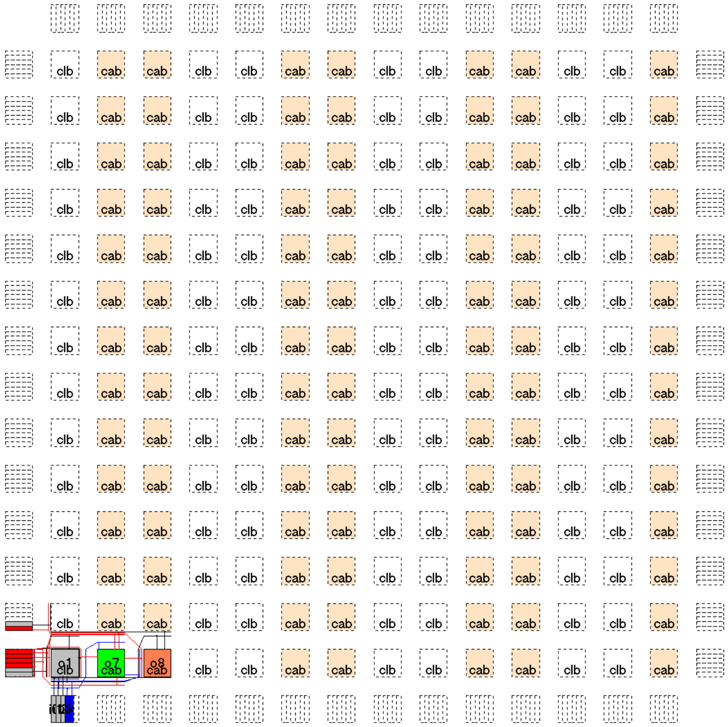

clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
clb cab cab clb clb cab cab clb clb cab cab clb clb cab
o1 clb o7 cab o8 cab clb clb cab cab clb clb cab cab clb clb cab

Figure 90. A block diagram of the RASP3.0 array as viewed in VPR's graphical tool. Heterogeneous columns of CLBs and CABs can be seen.

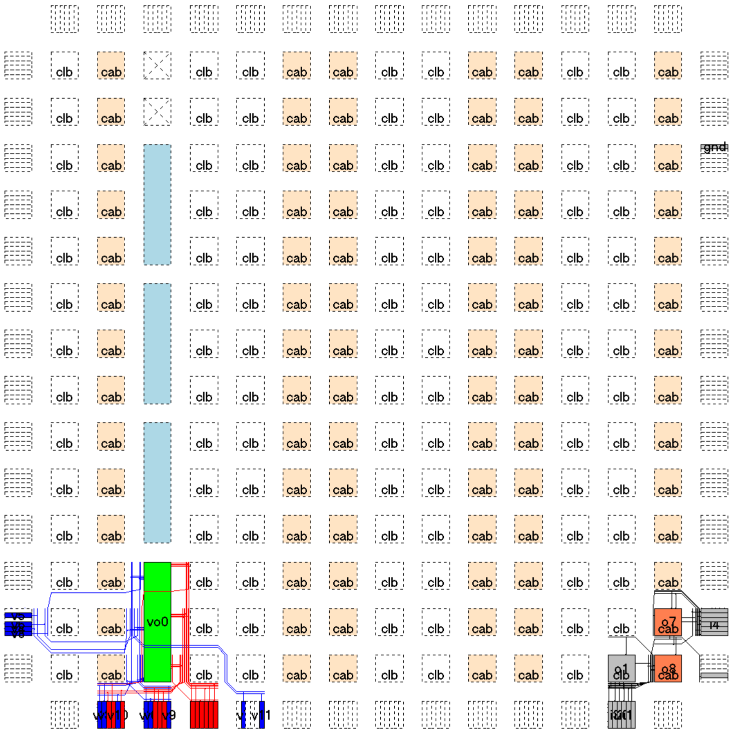


Figure 91. This is the architectural view when lumping together sections of CABs and preconfiguring them to implement hardwired macroblocks. In this instance, strings of three CABs on one column are preconfigured to implemented 16-input, 16-output VMMs. The routing tool is then free to place these VMMs in any of these prepacked locations.

CHAPTER 8
CONCLUSION

Floating-gates were a core technology used throughout this work. Their ability to operate like a transistor where the threshold voltage can be precisely modified at run time made them an attractive circuit solution for a host of problems. They were used to implement the interconnect in reconfigurable architectures, and they were integrated into many analog, digital, and neuromorphic circuits to modify their behavior. In analog circuits they allowed for tunability of the functions being implemented, or to mitigate the effects of process mismatch. When incorporated into digital circuits they allowed the delay and power consumption of gates to be modified.

An application of floating-gate ideas led to the creation of the CCFG CMOS logic family, which was shown to offer trade-offs similar to that of dynamic voltage scaling (DVS) and multiple threshold technologies (MVT), but without a lof of the limitations of either technology. CCFG CMOS were simulated to show the ability to go up to 15% faster at the same energy per cycle as standard logic, or on non-critical paths, they could be slowed down and were shown, in some cases, to reduce power as much as 90% over standard gates operating with positive slack. The analysis of the complicated effects that using floating-gate transistors have on the various components of power consumption in this logic family, led to the discovery of a better and more accurate way to measure short-circuit power in simulation. This line of research then led to the creation of a Logical Effort compatible power analysis technique.

The FPAADD was created to supplement the reconfigurable analog processing capabilities of previous RASP chips by adding reconfigurable digital circuitry to the arrays. This allowed portions of problems to be partioned into pieces solved in either analog, digital, or both. The architecture was designed to be much more general purpose than existing architectures mixing FPGA and FPAA arrays. The goal of the FPAADD was to be able to target applications that were largely solved in analog, but needed small amounts of digital circuitry for things like data flow control. The FPAADD was fabricated in a 0.35um process and tested. Data converters were shown to be synthesizable from the reconfigurable fabric in a range of topologies and parameters: A sigma delta converter and a VCO-based DAC were tested. The VPR simulation suite, for FPGA architectural trade-off analysis, was modified to target and perform placement and routing of circuits on the FPAADD.

Partly in order to target applications where more of the problem was solved in the digital domain than could be solved on the FPAADD, the RASP3.0 architecture and line of chips were created that added a processor, SRAM, and hard-coded data converters and peripherals to an FPAADD array. Several different flavors of chips have been built: the RASP3.0 is the most general chip, the RASP3.0a is a smaller verion for educational purposes, the RASP3.0n supplements the CABs and CLBs with blocks containing neurons, and the RASP3.0rf was dessigned for high speed RF applications and built in a 40nm process. These new SoCs are designed to be larger, more self-contained, and flexible in solution implementation than previous architectures. Hard-coded data converters exist as peripherals to the processor, between the array and the processor, and between data representations. These can serve as instrumentation- a way to take measurements and apply signals to the circuits being implemented, or be used to put the processor in the computational path itself. The place and route tool uses a newer version of VPR, and has been modified to support the targetting of floating-gate transistors from the interconnect not only as switches, but as computational elements. For the first time, efficient automated routing of circuits containing VMMs built from interconnect switch matricies was possible. The RASP3.0 chip was fabricated in a 0.35um process. The system is currently being tested: the processor, peripherals, and floating-gate programming have all been shown to be functional. And much of the software infrastructure has been written and debuged.

These chips will serve as powerful platforms for exploring and implementing non-traditional solutions to problems; solutions able to leverage analog, digital, and neuromorphic computational techniques. I look forward to seeing what my colleagues are able to do with them in the years to come.

REFERENCES

[1] P. Hasler, A. Basu, and S. Kozil, "Above threshold pfet injectionmodeling intended for programmingfloating-gate systems," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*. IEEE, pp. 1557–1560.

[2] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 868–873.

[3] J. T. Kao, M. Miyazaki, and A. Chandrakasan, "A 175-mv multiply-accumulate unit using an adaptive supply voltage and body bias architecture," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 11, pp. 1545–1554, 2002.

[4] B. P. Degnan, R. B. Wunderlich, and P. Halser, "Programmable floating-gate techniques for cmos inverters," in *IEEE International Symposium on Circuits and Systems*, 2005, pp. 2441–2444.

[5] Y. Berg, D. Wisland, and T. Lande, "Ultra low-voltage/low-power digital floating-gate circuits," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 7, July 1999.

[6] D. H. Ivan Sutherland, Bob Sproull, *Logical Effort: designing fast CMOS circuits*. Morgan Kaufmann, 1999.

[7] D. Harris, "Logical effort of higher valency adders," *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, pp. 1358–1362, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1399375

[8] D. H. Ivan Sutherland, Bob Sproull, *Logical Effort: designing fast CMOS circuits*. Morgan Kaufmann, 1999.

[9] J. Ebergen, J. Gainsley, and P. Cunningham, “Transistor sizing: how to control the speed and energy consumption of a circuit,” *10th International Symposium on Asynchronous Circuits and Systems, 2004. Proceedings.*, pp. 51–61, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1299287

[10] J. Fishburn and S. Taneja, “Transistor sizing for high performance and low power,” *Proceedings of CICC 97 - Custom Integrated Circuits Conference*, pp. 591–594, 1997. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=606695

[11] H. J. M. Veendrick, “Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits,” vol. 19, no. 4, pp. 468–473, 1984.

[12] S. M. Kang, “Accurate simulation of power dissipation in vlsi circuits,” *IEEE JOURNAL OF SOLID-STATE CIRCUITS*.

[13] L. Bisdounis and O. Koufopavlou, “Short-circuit energy dissipation modeling for submicrometer cmos gates,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: FUNDAMENTAL THEORY AND APPLICATIONS*, vol. 47, pp. 1350–1361, 2000.

[14] N. S. Srinivasa R. Vemuru, “Short-circuit power dissipation estimation of cmos logic gates,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I: FUNDAMENTAL THEORY AND APPLICATIONS*, vol. 41, pp. 762–765, 1994.

[15] W. Grabinski, “Ekv model v2.6 and extraction methodologies,” 2001.

[16] T. S. Koichi Nose, “Analysis and future trend of short-circuit power,” *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 19, pp. 1023–1030, 2000.

[17] K. F. E. Lee and P. G. Gulak, “A transconductor-based field-programmable analog array,” in *ISSCC Digest of Technical Papers*, Feb. 1995, pp. 198–199.

[18] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Y. Sun, “A field programmable analog array for cmos continuous-time ota-c filter applications,” *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 2, pp. 125–136, 2002.

[19] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli, “A field-programmable analog array of 55 digitally tunable otas in a hexagonal lattice,” *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 12, pp. 2759–2768, 2008.

[20] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. M. Twigg, and P. Hasler, “A floating-gate-based field-programmable analog array,” *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 9, pp. 1781–1794, 2010.

[21] *AN13x series AN23x Series AnadigmApex dpASP Family User Manual*.

[22] D. Fernández, L. Martínez-Alvarado, and J. Madrenas, “A translinear, log-domain fpaa on standard cmos technology,” *Solid-State Circuits, IEEE Journal of*, no. 99, pp. 1–1, 2012.

[23] S. Ganesan, “Synthesis of mixed-signal systems based on rapid prototyping,” 2001.

[24] P. Chow and P. G. Gulak, “A field-programmable mixed-analog-digital array,” in *Proc. Third Int. ACM Symp. Field-Programmable Gate Arrays FPGA ’95*, 1995, pp. 104–109.

[25] J. Madrenas, J. M. Moreno, E. Canto, J. Cabestany, J. Faura, I. Lacadena, and J. M. Insenser, “Rapid prototyping of electronic systems using fipsoc,” in *Proc. 7th IEEE Int. Conf. Emerging Technologies and Factory Automation ETFA ’99*, vol. 1, 1999, pp. 287–296.

[26] M. Mar, B. Sullam, and E. Blom, “An architecture for a configurable mixed-signal device,” *Solid-State Circuits, IEEE Journal of*, vol. 38, no. 3, pp. 565–568, 2003.

[27] W. Fu, J. Jiang, X. Qin, T. Yi, and Z. Hong, “A reconfigurable analog processor using coarse-grained, heterogeneous configurable analog blocks for field programmable mixed-signal processing,” *Analog Integrated Circuits and Signal Processing*, vol. 68, no. 1, pp. 93–100, 2011.

[28] J. Gray, C. Twigg, D. Abramson, and P. Hasler, “Characteristics and programming of floating-gate pfet switches in an fpaa crossbar network,” in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*. IEEE, 2005, pp. 468–471.

[29] C. Schlottmann, C. Petre, and P. Hasler, “Vector matrix multiplier on field programmable analog array,” in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1522–1525.

[30] B. A. Minch, C. Diorio, P. Hasler, and C. A. Mead, “Translinear circuits using subthreshold floating-gate MOS transistors,” *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 2, pp. 167–179, 1996.

[31] C. M. Twigg and P. Hasler, “A large-scale reconfigurable analog signal processor (rasp) ic,” in *Proc. IEEE Custom Integrated Circuits Conf. CICC ’06*, 2006, pp. 5–8.

[32] T. S. Hall, C. M. Twigg, J. D. Gray, P. Hasler, and D. V. Anderson, “Large-scale field-programmable analog arrays for analog signal processing,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 11, pp. 2298–2307, 2005.

[33] A. M. Vaughn Betz, Jonathan Rose, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[34] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," vol. 81, no. 7, pp. 1013–1029, 1993.

[35] P. Leventis, B. Vest, M. Hutton, and D. Lewis, "Max ii: A low-cost, high-performance lut-based cpld," in *Proc. Custom Integrated Circuits Conf the IEEE 2004*, 2004, pp. 443–446.

[36] C. Hu, "Interconnect devices for field programmable gate array," in *Proc. Int. Electron Devices Meeting Technical Digest*, 1992, pp. 591–594.

[37] F. Baskaya, S. Reddy, S. K. Lim, and D. V. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 8, pp. 906–910, 2006.

[38] S. Ganesan and R. Vemuri, "Analog-digital partitioning for field–programmable mixed signal systems," in *2001 Conference on Advanced Research in VLSI*, E. Brunvand and C. Myers, Eds. IEEE Computer Society, March 2001, pp. 172–185.

[39] ——, "Technology mapping and retargeting for field–programmable analog arrays," in *DATE 2000 Proceedings: Design, Automation and Test in Europe Conference 2000*, Mar. 2000.

[40] ——, "Behavioral partitioning in the synthesis of mixed analog-digital systems," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, pp. 133–138.

[41] P. A. Jamieson and K. B. Kent, "Odin ii: an open-source verilog hdl synthesis tool for fpga cad flows (abstract only)," in *Proceedings of the 18th annual ACM/SIGDA*

international symposium on Field programmable gate arrays, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 288–288.

[42] "Berkeley logic synthesis and verification group, abc: A system for sequential synthesis and verification, release 70731. http://www.eecs.berkeley.edu/ alanmi/abc/," 2007.

[43] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 133–142.

[44] J. Noullet and A. Ferreira-Noullet, "Do we need so many cells for digital asic synthesis?" *ELECTRON TECHNOLOGY-WARSAW-*, vol. 32, pp. 272–276, 1999.

[45] M. Qazi, K. Stawiasz, L. Chang, and A. P. Chandrakasan, "A 512kb 8t sram macro operating down to 0.57 v with an ac-coupled sense amplifier and embedded data-retention-voltage sensor in 45 nm soi cmos," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 85–96, 2011.

APPENDIX A

LOGICAL POWER EXTRACTION SOFTWARE

The software is readily available through a Subversion repository .

We want to characterize a logic gate. Take for example, this two-input NAND gate.

A.1 Characterization Circuit

In order to extract the parameters of Logical Effort and Logical Power the gate to be characterized will be placed in a characterization circuit for simulation. The circuit is composed of many paths constructed out of the gate. Each path has five stages, and the electrical effort of each stage is constant within a path, with each path having a different electrical effort for each stage.

Shown is the first stage of the first two paths. In the first path $h = 1$, in the second $h = 2$. This structure is particularly convenient in that we do not actually need to know the absolute values of the output and load capacitances to get h as we've set it up to be the ratio of multiples of the same capacitance.

Measurements will actually be performed on the third stage of each path, with the first two stages shaping the input pulse into something reasonable. Gates designed to be loads, themselves are loaded, otherwise their outputs would swing unrealistically fast skewing their own input capacitance. The measurements will be input and output voltages, gate input, output, vdd, gnd, bulk, and well currents as a function of time.

A.2 Installation and Setup

From a command prompt:

```
svn checkout svn+ssh://<you>@ecelinsrv2.ece.gatech.edu/tools/cadsp/svn/mad/logical_power/
    logical_power

cd matlab
```

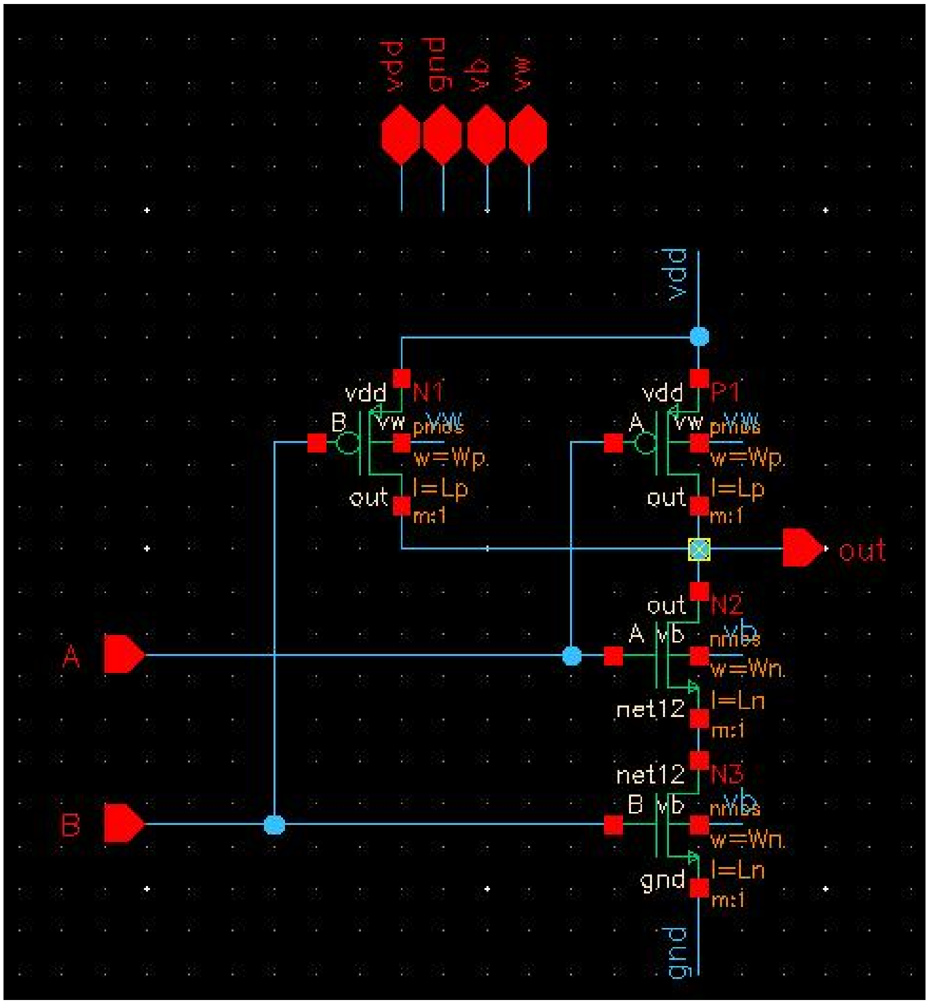


Figure 92. 2-input NAND gate

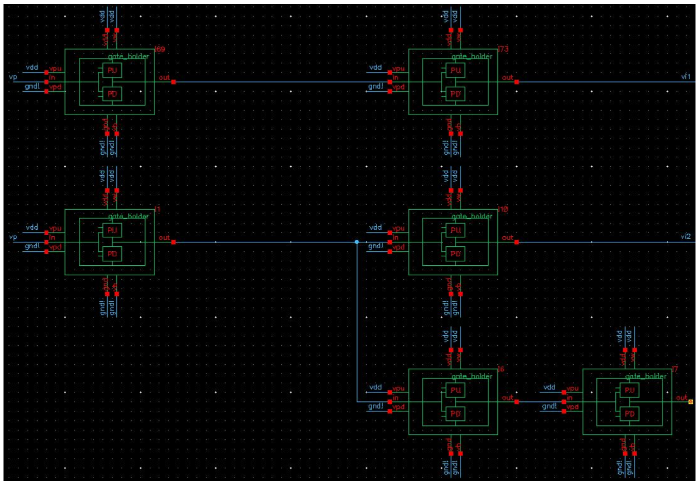


Figure 93. gate char stage one

```
matlab
```

At this point, the Cadence side of the software suite should be working as well. You can try:

```
cd cadence

source cshrc.ncsu61.ece.lin

virtuoso
```

A.3 Simple Example

Start Matlab and then try:

```
>> inv = gateChar(’ibm_130n.scs’);
```

This sets up a simulation in a 130n technology defined by the file ibm_130n.scs. A lot of defaults have been set at this point to quickly get to simulating. Check around in the inv.params.cir, inv.params.dim, inv.params.tim, inv.params.ana structures. For instance:

```
>> inv.params.cir

ans =

tech_file: ’"ptm_130n.scs"’

gate_to_char: ’gate_inv’

circuit_file: ’gateChar_6h.scs’

gates_file: ’"gates.scs"’

raw_out_file: ’temp.out’

simulator: ’spectre’

VDD: 1.2000
```

```
>> inv.params.dim.W

ans =

n: 1.8000e-07

p: 5.4000e-07
```

The default gate to simulate is an inverter, labeled gate_inv, where the gate_ terminology means we've wrapped the gate into a universal gate holder that can be plug and placed with any other gate into the characterization circuitry.

The default voltages and dimensions are set based on the technology file used. From here we can immediately simulate:

```
>> inv.runSim();
```

When this completes we can quickly generate some plots by:

```
>> inv.plotD

>> inv.plotI

>> inv.plotQ
```

Or we can access the data structures in the class gateChar directly to generate less obvious plots. For more information on the data structures, see logical_power/matlab/@gateChar/gateCh and take a good look at the comments in the properties section especially regarding the properties rawData, extData, and anaData.

A.4 Modifying Default Parameters

There are a pile of default parameters set by the gateChar constructor.

Anywhere in any of the circuit files (gates.scs, tech_file.scs, gate_char.scs) variables can be made and set by matlab. The syntax for the variable placeholder in the circuit as well as the structure definition for setting the variable in matlab is:

in gates.scs

```
...

x0 1 0 res <resx>
```

in matlab

```
>> x = gateChar('tech_lib.scs');

>> x.params.some_category.res.x = val;

>> x.runSim();
```

Where some_category is any name you want it to be, its simply for organizational purposes. dim holds transistor dimensions, cir holds gate type, vdd, etc, tim holds timing information and so on. Evertying to the right of some_category gets turned into the variable name to search for. In this case, res.x gets turned into resx. Again this is just for convenience and the simulator supports any number of nestings in the structure.

When runSim is called it builds a new circuit file with the variables specified. The resulting circuit file will have <resx> replaced with val when simulated.

A.5 Sweeps

The sweeps functionality allows one to sweep parameters and characterize tweaked versions of a base gate. The syntax is as follows:

```
x = gateChar('tech.scs');

x.params = some_struct;

x.sweeps(1).some_category.vector_11;

x.sweeps(1).some_category.vector_12;

...

x.sweeps(2).some_category.vector_1n;

x.sweeps(2).some_category.vector_21;
```

```
x.sweeps(2).some_category.vector_22;
...
x.sweeps(2).some_category.vector_2m;
...
x.sweeps(N).some_category.vector_N1;
x.sweeps(N).some_category.vector_N2;
...
x.sweeps(N).some_category.vector_Nz;
x.runSim();
```

Where some_struct contains whatever initial parameters you want to set. And the vectors are values to be swept. All vectors within sweeps(1) will be swept together and versus all vectors in sweeps(2) versus all vectors in sweeps(3) versus ... all vectors in sweeps(N).

The software exploits the parallelization of the parametric simulation on multi-core systems. Each sweep point is a new SPICE simulation independent of the previous, all SPICE simulations are setup in advance and then dispatched to as many cores as the system has access to (the current version of MATLAB 2009a supports a maximum of eight cores).

A.6 2D Sweep Example: r, VDD

Lets try the following code:

```
>> inv2 = gateChar('ptm_130n.scs'); %default is inverter
>> inv2.sweeps(1).dim.W.p = [1:0.5:2]*inv2.params.dim.W.n;
>> inv2.sweeps(2).cir.VDD = [0.8, 1.2];
>> inv2.runSim();
```

From there we've told the suite that we want to characterize this gate for an array of different parameters. The gate will be characterized for the following 2d space, various r = Wp / Wn ratios and various VDD values. The results can then be accessed and viewed in various ways.

```
>> inv2.plotD({1,1}) %plots delay for r = 1.0 vdd = 0.8
>> inv2.plotD({2,2}) %plots delay for r = 1.5 vdd = 1.2
```

In general, a netlist of gates for possible characterization is defined. These gates can have many variables associated with them, for instance, widths and lengths of transistors, operating voltages, etc. In fact, in the gate characterization netlist, the gate to be characterized is itself a variable as well as the technology file defining the transistor models.

In Matlab you set up a simulation by defining what all of these variables are, Matlab then parses the netlist replacing the variables with actual values and calls Spectre to simulate the netlist, Matlab then parses the output and analyzes the data produced, extracting all of the relevant gate characterization parameters.

Matlab will also set up and simulate an arbitrary dimension of sweeps, 3d, 4d, ... Nd there's no limit.

A.7 1D Sweep Example: r

However, lets explore 1d with one more example. Sweeps make it easy to do things like find the r that minimizes gavg for a variety of gates, etc. For instance:

```
>> r = [1:0.1:3];
>> inv3 = gateChar('ibm_130n.scs');
>> inv3.sweeps(1).dim.W.p = r*inv3.params.dim.W.n;
>> inv3.runSim(); >> figure, plot(r, inv3.extData(:,1:6), '-');
```

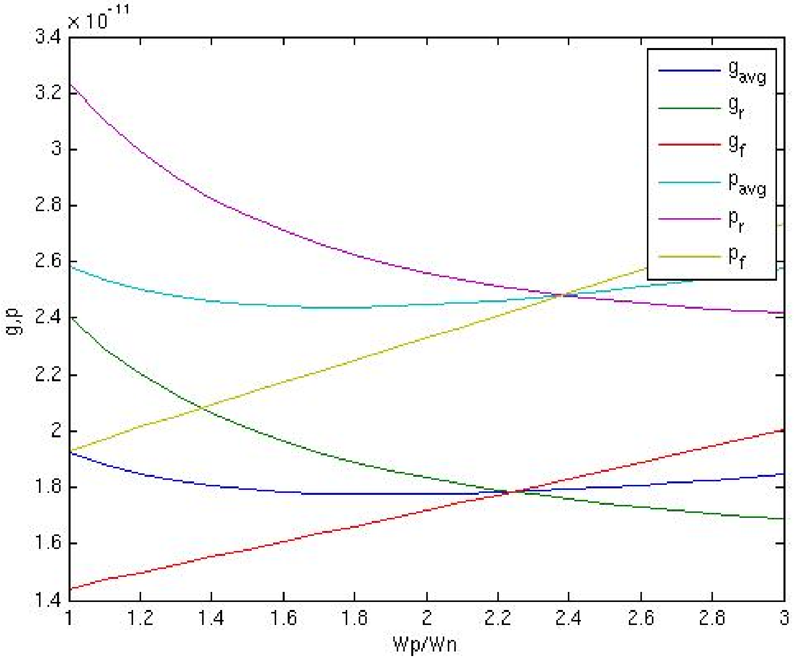


Figure 94. ***Logical effort*** **and** ***parasitic delay*** **for an Inverter versus** $r = W_p/W_n$

```
>> xlabel('Wp/Wn'); >> ylabel('g,p');

>> legend('g_{avg}', 'g_r', 'g_f', 'p_{avg}', 'p_r', 'p_f');
```

And here's the resulting plot to the right. Note that this is a good example of why equal rise and fall times are unobtainable for arbitrary load sizes, as when $g_r = g_f$ $p_r \neq p_f$.

Netlists describing the gates can be generated by hand, or graphically / schemactically through Cadence which will be the topic of the next section.

A.8 Generating Netlists

Ok, so you've got some gates in mind that you want to characterize, and you don't want to write netlists by hand. Here's what we do. We're going to generate a gates netlist file like the one located in

```
logical_power/matlab/circuits/gates.scs
```

So open it up, check it out, and keep it in mind as we go through the following. Move to the cadence directory, then:

```
source cshrc.ncsu61.ece.lin virtuoso &
```

Open up the library manager (tools -> library manager) and go to the logical_power library. We'll be concerned with the following cells:

- GATES
- GATE_CHAR
- gate_nand2_a
- nand2

Open up GATES, this is the schematic that will hold all of the gates that we hope to characterize. Notice that they all have the same ports. Each gate at this view represents a particular input of a particular gate to be characterized. For instance, there are two for the nand2 gate, one for input a and one for input b. If you'll notice, all gates have the same number of ports so that they can be exchanged without headache into the gate characterization circuit. Now descend all the way down to the nand2 schematic by way of the gate_nand2_a schematic.

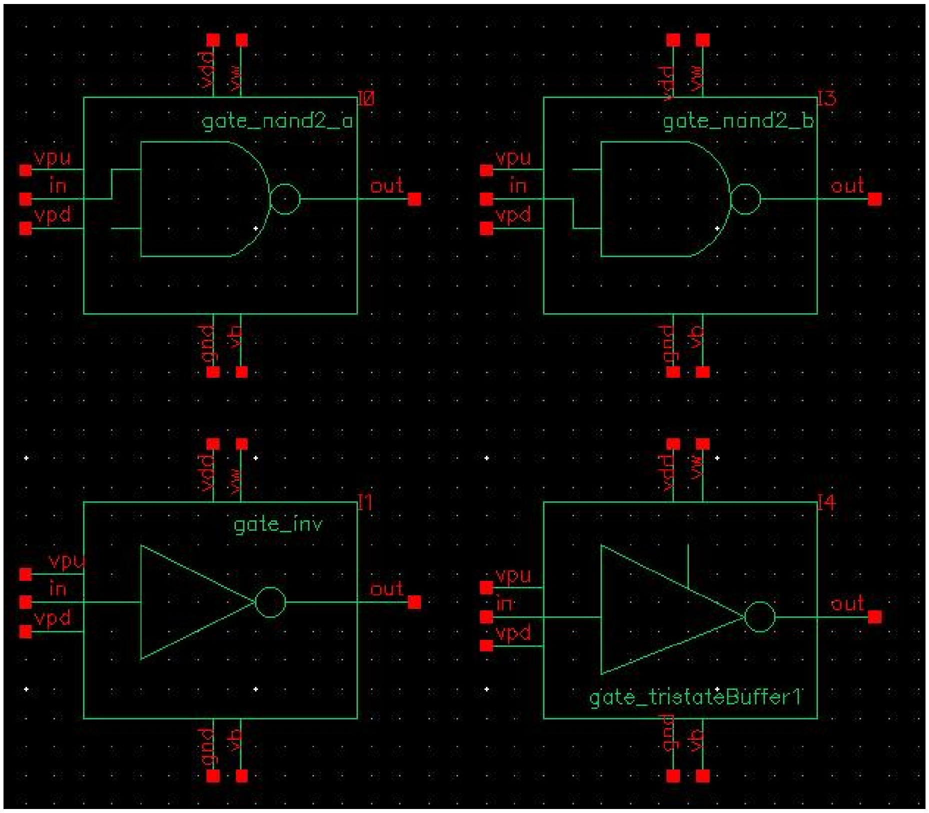
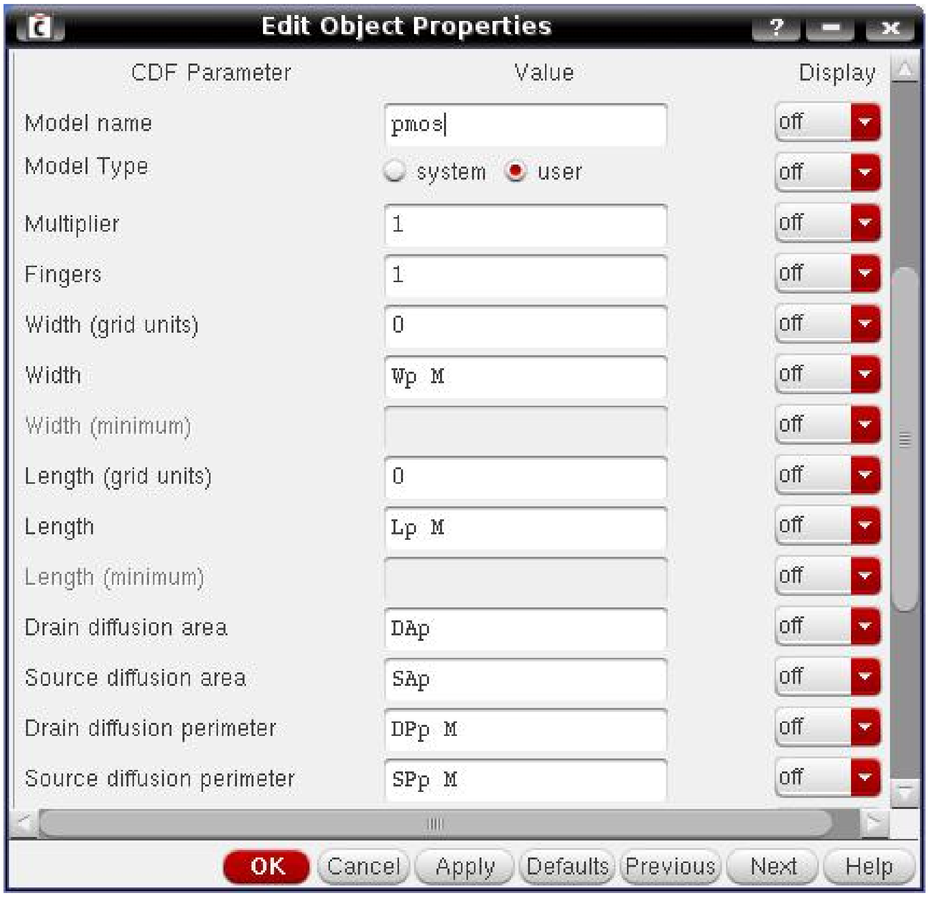


Figure 95. Gates to be characterized



CDF Parameter	Value	Display
Model name	pmos	off
Model Type	○ system ● user	off
Multiplier	1	off
Fingers	1	off
Width (grid units)	0	off
Width	Wp M	off
Width (minimum)		off
Length (grid units)	0	off
Length	Lp M	off
Length (minimum)		off
Drain diffusion area	DAp	off
Source diffusion area	SAp	off
Drain diffusion perimeter	DPp M	off
Source diffusion perimeter	SPp M	off

Figure 96. PFET properties

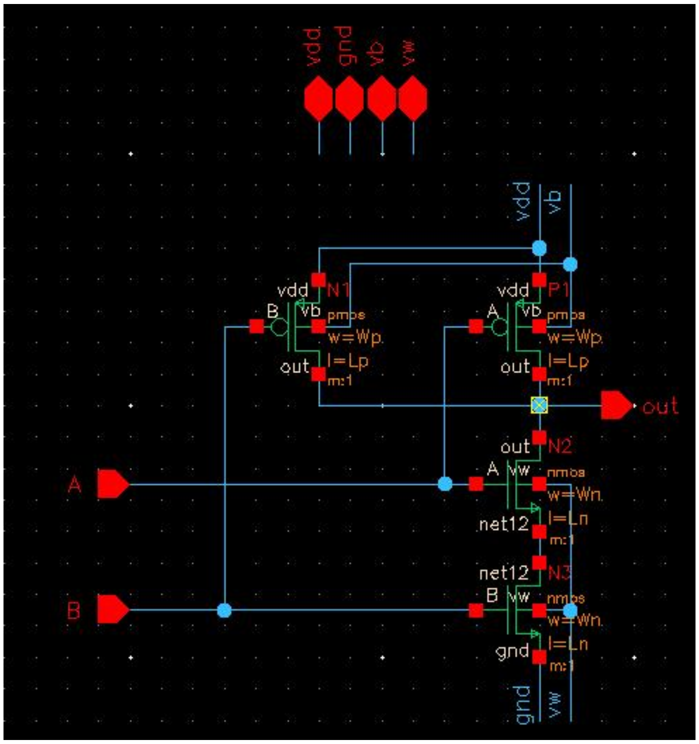


Figure 97. 2-input NAND gate

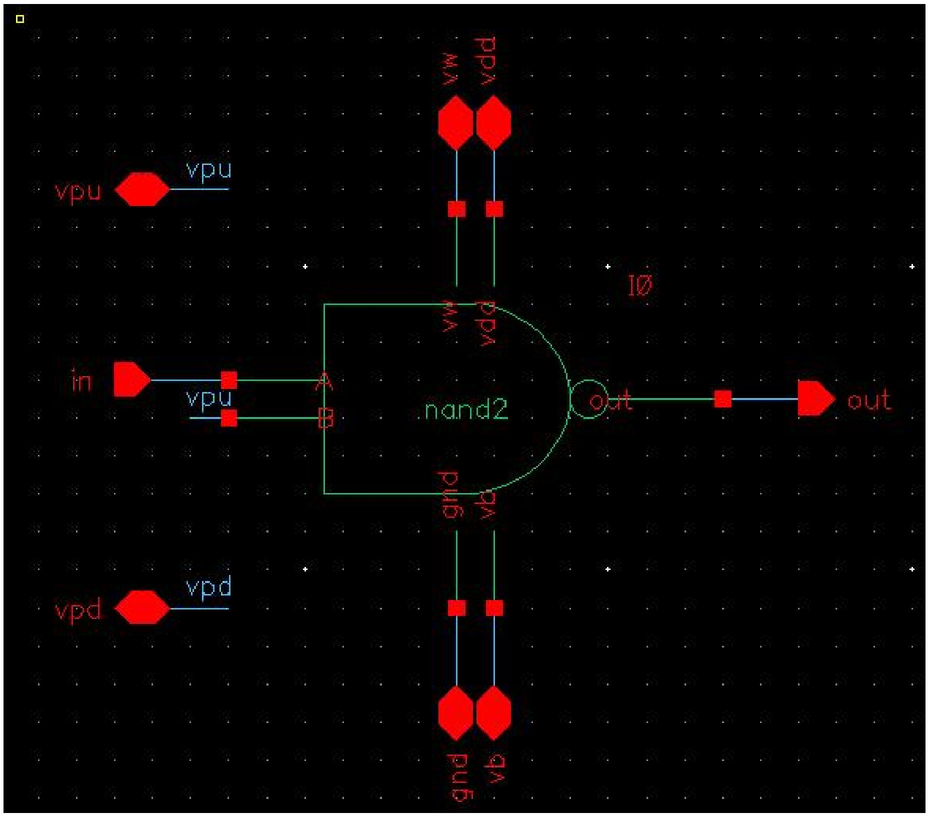


Figure 98. NAND2 input-a

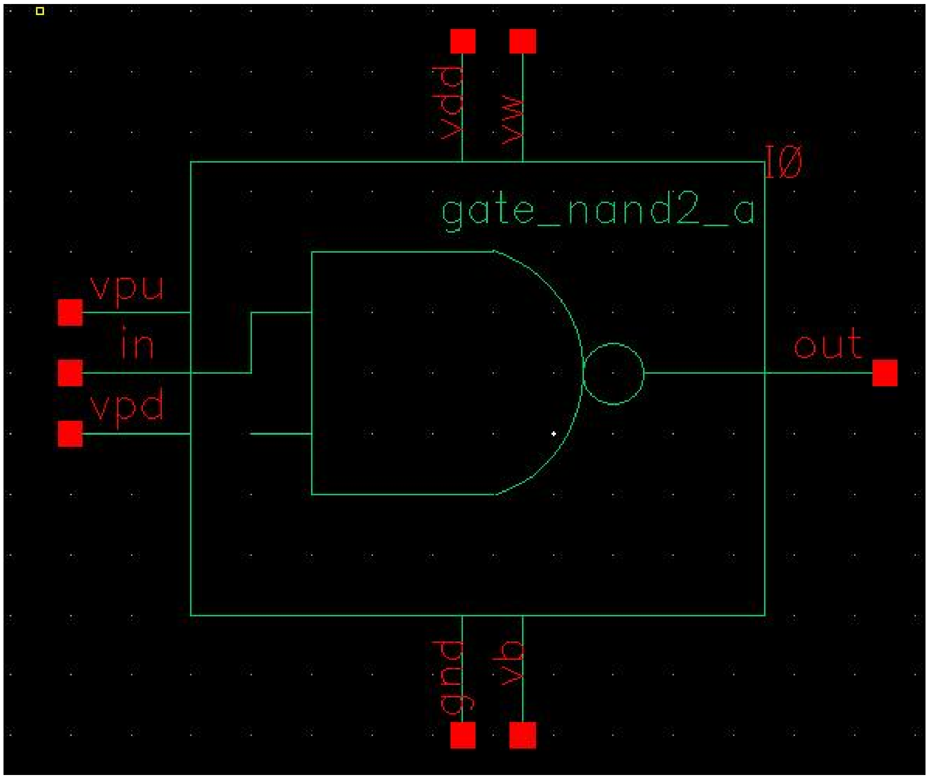


Figure 99. input-a of a NAND2 gate