



PowerAnalyzer for Pocket Computers

Dr. Robert Graybill's PAC/C Program

Fifth Review: June 23, 2003

Todd Austin and Trevor Mudge, U. Michigan

Dirk Grunwald, U. Colorado

<http://www.eecs.umich.edu/~jringenb/power/>



Status: PowerAnalyzer + Related Projects



- Overall status and remaining schedule
- Status of PowerAnalyzer and future work
- Razor



Remaining Schedule



- No-cost extension through September 2003
- PowerAnalyzer
 - Latest version (version 2) released January 2003
 - Static power dissipation support, memory models, interconnect models
 - Next release (version 3) planned for August 2003
 - More accurate structural modeling framework
 - Completed user manual August 2003
- SimpleScalar platform
 - Latest release (version 3) February 2003
 - I/O tracing support, test release of PDA platform emulation
 - Version 2 planned for August 2003
 - Stable release of PDA platform emulation
 - Additions to SimpleScalar manual September 2003



PowerAnalyzer



- High-level high-performance power estimates
 - Generates power/performance estimates early in design cycle
 - Based on SimpleScalar cycle accurate simulators, extended to ARM platform
- Improvement over previous high-level power simulators
 - Uses actual technology parameters – TSMC 0.18
 - Monitors hamming distances between consecutive inputs of real data
 - Interconnect length is input explicitly – requires early layout estimates
- Performance impact ~4x slowdown over baseline simulators



PowerAnalyzer Validation Effort



- Stochastic power modeling approach
 - Characterize (via SPICE) energy as a function of input hamming distance
 - Performed for ALU, cache, BTB, datapaths, etc...
 - Power function integrated into SimpleScalar performance model
- Accuracy: typically 20% of SPICE for random input vectors

- Sources of power estimation error

- Hamming distance often correlates poorly with energy consumed
 - Consider, for example these inputs to an adder, with hamming distance 1

Case #1: t=1 a: 0x00000000 b:0x00000000 cin: 0
t=2 a: 0x00000000 b:0x00000001 cin: 0

Case #2: t=1 a: 0xFFFFFFFF b:0x00000000 cin: 0
t=2 a: 0xFFFFFFFF b:0x00000001 cin: 0

But energy from SPICE for case 1 is 5.662E-15 J, case 2 is 2.372E-12 J

- Stochastic approach does not consider the underlying structure of unit
 - Consider, for example, a ripple carry adder vs. a Kogge-Stone adder



Structural Power/Delay Modeling



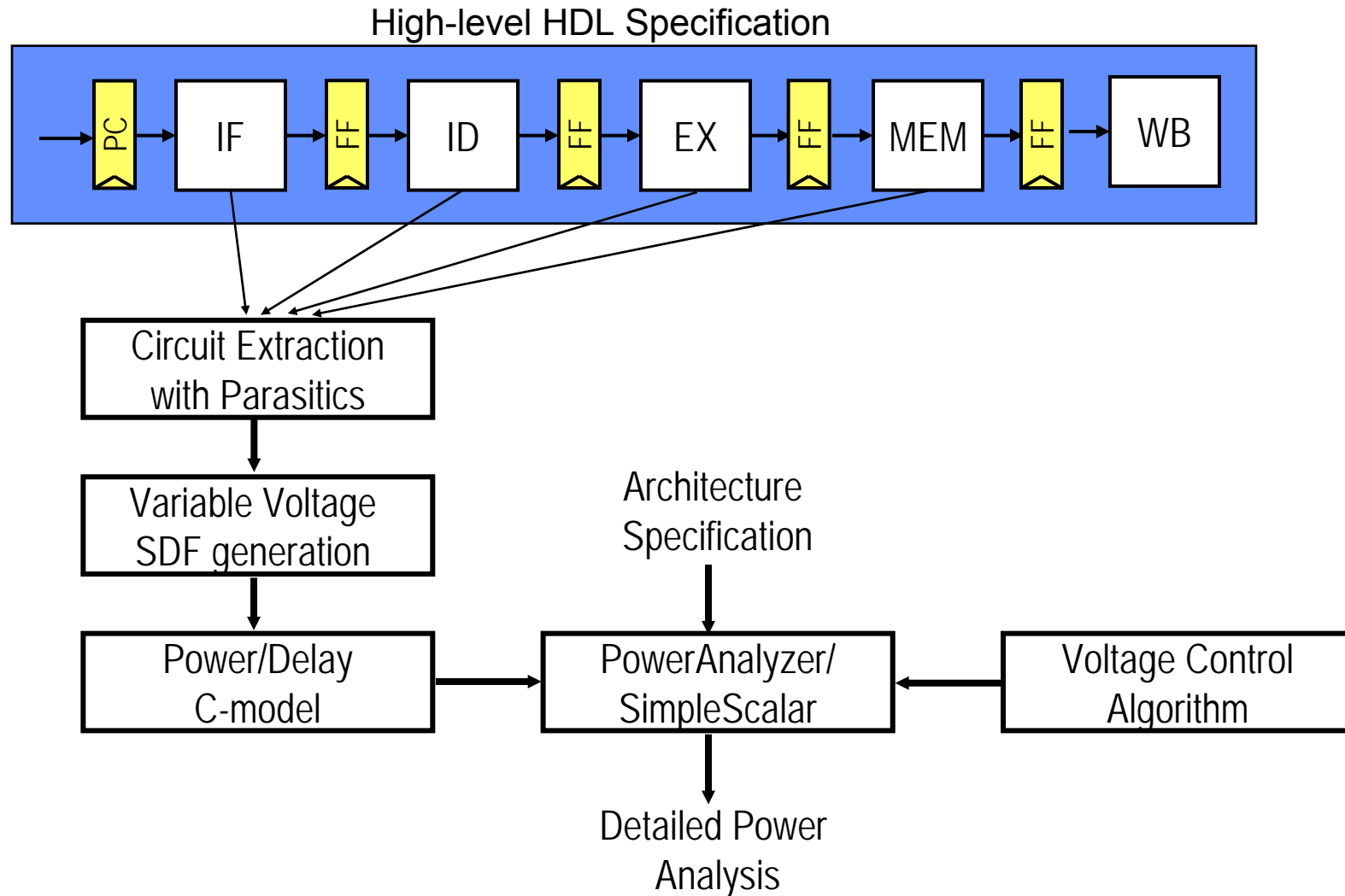
- Developed structural power-delay modeling technique
 - Circuit-level models of architectural units characterize power-delay
 - Gate-level cells characterized for range of operating voltages
 - Levelized simulation technique permits fast evaluation
 - Implemented using high-performance C models
- Accuracy: typically within 9% of SPICE for random vectors
 - Follows magnitude and trends much better than stochastic approach
- Benefits of structural modeling approach
 - Models consider internal and input/output node capacitances
 - Specified by user with circuit topology and transistor sizes
 - Models consider actual node transitions using embedded logic simulator
 - Models allow user to estimate parasitic capacitances
- Issues with structural modeling
 - Levelized simulation does not detect glitch power
 - Model construction is a labor intensive endeavor



Next Generation Power/Delay Analysis



- Automate creation of very accurate power/delay C-models

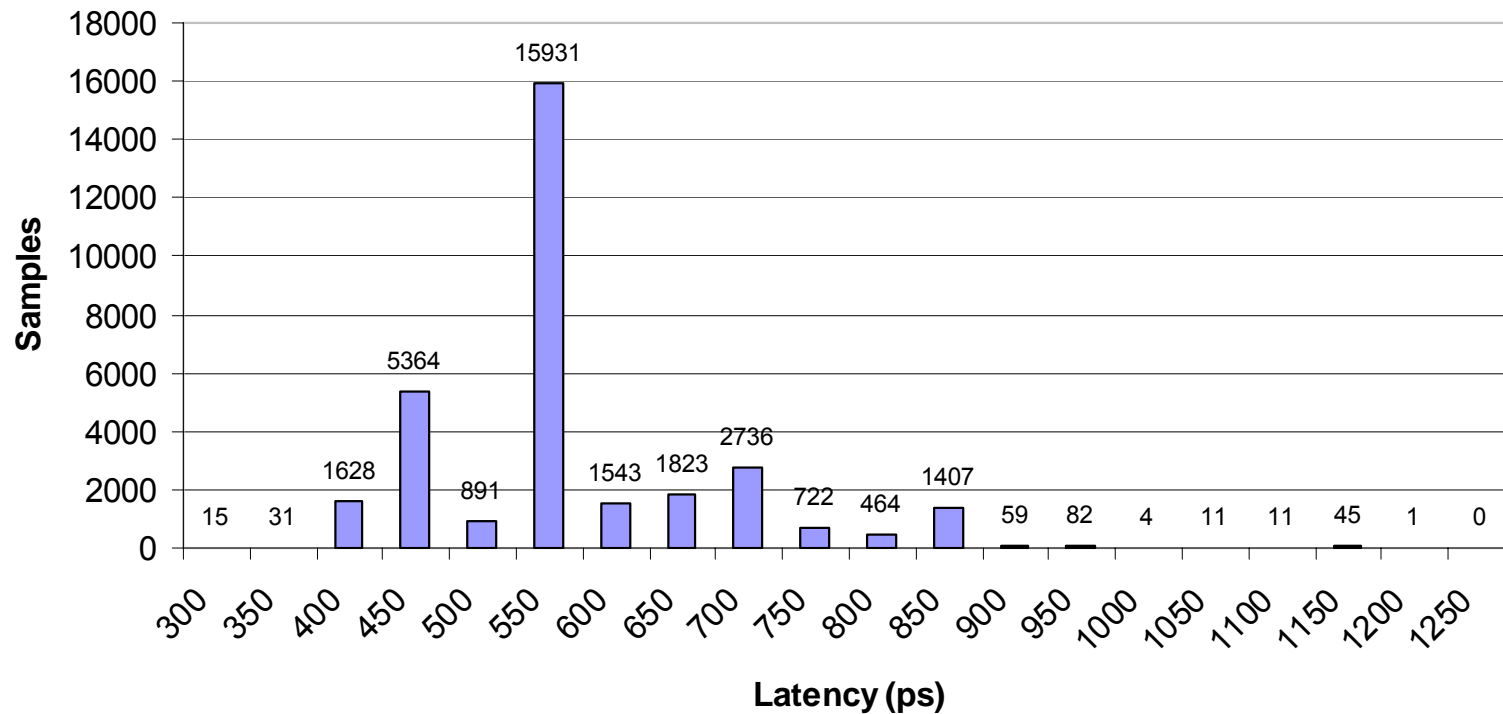




Delay Analysis with Random Vectors



64-bit Kogge-Stone Adder Latency Distribution - 32k Random Vectors
in 0.18um at 1.8V, 870 MHz, and 27 C



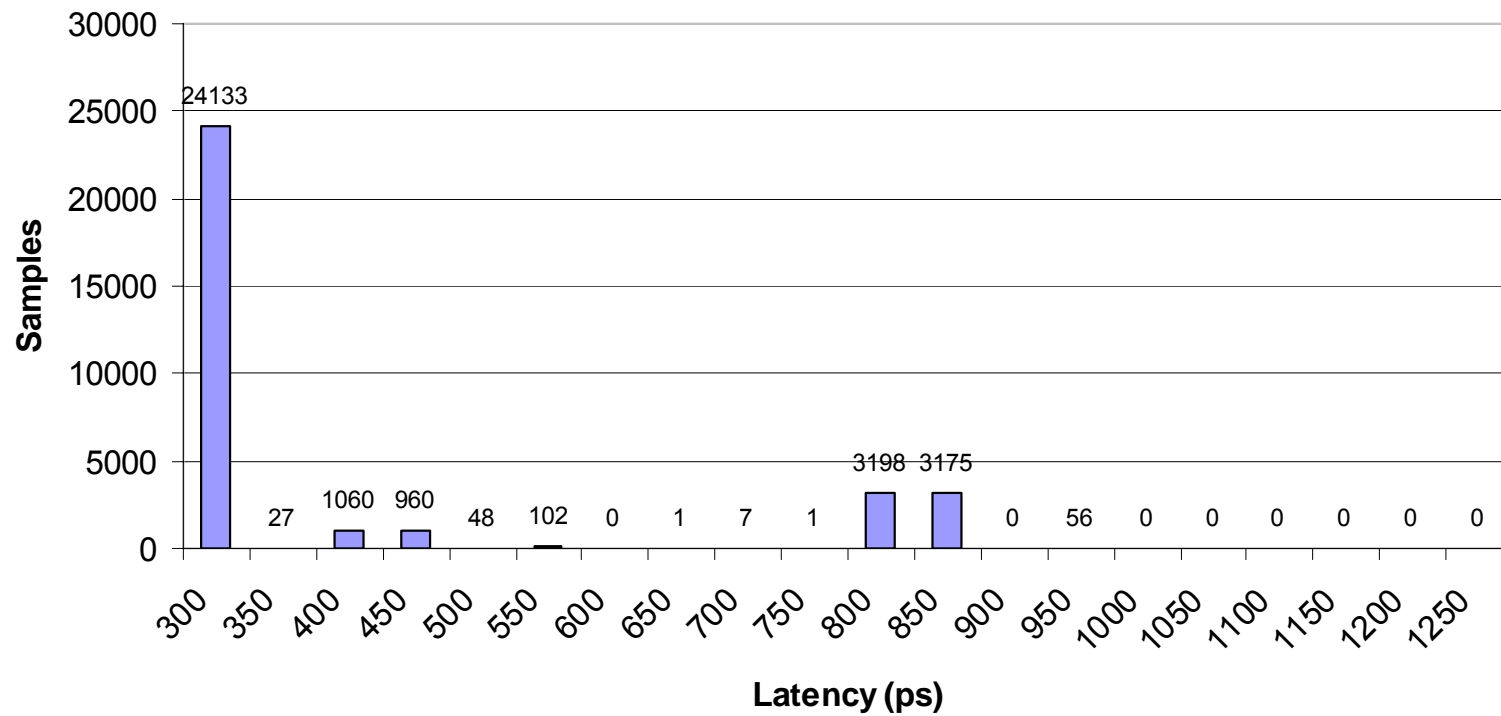
- Experimental analyses of real silicon have confirmed this phenomena



Delay Analysis with BZIP Vectors



64-bit Kogge-Stone Adder Latency Distribution - 32k BZIP Vectors
in 0.18um at 1.8V, 870 MHz, and 27 C

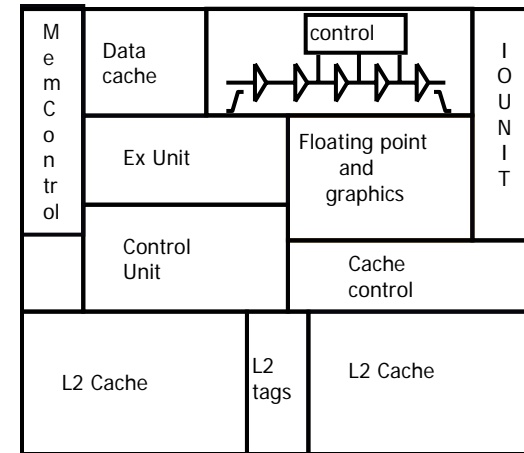




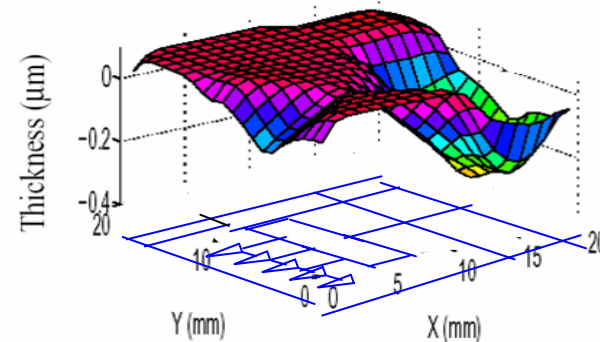
How is Operating Voltage Set?



- Voltage must be high enough for reliable operation at desired freq
 - Static timing analysis at worst-case corner sets *zero-margin* voltage
 - Voltage *safety margin* necessary to guarantee correct operation
 - Delay model uncertainties
 - Intra-die voltage fluctuations
 - Temperature variation (hot spots)
 - Intra-die process variations
 - Delay impact of noise events
- Delay line speed detectors reclaim some ambient margins
- Increasing environmental and process variation
 - Larger safety margins
 - Loss of power savings



Intra-die variations in ILD thickness

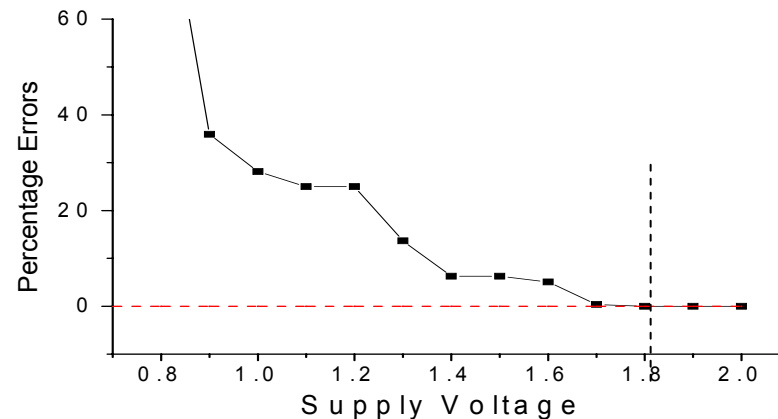
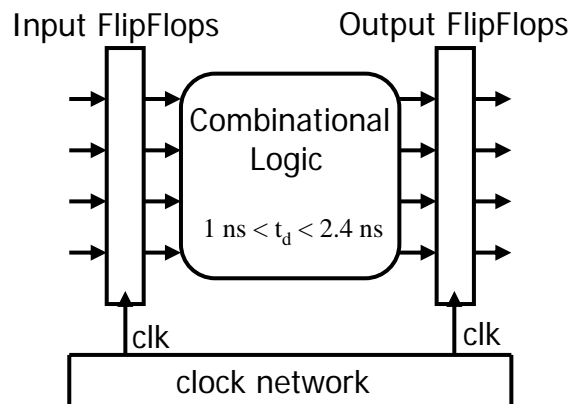




Shaving Voltage Margins with Razor



- Razor implements *in-situ* error detection and correction
 - Build in the ability to *detect and correct* circuit timing errors
 - At a given frequency, tune processor voltage based on error rate
 - Eliminate process, temperature, and safety margins (tune for near-zero errors)
 - Purposely run *below* critical voltage to capture *data-dependent latency margins*
 - Trade-off: voltage power savings vs. overhead of correction



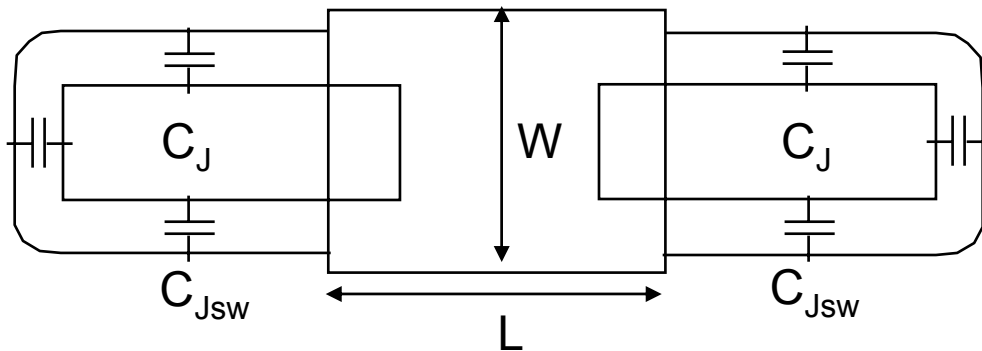
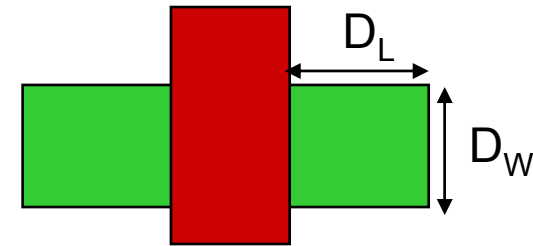
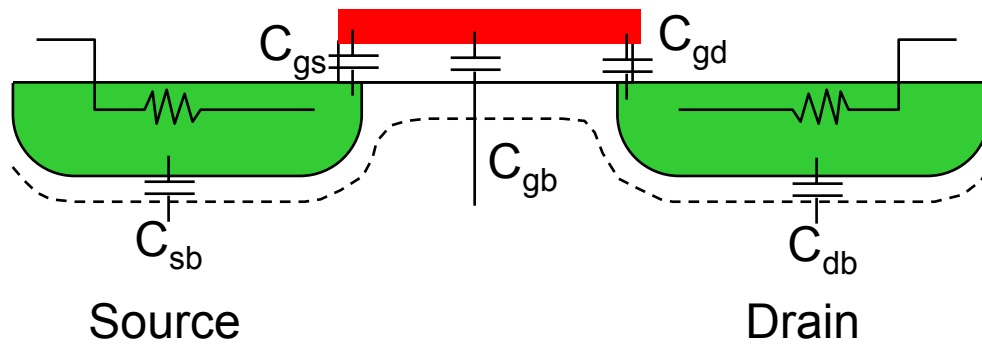
- Analogous to wireless power modulation



These also go in the poster...



First-Order Device Modeling Technique



$$AD = D_L \times D_W$$

$$PD = 2 \times D_L + D_W$$

$$C_{\text{drain area}} = C_J \times AD$$

$$C_{\text{drain sidewall}} = C_{Jsw} \times PD$$

$$C_{\text{gate}} = C_{gb} + C_{gs} + C_{gd} \approx \frac{\epsilon_{ox}}{t_{ox}} L \times W$$

$$C_{\text{drain}} = C_{\text{drain}} + C_{\text{drain sidewall}}$$

t_{ox} , ϵ_{ox} , C_J , C_{Jsw} : **SPICE** parameters

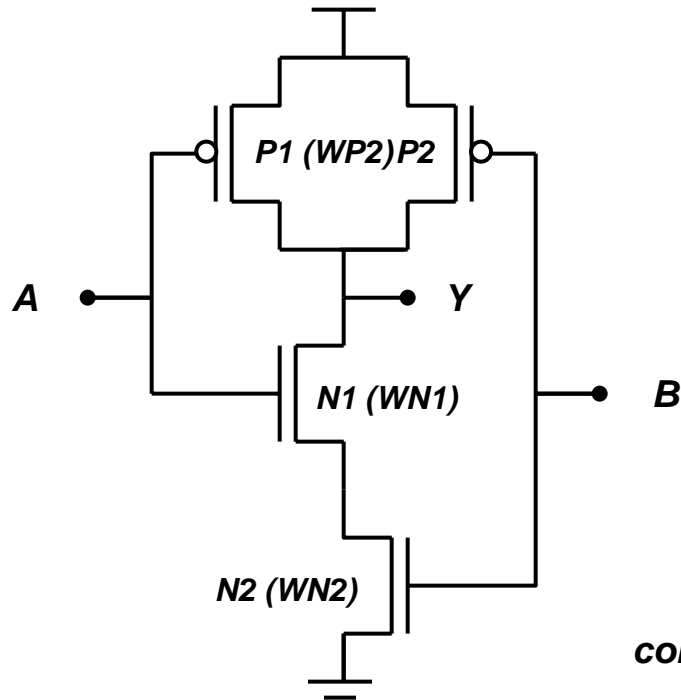
L , W , AD , PD : **layout** parameters



Creating Primitive Logic Gates



2-in / 1-out gate model (lgate2_t)



data structure creation:

```
struct node_t {
    bit_t logic_value;
    double capacitance;
};
```

```
struct logic_gate_2 {
    node_t Y;
    node_t A, B;
    double energy;
    double (*logic_gate_eval)(node_t Y,
                               node_t A,
                               node_t B)
};
```

in/out specification
logic evaluation fn

connect nodes

specify transistor size

logic primitive creation:

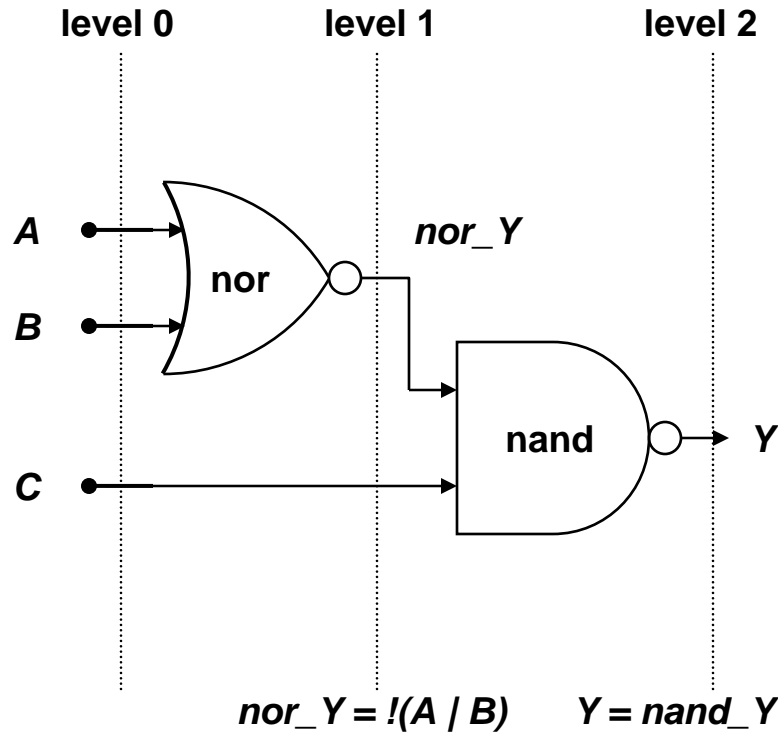
```
nand = create_logic_gate2(&A, &B, WP1, WN1, WP2, WN2);
```

logic primitive evaluation:

```
energy = logic_gate_eval(nand->Y, &A, &B, WP1, WN1, WP2, WN2) {
    Y = !(A & B);
    if(Y transits to "1")
        return nand->Y->capacitance*VDD*VDD;
}
```



Creating Complex Logic Function



data structure creation:

```
struct logic_gate_nor_nand {
    node_t Y;
    node_t A, B, C;
    logic_gate_2 nor, nand
    double energy;
    double (*logic_gate_eval)(node_t A,
                               node_t B,
                               node_t C)
};
```

logic function creation:

```
connect nodes      specify transistor size
```

```
nor = create_logic_gate2(&A, &B, WP1, WN1, WP2, WN2);
nand = create_logic_gate2(nor->Y, &C, WP1, WN1, WP2, WN2);
Y = nand->Y;
```