Digital Analog Design

Mark A. Horowitz, Metha Jeeradit, Frances Lau, Sabrina Liao, ByongChan Lim, James Mao

Electrical Engineering, Stanford University

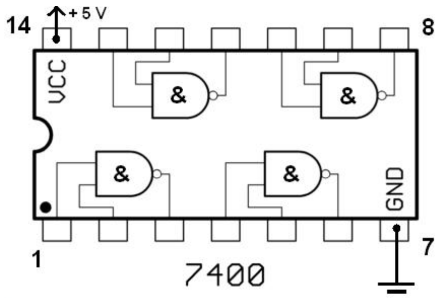
RAD

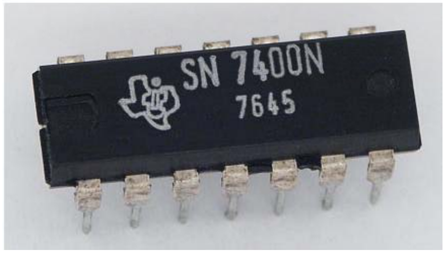
Rethinking Analog Design

STANFORD ELECTRICAL ENGINEERING

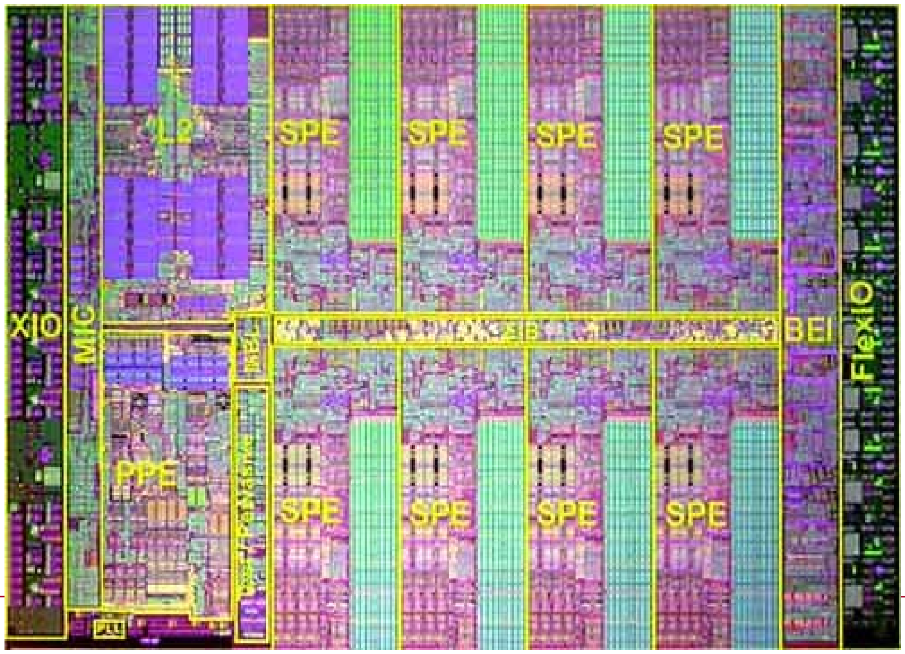
Chip Design Is Growing Up

- We have come a long way
  - From op amps and SSI components





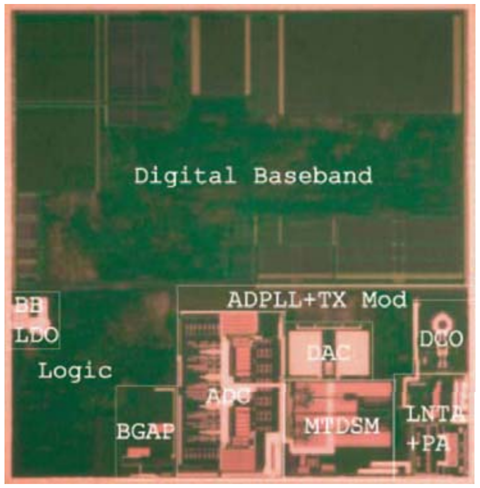
- To today's mega SOC



And Analog Design Is Getting Harder

- Requirements are growing
  - More bits, higher speeds, lower power

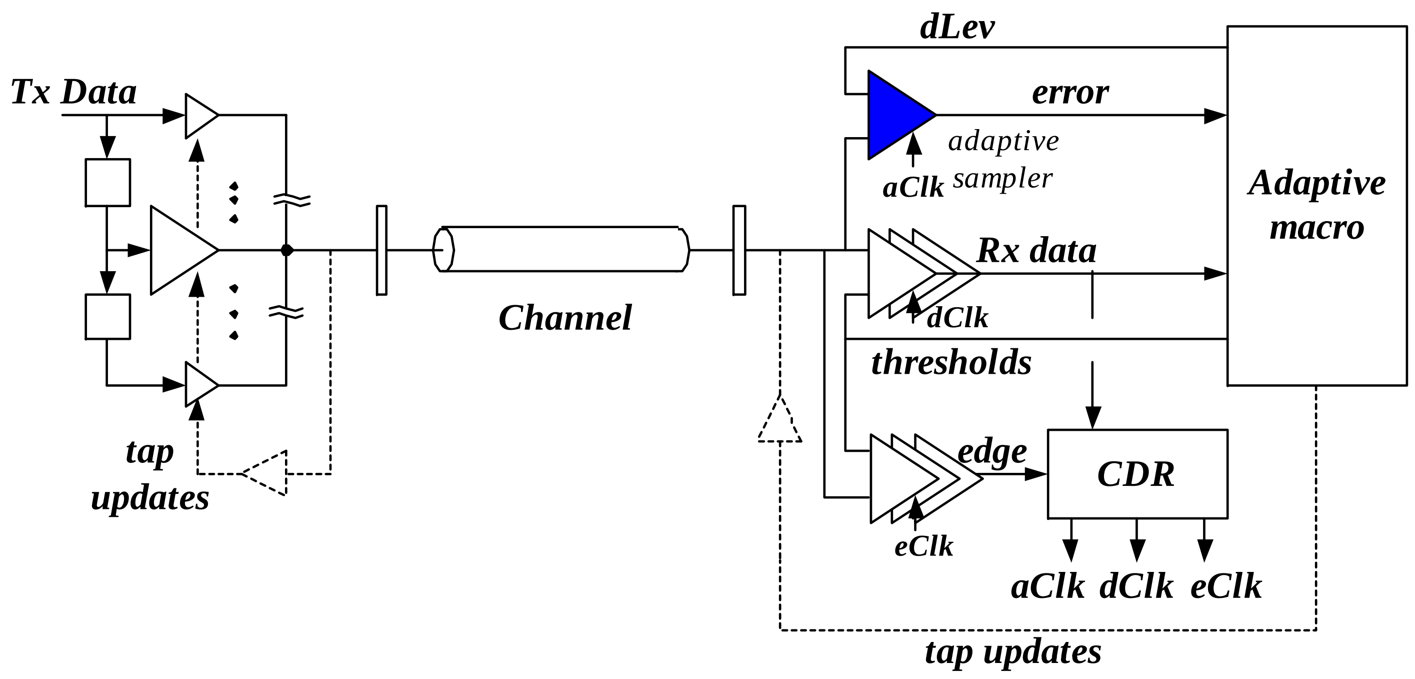
- Transistors are getting less precise
  - Ft might be better
  - Matching is worse
  - Vdd range smaller



- Analog a component in a larger system

Our Current Solutions …

- Digitally assisted analog
  - ADC calibration, PA and DAC pre-distortion, mostly digital PLLs, …



Embracing Change: System Level Analog

- What does the system really require
  - Is there a way to change the system?
- What is the minimal requirements
  - For measuring signals, might only need resolution
  - Accuracy, linearity, etc can be corrected
  - Digital logic is much easier to design
- Boris will talk more about this, and general program

Great! Are We Done?

- Is system optimized analog it?
  - Depends on how hard it is to design.
  - System optimized analog will still have some analog

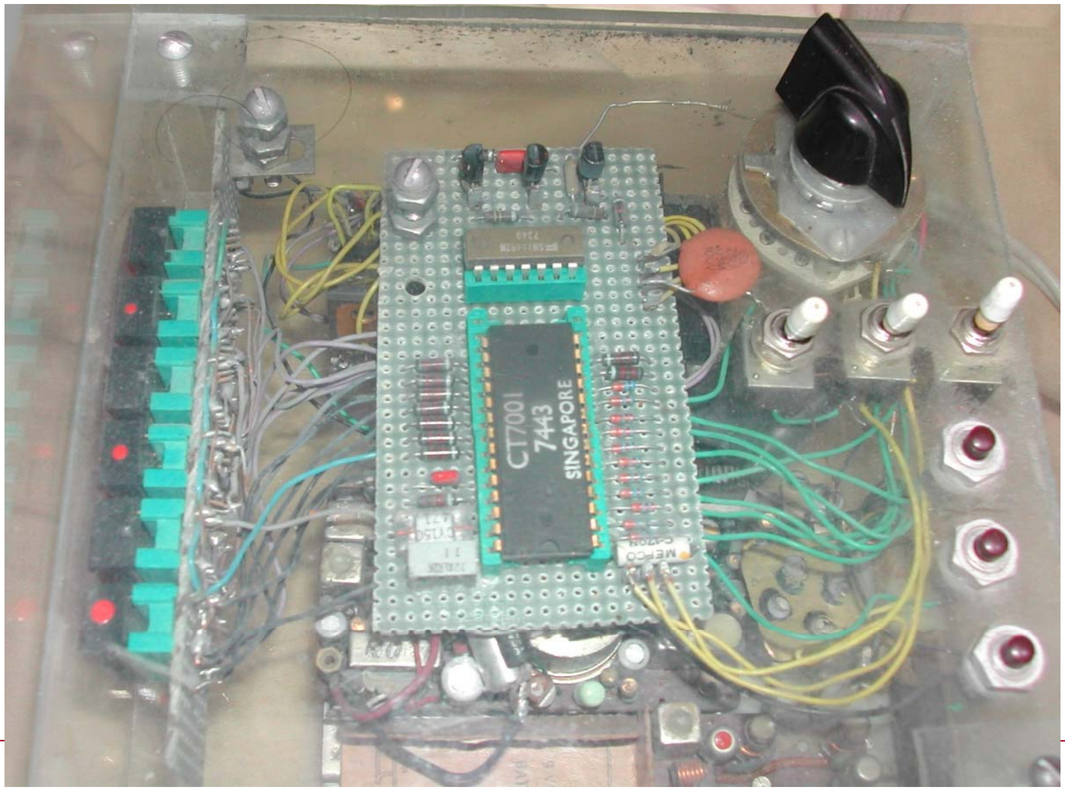
- Let’s do a brief trip through memory lane …

My Misspent Early Teens 70-74

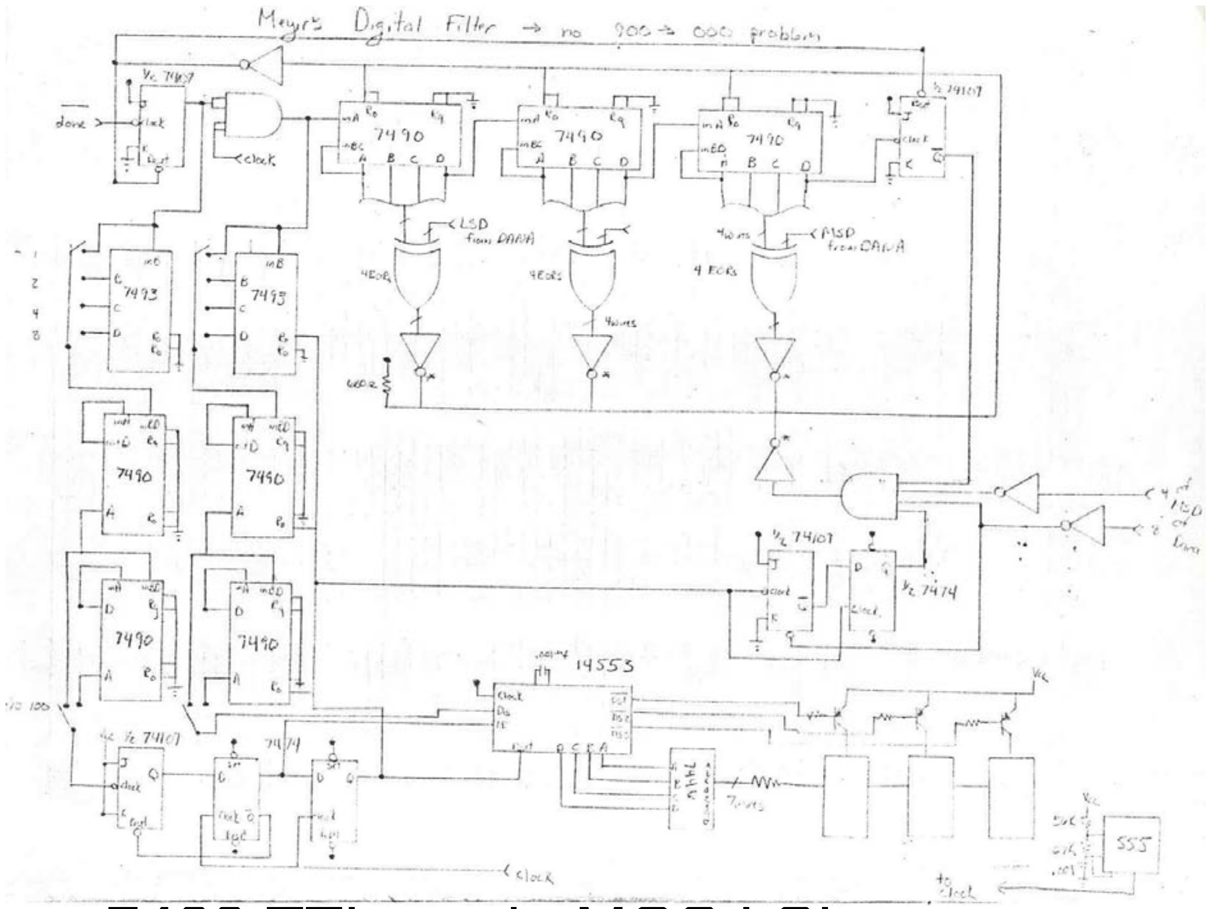
- Got my first IC around 71
  - Uncle was an EE
  - Signetics 8T80? (pre 7400)
  - Thought it was very cool, but never used it

The MIT Years 74-78

- Built lots of stuff at home
  - Calculator - 3 pMOS chips, incandescent 7-seg
  - Digital clock – 1 pMOS chip, LED 7-seg

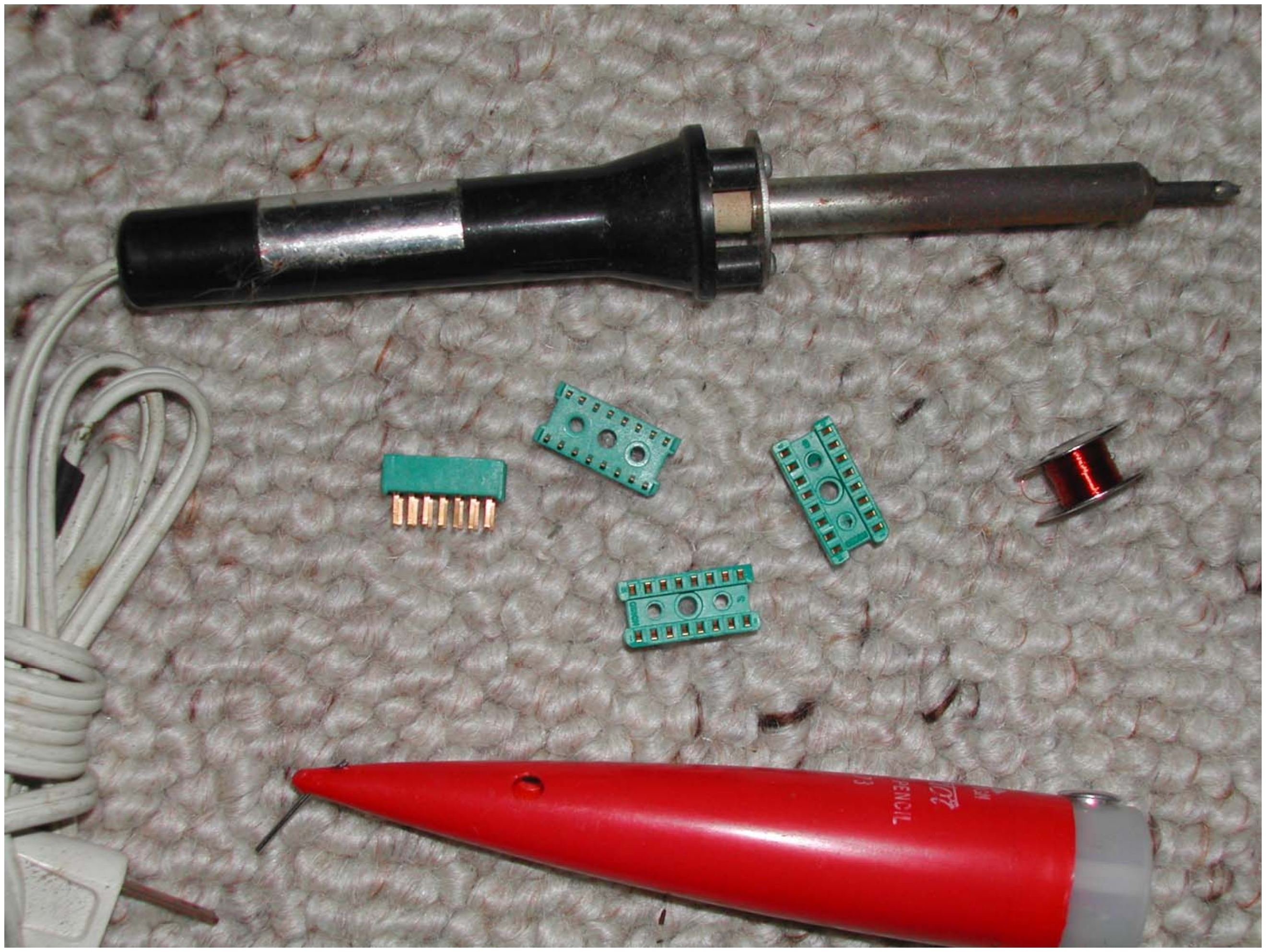


Design Tools



- Chips: 7400 TTL, and pMOS LSI

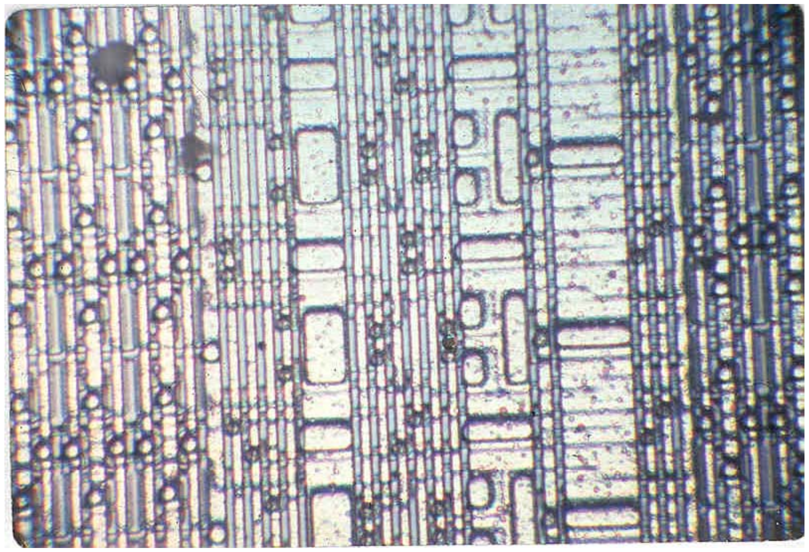
Fabrication Tools

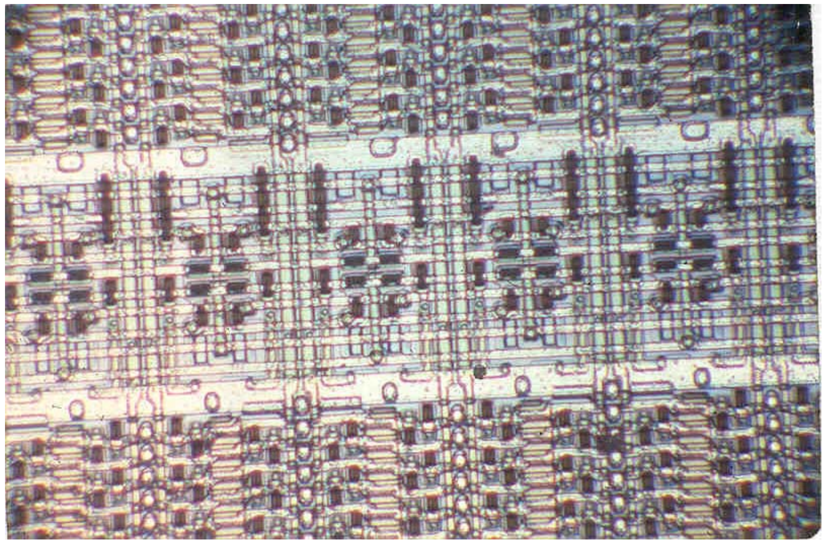


1978 – Hello Silicon Valley

Hot new technologies

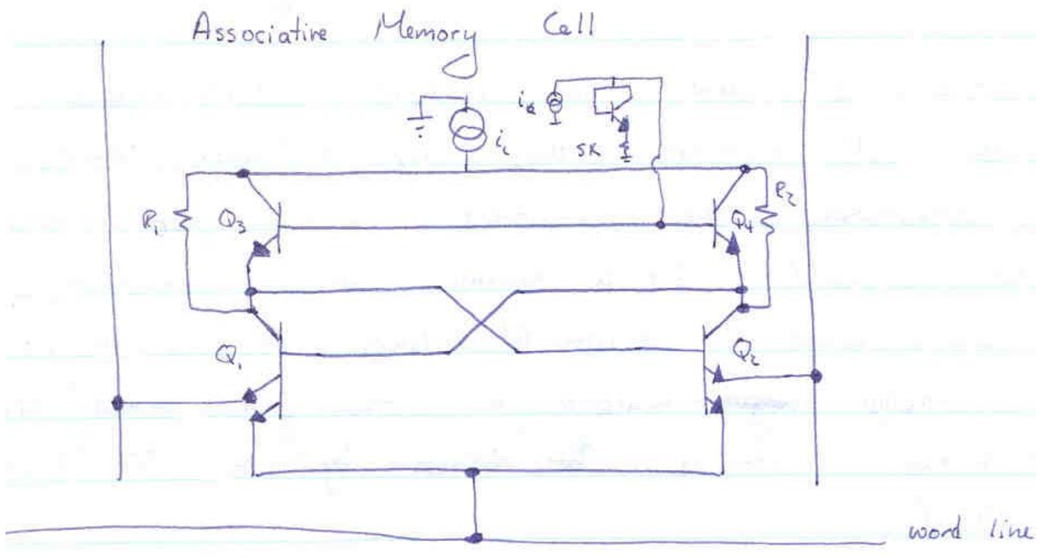
- 3μ nMOS
- Depletion loads and 5V operation, TTL I/O





Worked at Signetics

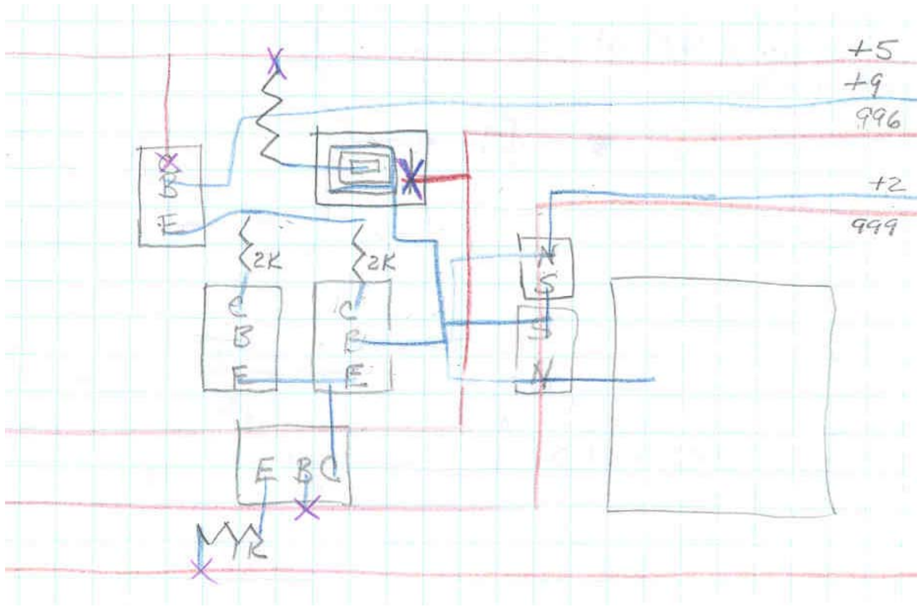
- Worked on bipolar designs
  - 1 kbit ECL CAM, ISL gate array

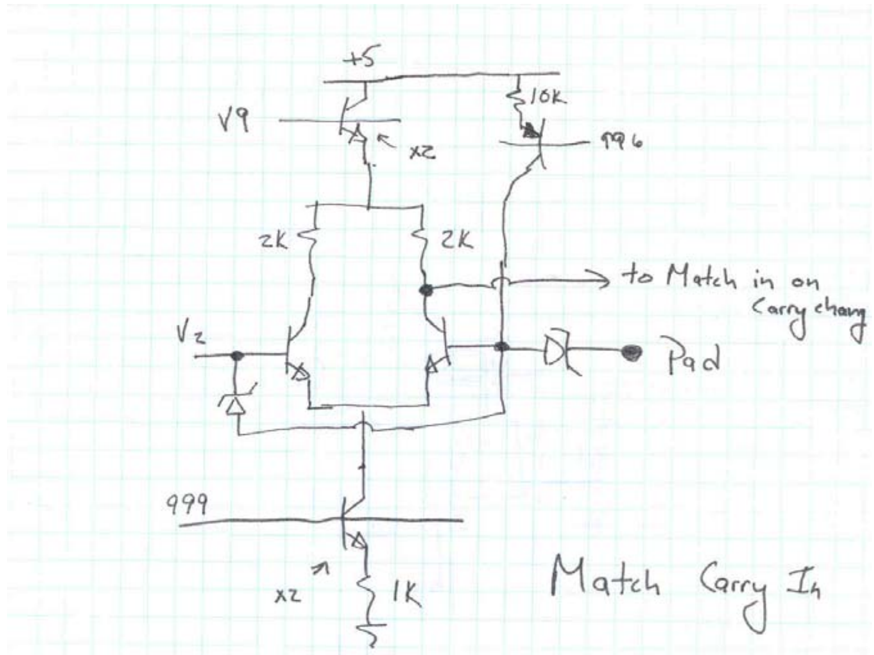


- Design Flow
  - We did have circuit simulation

Backend (Layout) Flow

- Handed schematic to layout designer
  - She produced stick diagrams to check





Note I made a mistake on the MCI cell. The load resistors should be 5K not 2K – sorry about that

Layout Flow Cont’d

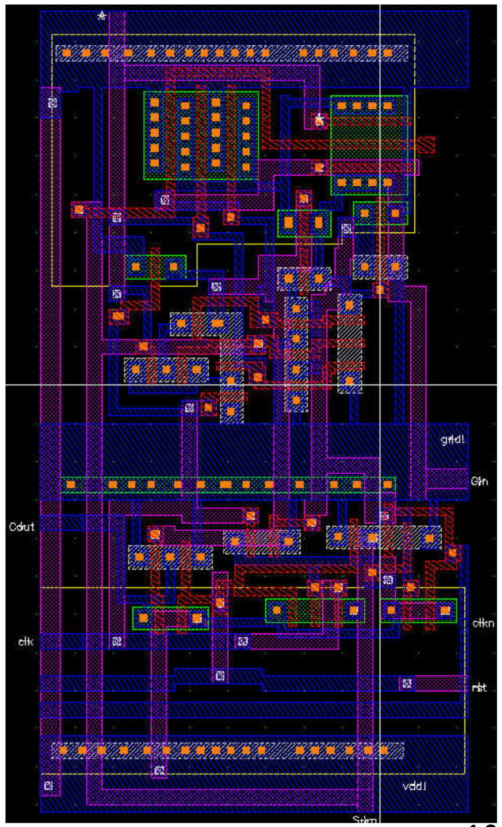
- She drew it with color pencils on mylar
    - A central group digitized it,
    - Plotted results on large flatbed pen plotters
- Manually checked DRC and ERC
- Computer DRC only before tape-out

We Have Come A Long Way,

- Digital chip design today is very different
  - Verilog input, with external IP
  - Floorplan information
  - Tools generate the chip
    - But you need to be expert in using the tools
- Design moved through many phases
  - Spice, custom layout
  - Logic, Std cells manual placement/routing
  - Synthesis, automatic placement/routing
  - SOC design – macro block reuse

Well, Maybe Not (for Analog)

- Now draw schematics on a computer
- And layout is done directly on computer too
- But the process is still manual



Analog Design

- Analog design tool scaling
  - SPICE and custom layout
  - Better SPICE and custom layout
  - Matlab, Spectre, and custom layout w/ Pcells
  - Not much change

- Basic Problem
  - Little/no encapsulation of functional blocks
    - No abstraction / ERC pairs
  - Leads to large amount of analog redesign

Management Problem

- Must port a mixed signal block to new fab
  - Old design is great
  - But former design is not working on the port
- New designer looks at block
  - And doesn’t like the way it was designed
    - It is not the way that she would do it
- There is little validation documentation around
  - So you trust the new designer
  - You need to redesign the block

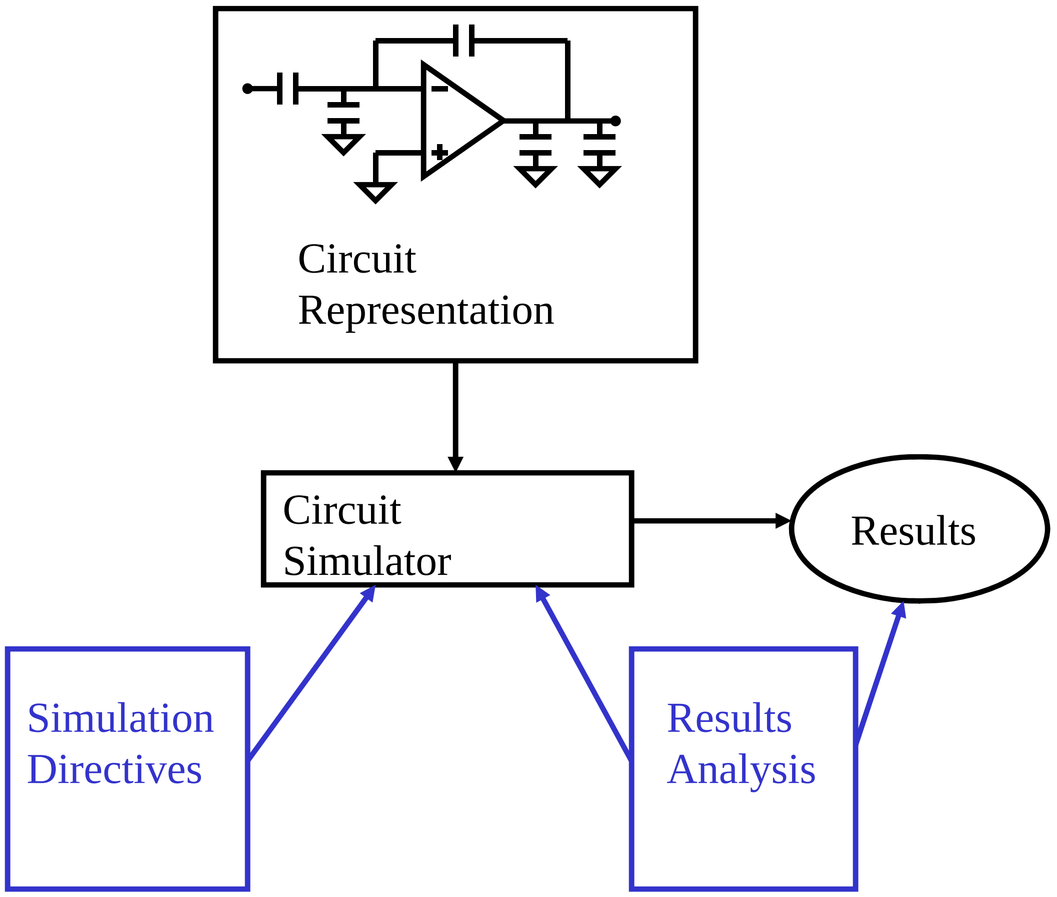
My Overall Goal: Digital Analog Design

- Don’t just use more digital gates
- Make analog design more like digital
  - Better encapsulation of function
  - Methods for system validation
  - Automatic electrical rules checking
  - Better reuse of components
- Reduce time to port design

Making an Analog Standard Cell

- Capturing the schematic is not enough
  - Would you trust someone else’s cell
- Trust digital cells
  - Since there is an electrical rule / functional checks
  - Work for every cell
- Analog designs don’t have universal ERC checks
  - So need to create them for each cell
- Capture the test routines for each cell
  - Both the functional tests, and the constraints

Tests Contain Three Parts



Types of Checks

- Test bench:
  - Contains stimuli generation and results analysis
  - Create for each major piece of the design
  - Gets run when you are “checking out” that module
- Assertions
  - Monitor operation of that module
  - Prevent the circuits from operating outside the constraints
  - Run every time the cell is run

To Reuse Analog Cells

- Need to record/archive
  - All the test-benches, and assertions
- These will be specific to an circuit type
  - No universal ERC for analog blocks
- We are starting to create an archive for these checks
  - Called Circuitbook
- Circuitbook
  - Object oriented for both circuits and tests
  - Schematic, tests, assertions, functional model

Fixing the Gap: Leveraging Abstraction

- Digital tools leverage “abstraction” effectively
  - Digital abstraction: Boolean (value), synchronous (time)
  - Leverage abstractions to:
    - Check circuits, measure coverage, check equivalence, etc.
  - Designers don’t just rely on fast circuit simulators
- Analog tools do not
  - No notion of analog abstraction
    - Focus mainly on fast simulation with accurate device models
  - Designer think *faster* SPICE is the answer
    - But it will never be fast enough
  - Causes problems with big D little A designs
    - How to do system level validation

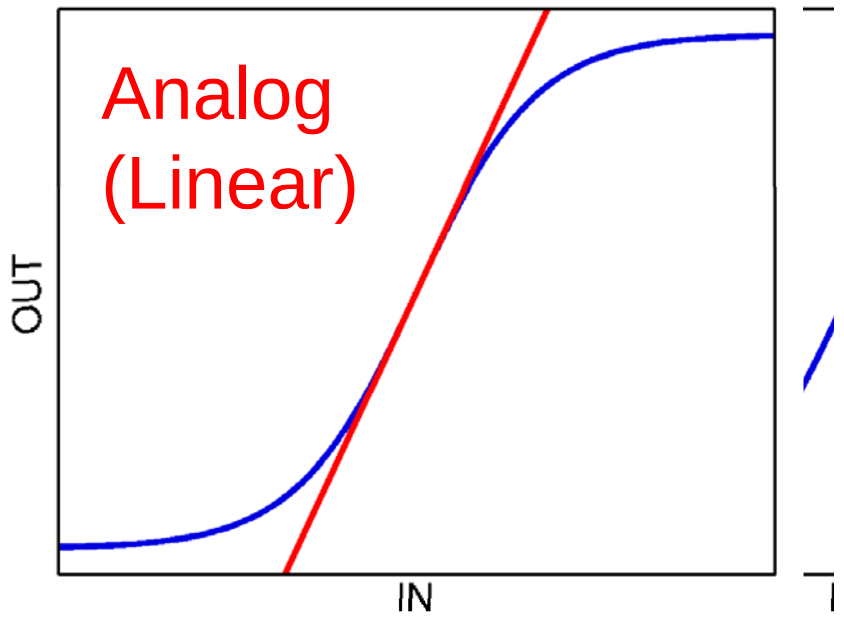
The Key Problem:

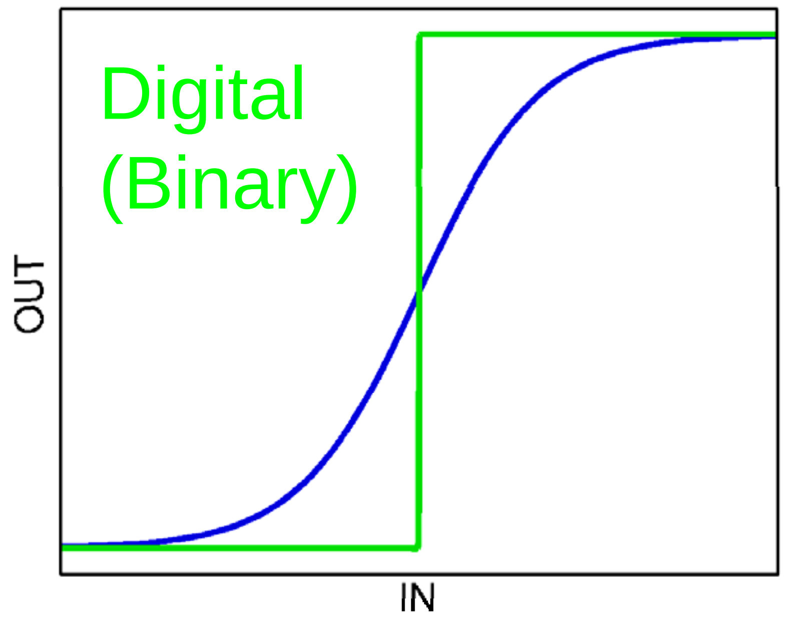
Generating an analog circuit abstraction

Analog vs. Digital

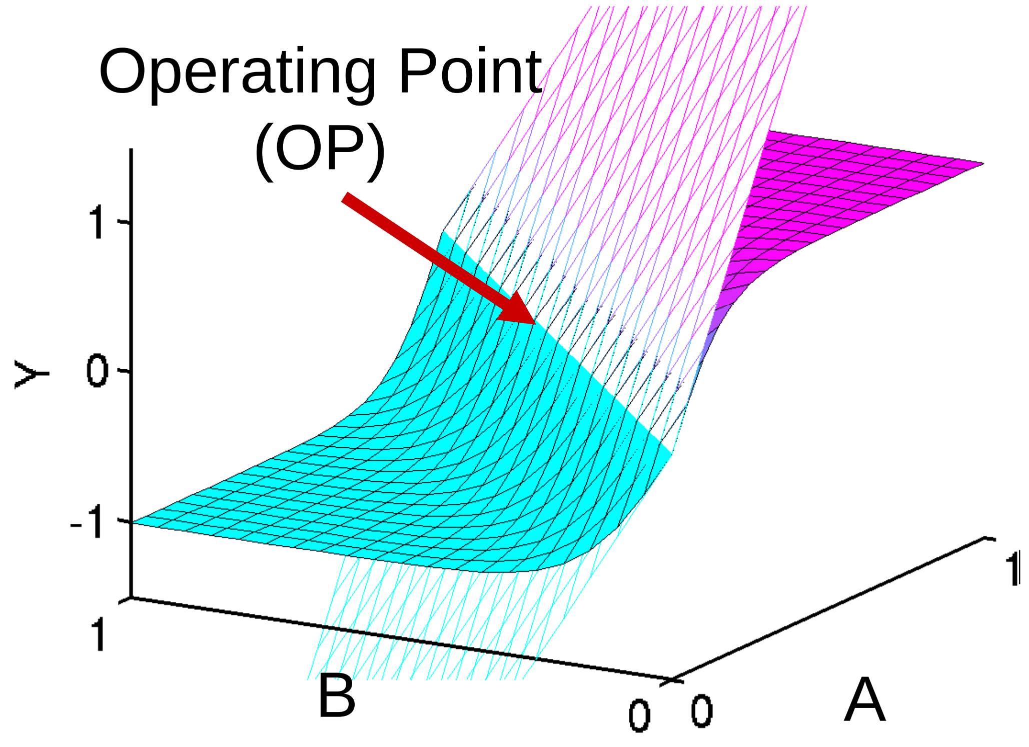
- Continuous vs. discrete?
- A and D are different in their world views

What do you see in this picture?





Analog Abstraction: Linear System



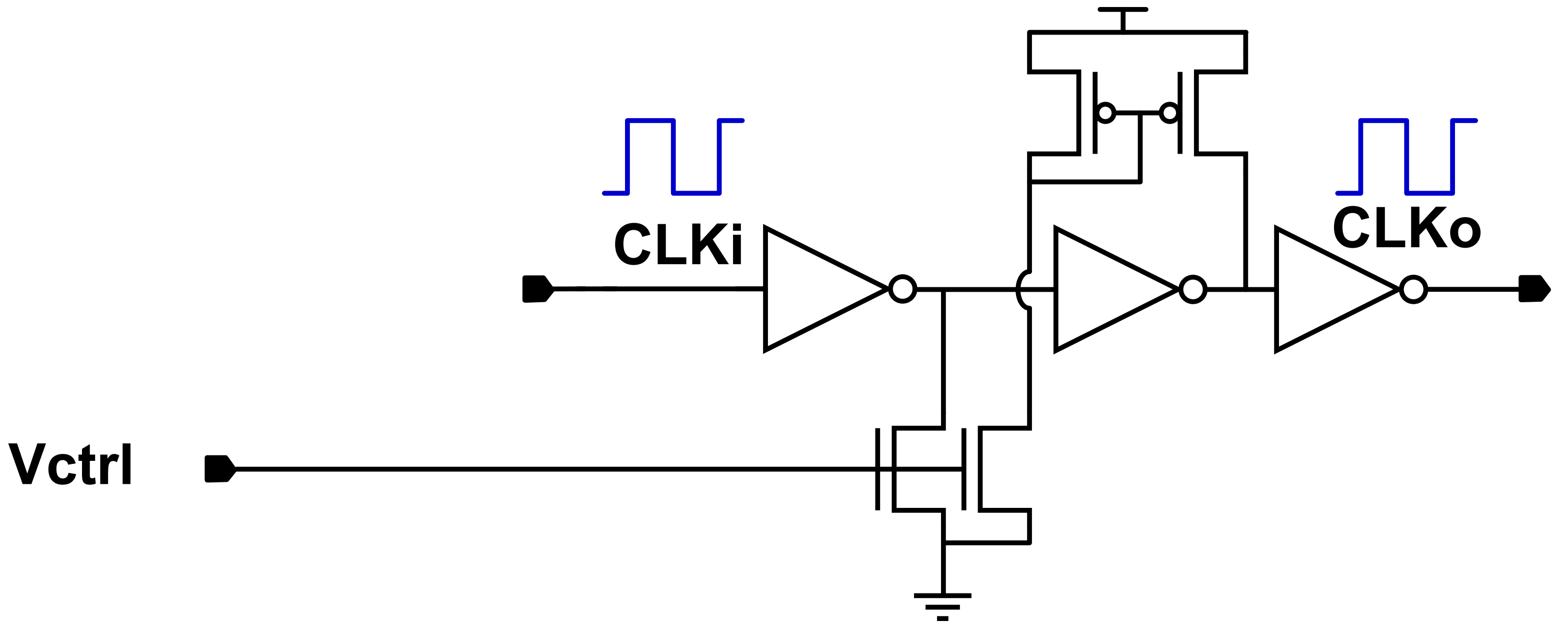
- Design intent is to use the linear region around the OP
- The ideal circuit has linear I/O relationship $\Delta Y = \alpha \cdot \Delta A + \beta \cdot \Delta B$
- In general, it's a linear dynamical system

- Our conjecture: all analog circuits have *linear intent*!
  - Then, the proper abstraction for analog is a linear system

Dealing with Non-Linear, Linear Circuits

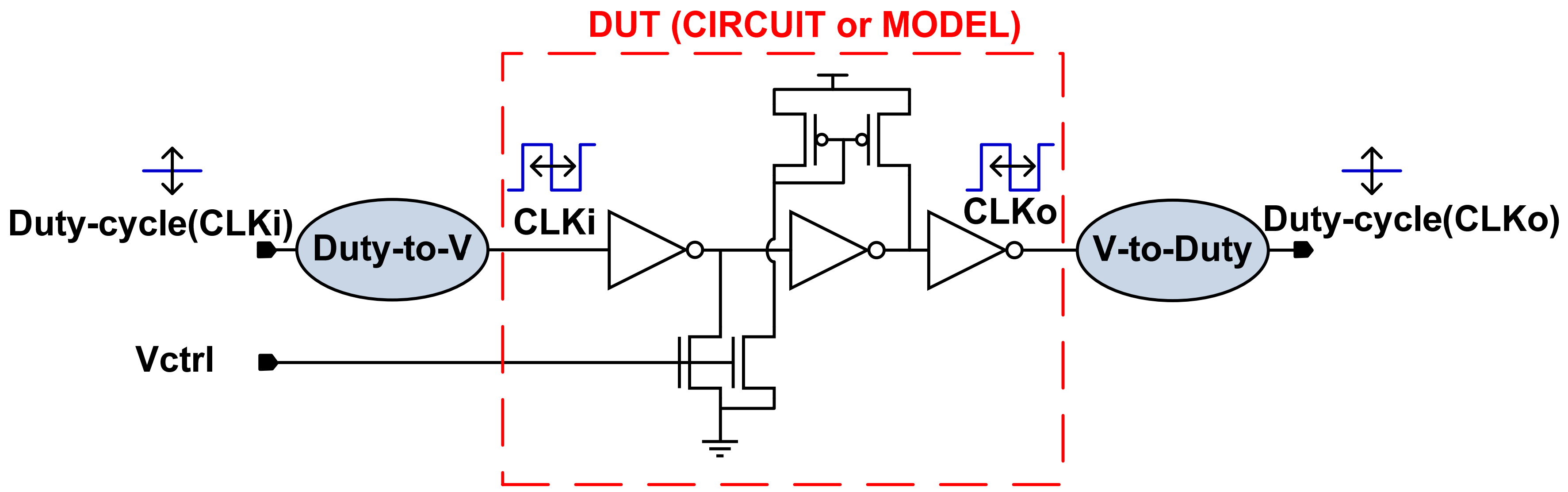
- No real circuit is linear
  - But that does not mean it doesn’t have a linear intent
  - Can we describe the circuit by its approximate linear function
    - And its deviation from that function (if needed)
    - Weakly non-linear function
- Two major types of non-linearity
  - Linear in a different domain than V,i, and t
  - Controllable systems
    - Can control gain / frequency of linear system
- Both of these are easily handled in this framework

Duty Cycle Adjuster



Variable Domain Translation

- Duty-Cycle Adjuster

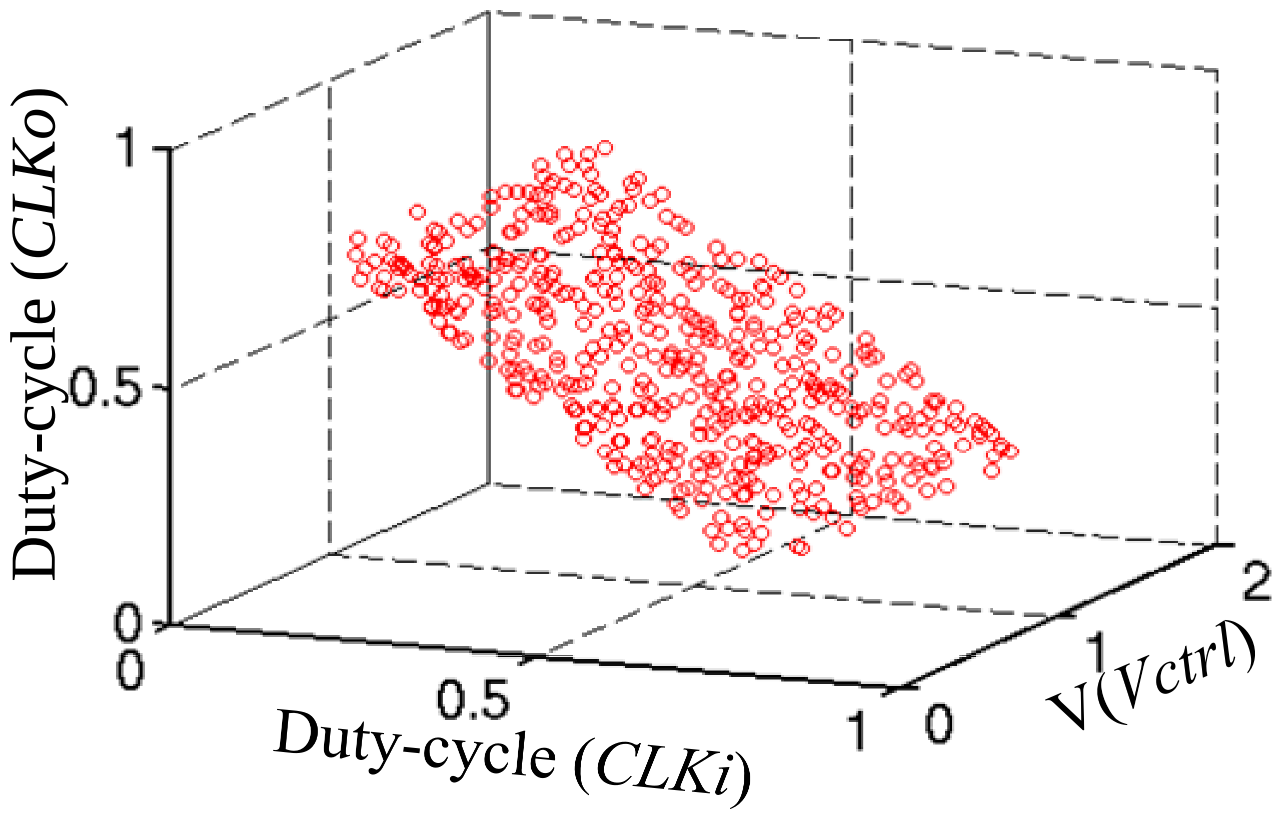


Design Intent is *Linear in Duty-cycle domain !*

$$\mathrm{Duty}(CLKo) = \alpha\cdot\mathrm{Duty}(CLKi) + \beta\cdot\mathrm{V}(Vctrl)$$

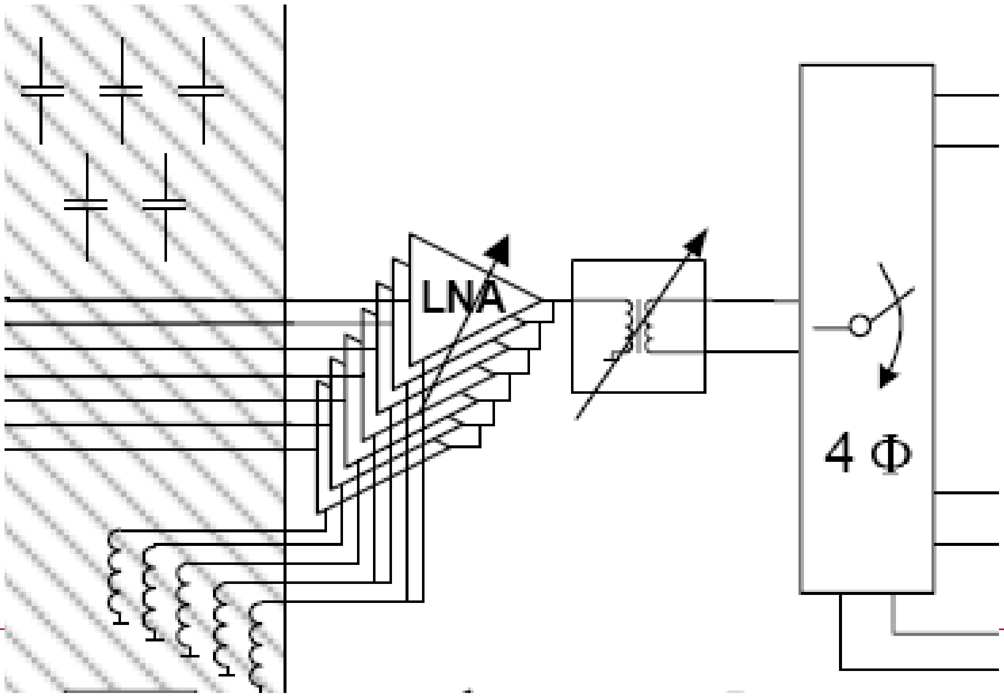
Result Surface

- Hyper-plane in *duty-cycle* domain
  - Linearity holds
    - Gain matrix comparison shows the equivalence



Controlled Linear System

- Many systems have control inputs
  - Inputs that change the system response
- We reason about these systems
  - As two coupled systems
  - So we model them that way



The Power of the Linear Abstraction

- As Boolean abstraction did for digital, the linear abstraction greatly simplifies analog verification

- The key is that superposition holds

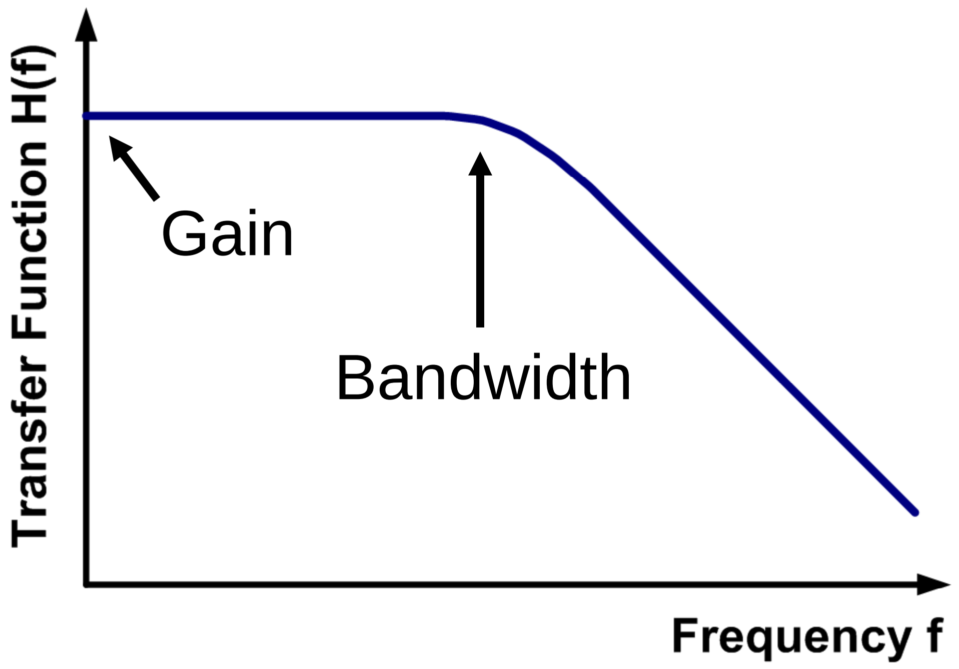
$$y = \sum_i \alpha_i \cdot x_i \qquad \text{(superposition)}$$

- This means generating input vectors is easy
  - Output is the sum of the change from each input
  - The output surface is smooth
    - Opposite of a digital system

Superposition in Time Works As Well

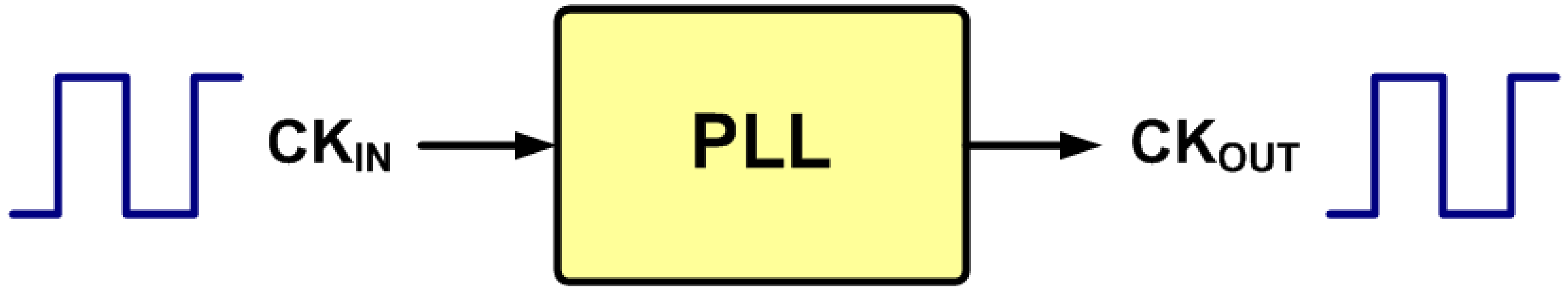
- If the intent is linear,
  - AC analysis is complete – small signal = large signal
- Thus transfer function is complete description

- TF is formal spec
  - Include sensitivities



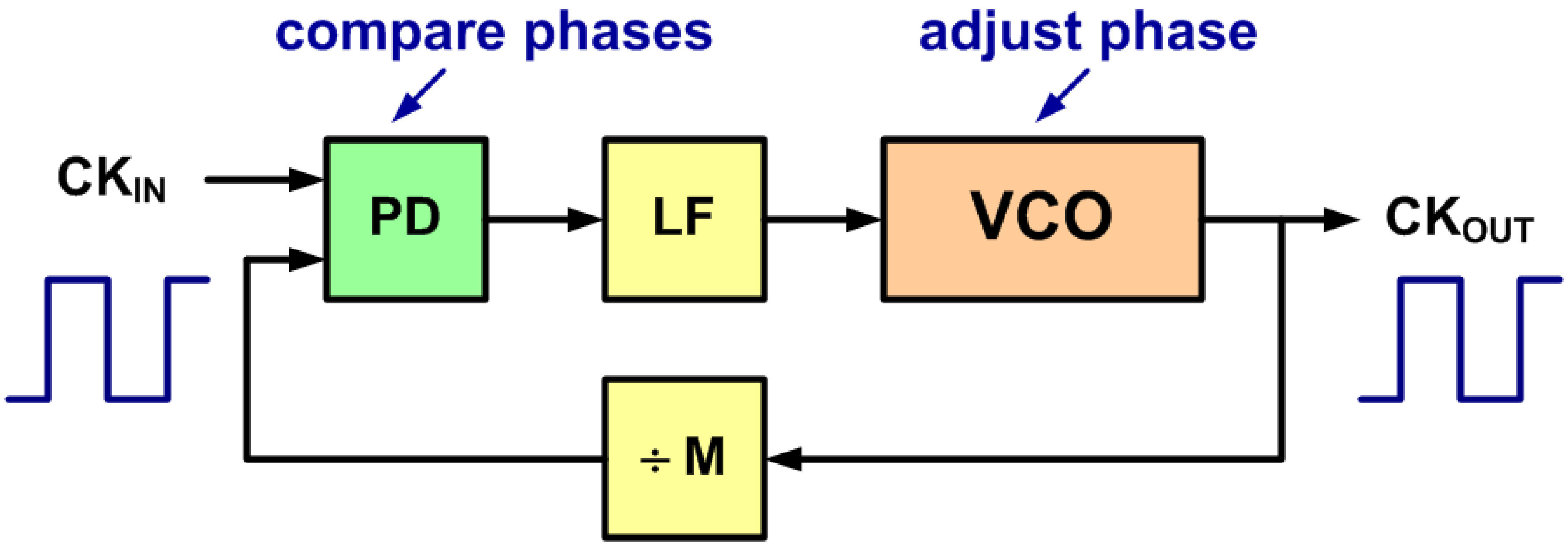
Extending AC Analysis to PLL/DLLs

- A PLL/DLL is highly nonlinear from a voltage perspective
  - Large-signal clock in, large-signal clock out



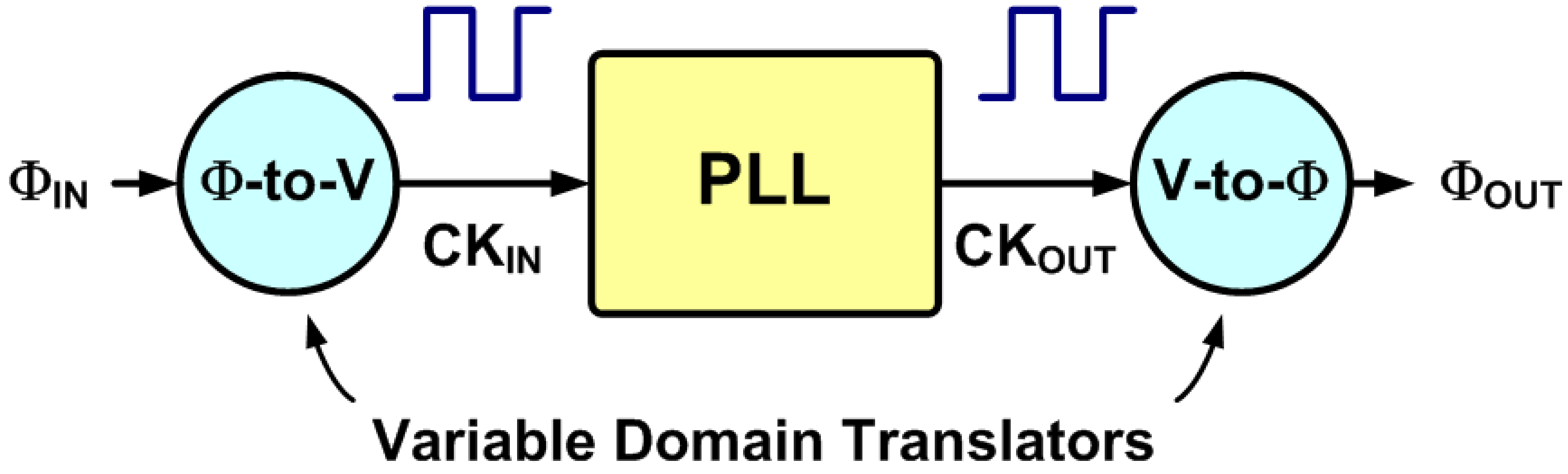
Extending AC Analysis to PLL/DLLs

- A PLL/DLL is highly nonlinear from a voltage perspective
  - Large-signal clock in, large-signal clock out
- But it is linear in its phase/delay variables
  - Can we do AC analysis in non-voltage/current variables?



Variable Domain Transformation

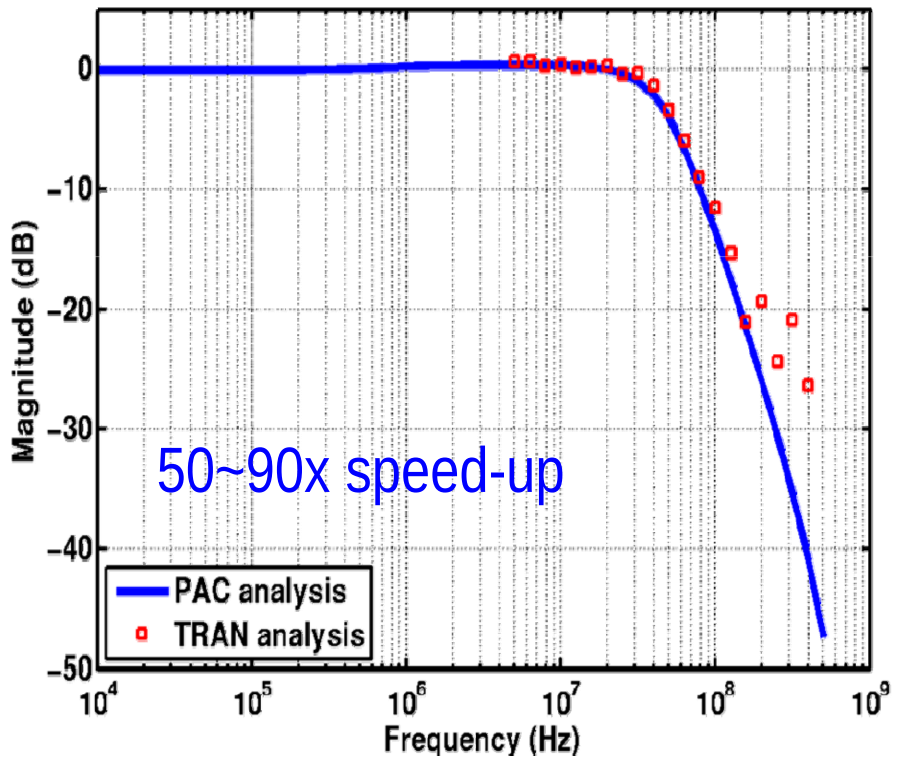
- Use translator modules
  - For SPICE write them in Verilog-A
  - Verilog-D model just inputs/outputs phase
    - If duty-cycle is important too, need 2 phases

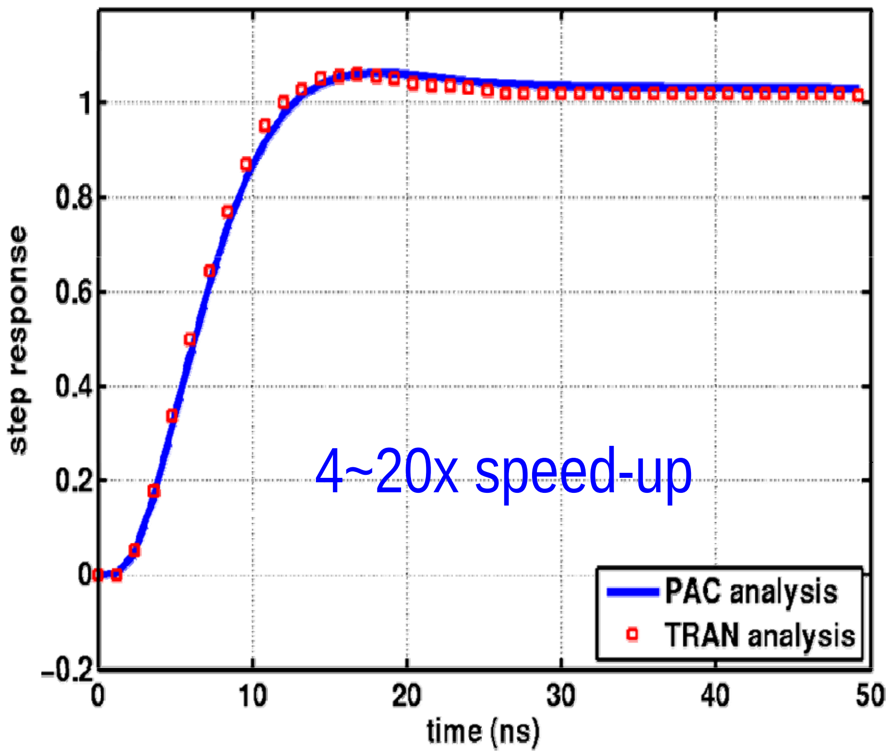


* J. Kim, et al., "Variable Domain Transformation for Linear PAC Analysis of Mixed-Signal Systems," ICCAD'07.

PLL Transfer Function Example

- AC analysis is always more efficient than transient sims
  - Option 1: explicit sinusoidal excitation at various frequencies
  - Option 2: system identification from step response





Extending AC to Stochastic Systems

- AC should be the best way to verify their linear intent
  - But they have neither DC nor periodic steady states

- They do have steady states – in a *stochastic* sense!
  - Steady state is an ensemble of waveforms with probabilities
  - e.g. PDF (jitter histogram), PSD (noise spectrum), etc.

- Then, the required steps are:
  - First, find the stochastic steady-state (SSS) of these systems
  - Second, linearize the system at SSS to measure the AC TF

* J. Kim, et al., “Stochastic Steady-State and AC Analyses of Mixed-Signal Systems,” DAC’09.

Stochastic SS and AC Analysis

- Model circuit/system as a Markov chain

$$\boxed{\mathbf{p}[n+1] = \mathbf{T} \cdot \mathbf{p}[n]}$$

where $\mathbf{p}$ is a probability vector and $\mathbf{T}$ is a transition probability matrix

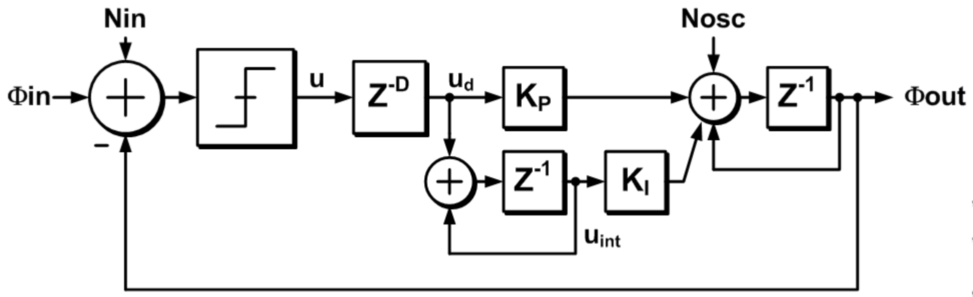
- Once the steady-state solution π is found, the system can be linearized around its stochastic steady-state:

$$\boxed{\delta\boldsymbol{\pi}[n+1] = \mathbf{T} \cdot \delta\boldsymbol{\pi}[n] + \left({}^{\partial \mathbf{T}}\!\big/_{\partial u} \cdot \boldsymbol{\pi} \right) u}$$

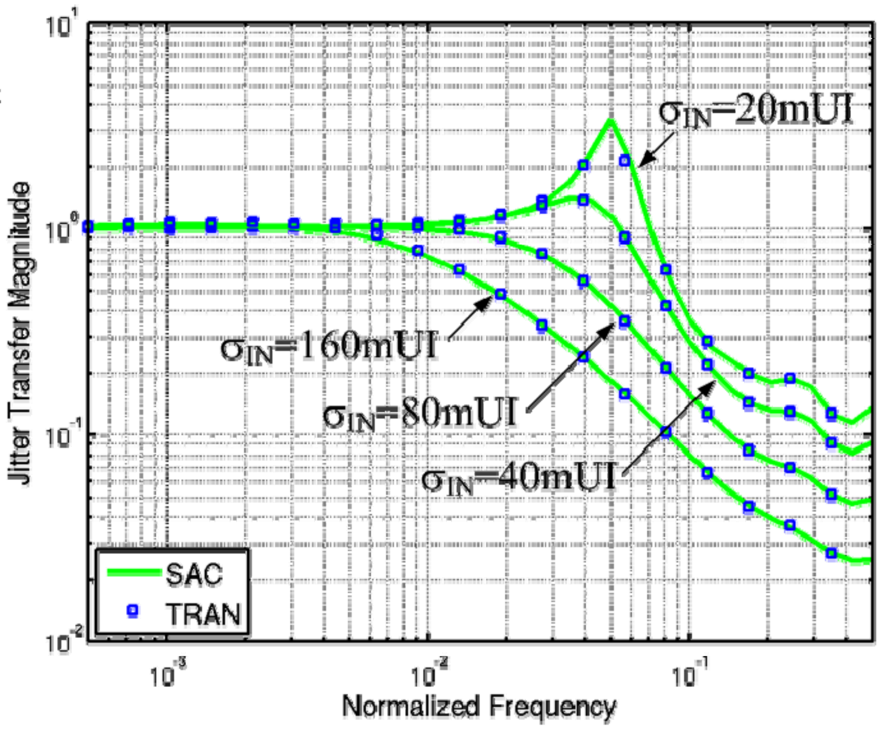
* J. Kim, et al., “Stochastic Steady-State and AC Analyses of Mixed-Signal Systems,” DAC’09.

Example: Second-Order Binary PLL

- Jitter transfer functions with various input jitter level (σ_{in})
  - Provides accurate results with 5~9x speed up vs. TRAN
  - Our algorithms keep the # of states in the Markov chain low



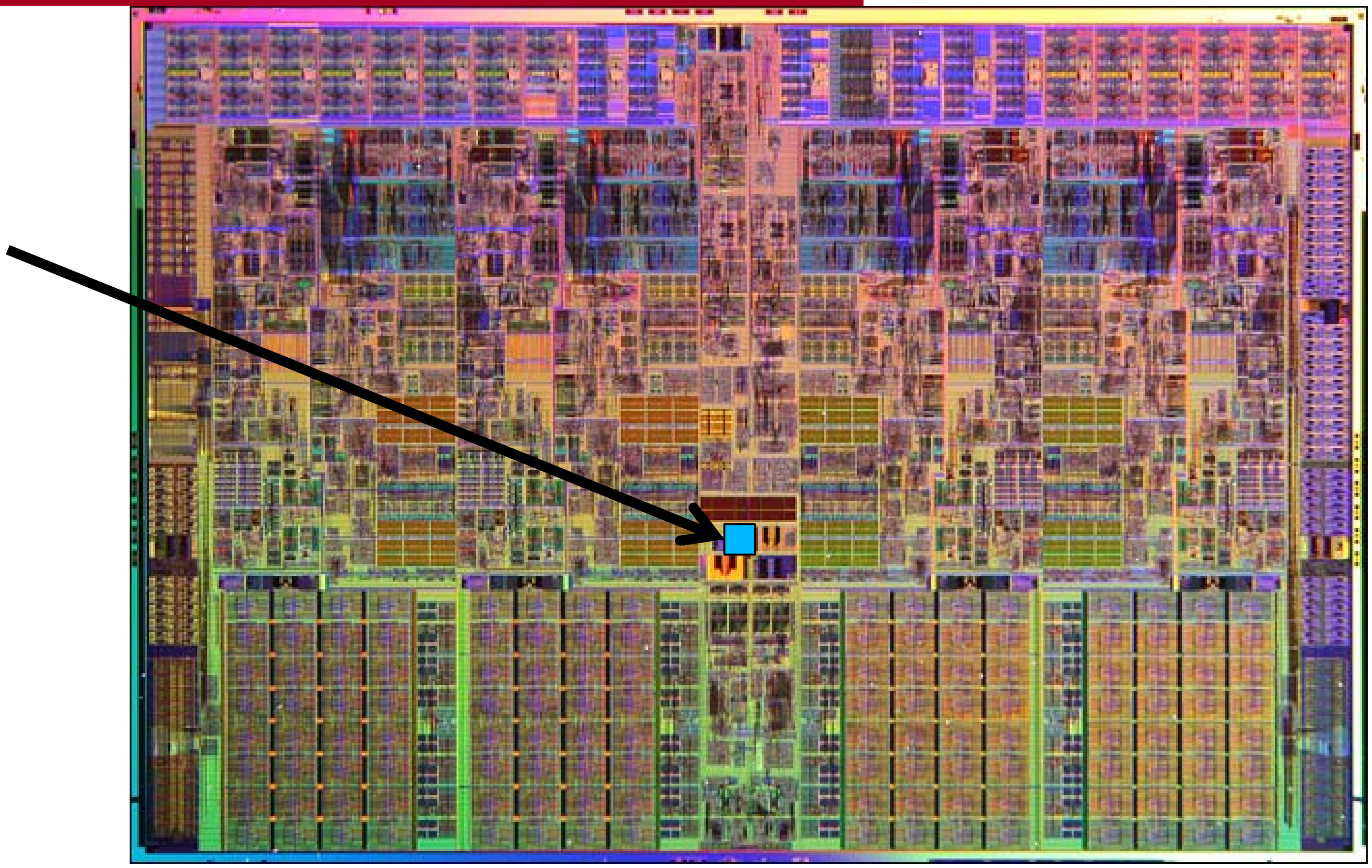
σin	# states	Time (SSS/SAC vs. TRAN)
20mUI	3496	33.6 sec vs. 316 sec
40mUI	4301	43.3 sec vs. 315 sec
80mUI	5555	59.5 sec vs. 316 sec



Linear Analysis – Summary

- It is a strong way to reason about systems
  - Provides powerful tools to use to understand operation
- And yes we know that no circuits a really linear
  - But most of the system behavior uses linear models
  - Linear models is how most designers think about design
- But digital circuits are not really digital either
  - There are checks to make sure it operations in digital mode
  - And you need to have checks for linear operation as well

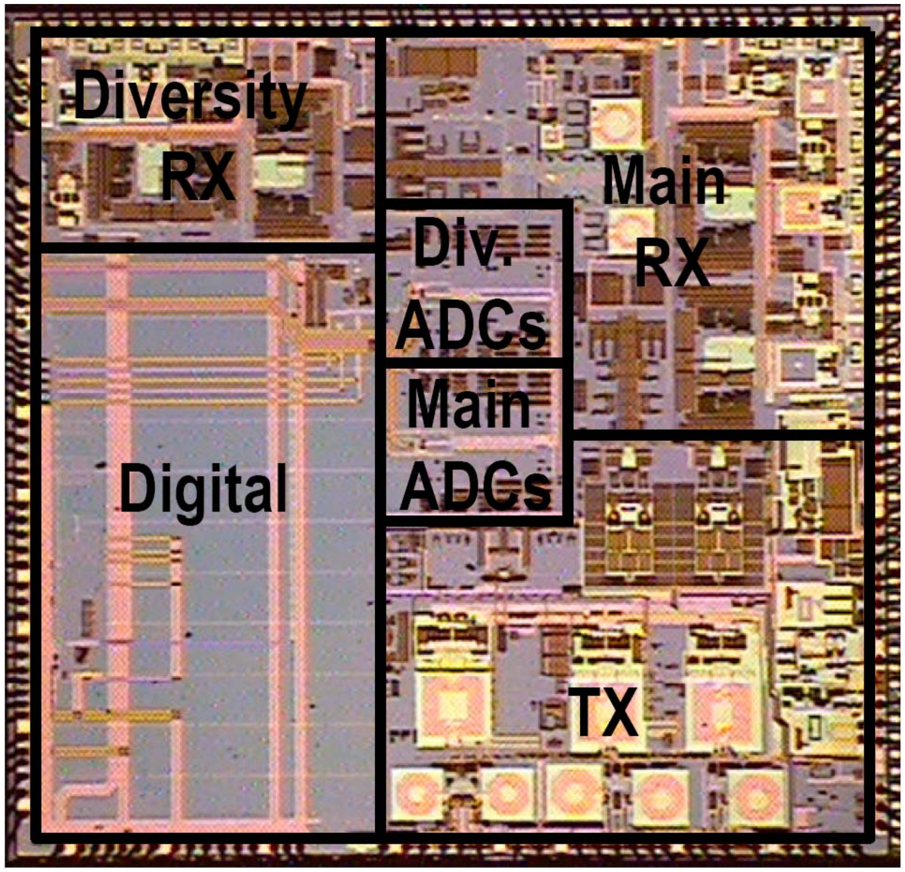
The Validation Problem:



- Really big D and very little a

Modern Analog Design

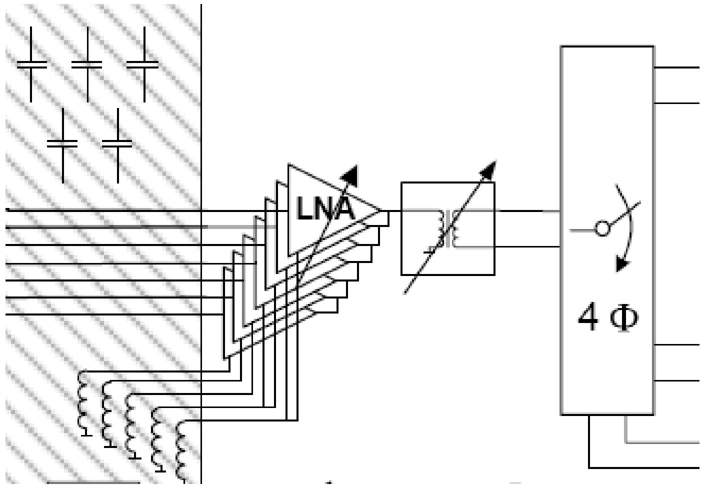
- Even in analog chips
  - Most of the transistors are in digital logic
- Still
  - Big D, little a



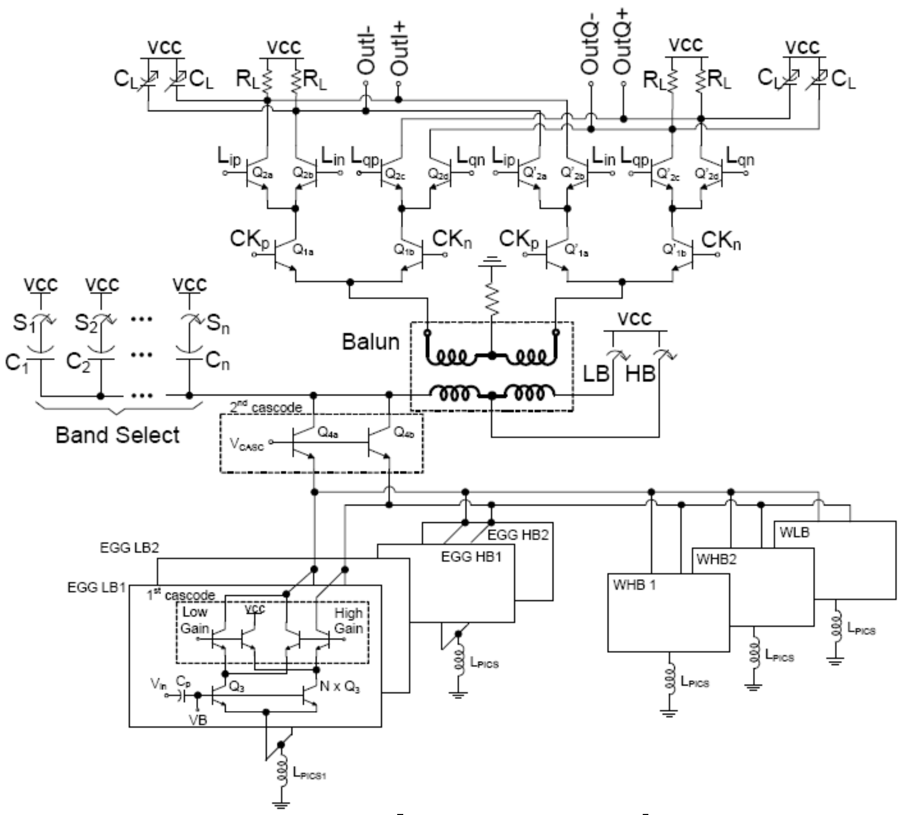
Single-Chip Multiband WCDMA/HSDPA/HSUPA/EGPRS Transceiver with Diversity Receiver and 3G DigRF Interface Without SAW Filters in Transmitter / 3G Receiver Paths, ISSCC 2009

The Model Problem

- Which really matters



Model



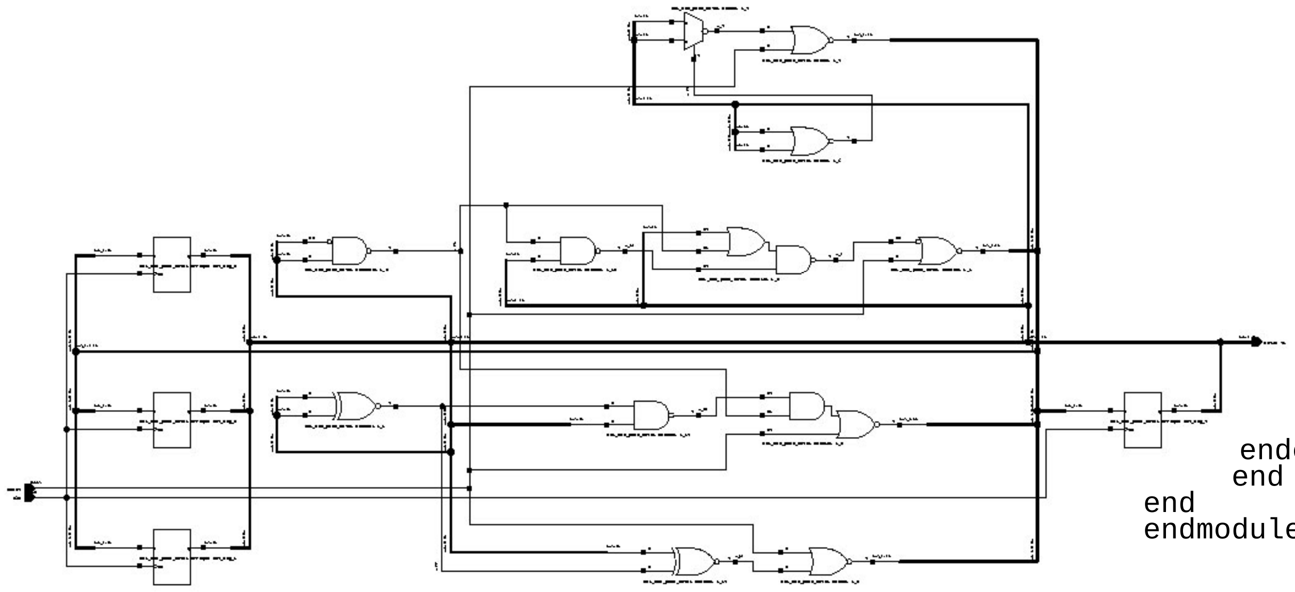
Implementation

A SAW-Less Multiband WEDGE Receiver, ISSCC 2009

The Model Problem, cont'd

- Which really matters here?

Implementation



Model

```
module gray(clk, reset,out);

input clk, reset;
output [3:0] out;

wire clk,reset;

reg [3:0] out;

always @(posedge clk)
begin
     if(reset == 1) out = 4'b0000;
     else begin
      case(out)
           4'b0000: out = 4'b0001;
           4'b0001: out = 4'b0011;
           4'b0010: out = 4'b0110;
           4'b0011: out = 4'b0010;
           4'b0100: out = 4'b1100;
           4'b0101: out = 4'b0100;
           4'b0110: out = 4'b0111;
           4'b0111: out = 4'b0101;
           4'b1000: out = 4'b0000;
           4'b1001: out = 4'b1000;
           4'b1010: out = 4'b1011;
           4'b1011: out = 4'b1001;
           4'b1100: out = 4'b1101;
           4'b1101: out = 4'b1111;
           4'b1110: out = 4'b1010;
           4'b1111: out = 4'b1110;
      endcase
     end
end
endmodule
```

The Problem:

- Digital designers control validation
  - They believe their “model” of the chip
- But for analog designers
  - That model is an approximation
    - No one would be so stupid to believe a model
  - They validated the circuit
- Leads to errors in mixed signal design
  - Bugs slip when digital designers *trust* analog models
  - Many bugs are trivial:
    - Mislabeled pins, inverted polarity, wrong bus ordering/encoding, missing connections, etc.
  - Even worse, bugs are repeated

The Solution – Model First Design

- The validation engineers will win
  - So the model really does matter

- Need to change mixed signal design
  - But they really want to have a high-level model too
  - Need to estimate overall system performance
    - Often done in matlab/simulink
  - Big change
    - The model becomes the spec
    - Circuit needs to match the model

- Only way to ensure two descriptions match:
  - Have model / circuit regressions checks

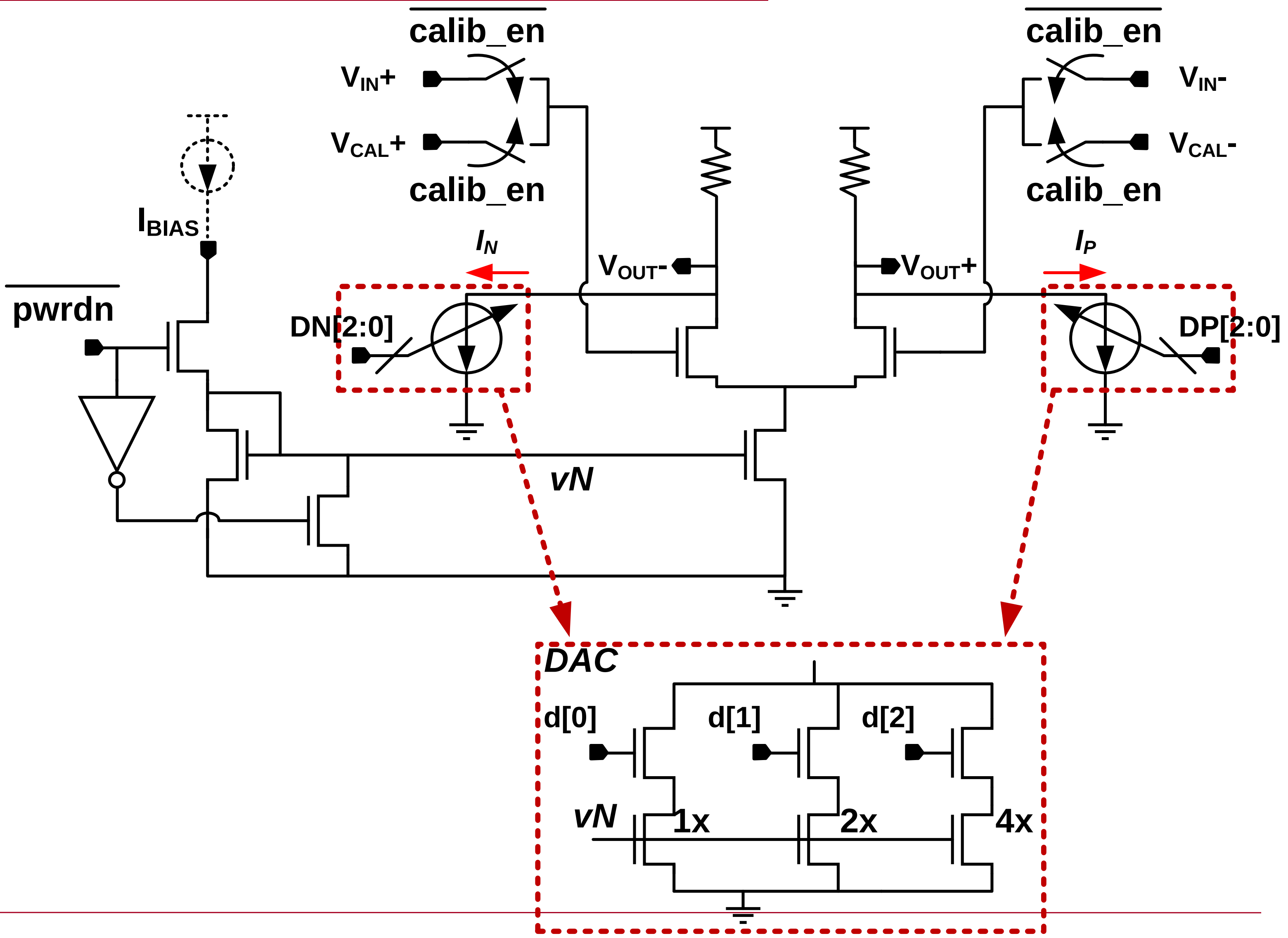
Analog Functional Specification

- For a linear system
  - Matrix of transfer functions, from each input to each output
- For a non-linear, linear system
  - Set of domain translators, and transfer matrix and/or
  - Two sets of transfer matrices
    - One from control inputs to control parameters
    - The other is a matrix which is a function of control parameters
- Use this framework for to validate functional model
  - Ultimately we might be able to generate the model directly

Validating Analog Functional Models

- Create an equivalence checker
  - Compares functional model with circuit implementation
  - Similar to Boolean equivalence checkers for digital std cells
  - We are going to use the linear model abstraction
- Functional / circuit comparison
  - Create a “spanning” set of test vectors
    - Oversample to ensure linear model is valid
  - Use set of domain translators
    - To convert to “linear” projection, and relate outputs
  - Run vectors through both simulators
  - Compare transfer matrices that are generated
    - Match if matrices are close enough

Example

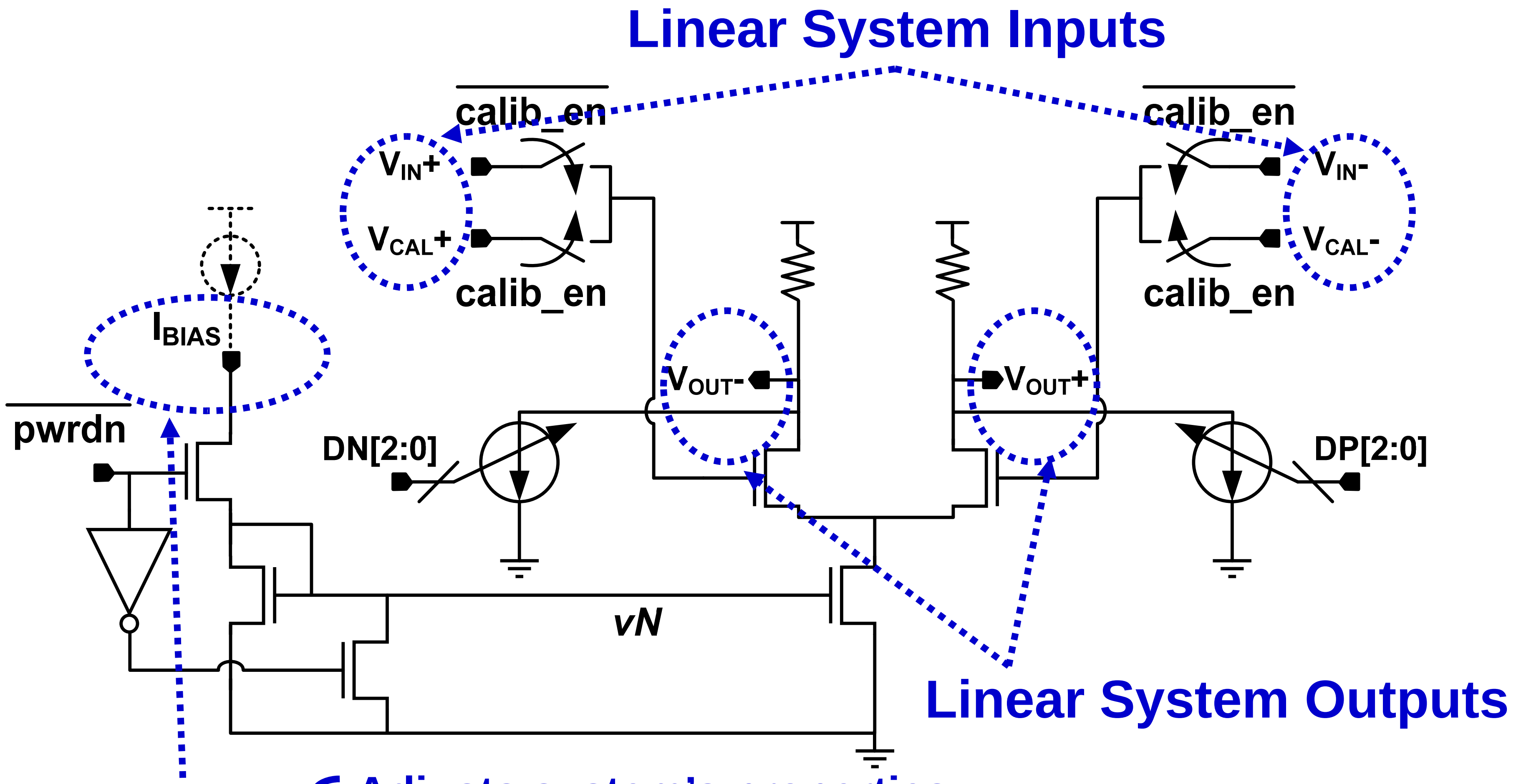

$\overline{\text{calib\_en}}$
V_{IN}+
V_{CAL}+
calib_en
$\overline{\text{calib\_en}}$
V_{IN}-
V_{CAL}-
calib_en
I_{BIAS}
$\overline{\text{pwrdn}}$
I_N
I_P
V_{OUT}-
V_{OUT}+
DN[2:0]
DP[2:0]
vN
DAC
d[0]
d[1]
d[2]
vN
1x
2x
4x

Generating Vectors: Using Port Types

- Analog I/O port
  - I/O of the intended linear system
  - Similar to I/O along the data path in digital systems

- Analog control port
  - Analog control input adjusts the system’s properties
    - Gain, bandwidth, offset, etc.
  - The controlled properties depend on the designer’s intent

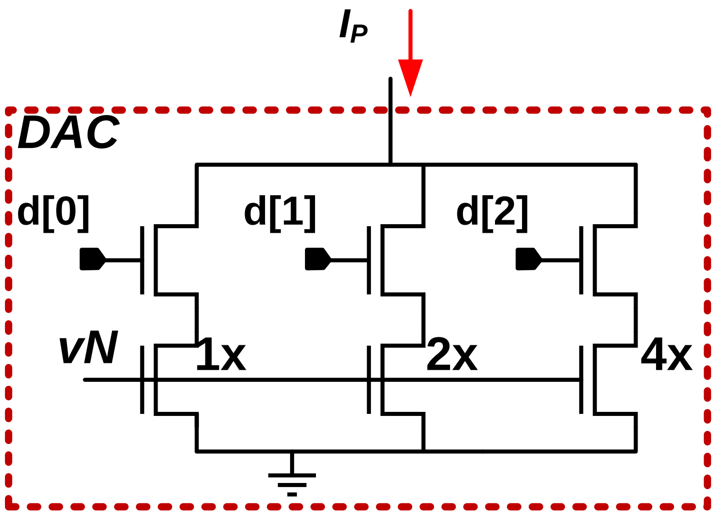
Analog I/O & Control Port: Example



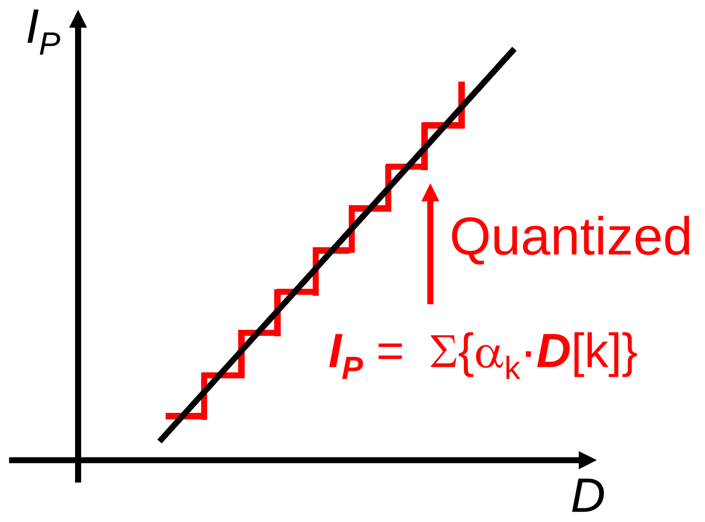
Control Input { Adjusts system's properties
- Gain
- Output Swing
- Bandwidth

Quantized Analog Port

- It adjusts the analog quantity in a quantized step
  - Most digital ports in digitally-assisted analog circuits
- Linearity holds
  - Test each bit independently
  - It's tested independently w/ other analog inputs



< Current D/A converter>



< The response, current vs. digital code>

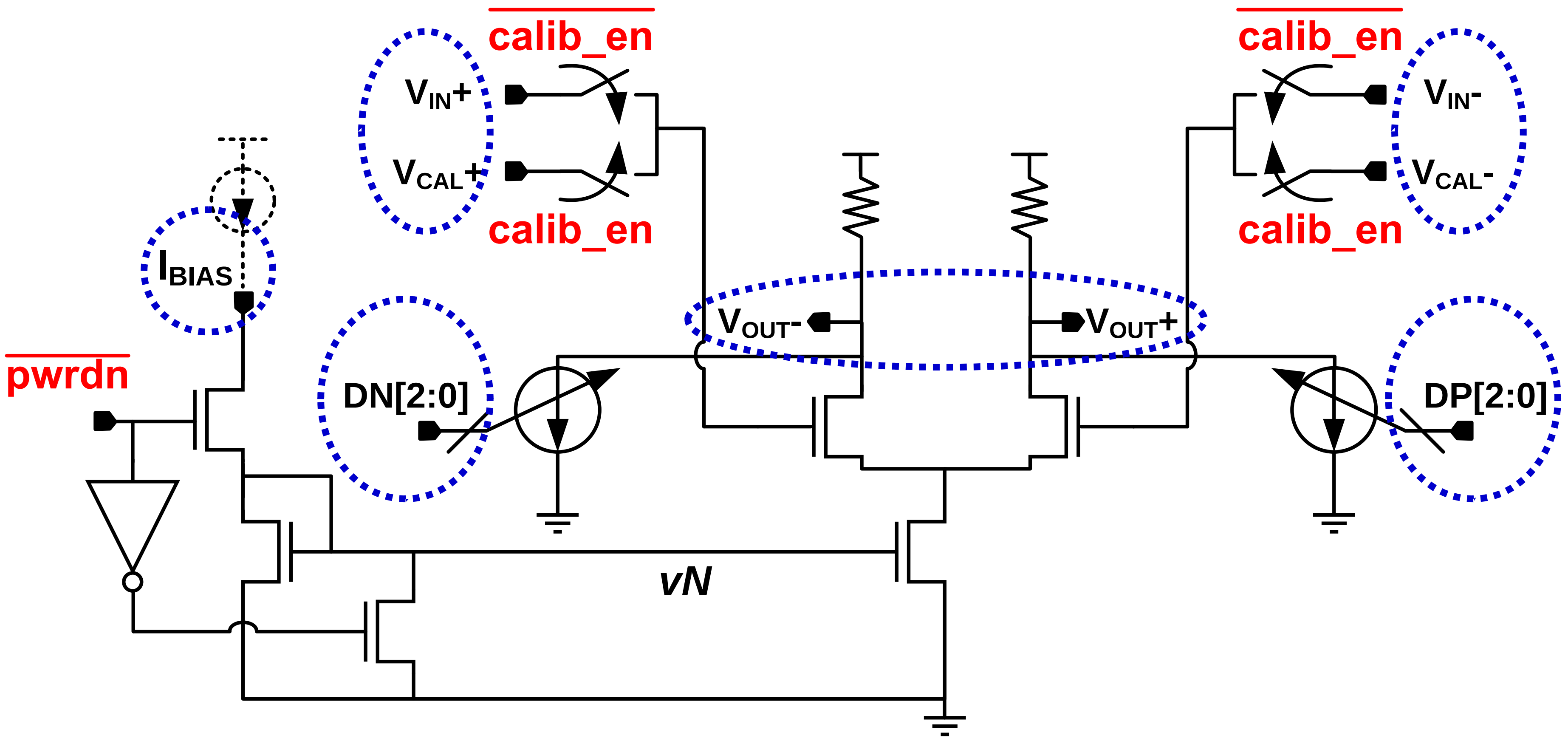
True Digital Port/ Function Port

- True Digital Port
  - It configures different linear systems
    - For M true digital ports, 2^M linear systems
  - It needs to check (quantized) analog ports of each linear system

- Function Port
  - It enables the operation of the circuit
    - It bears no information for the system
  - It is essentially part of the circuit
    - Not really an I/O to the circuit
  - Example:
    - Sequencing clocks in switched-capacitor circuits

True Digital Port: Example

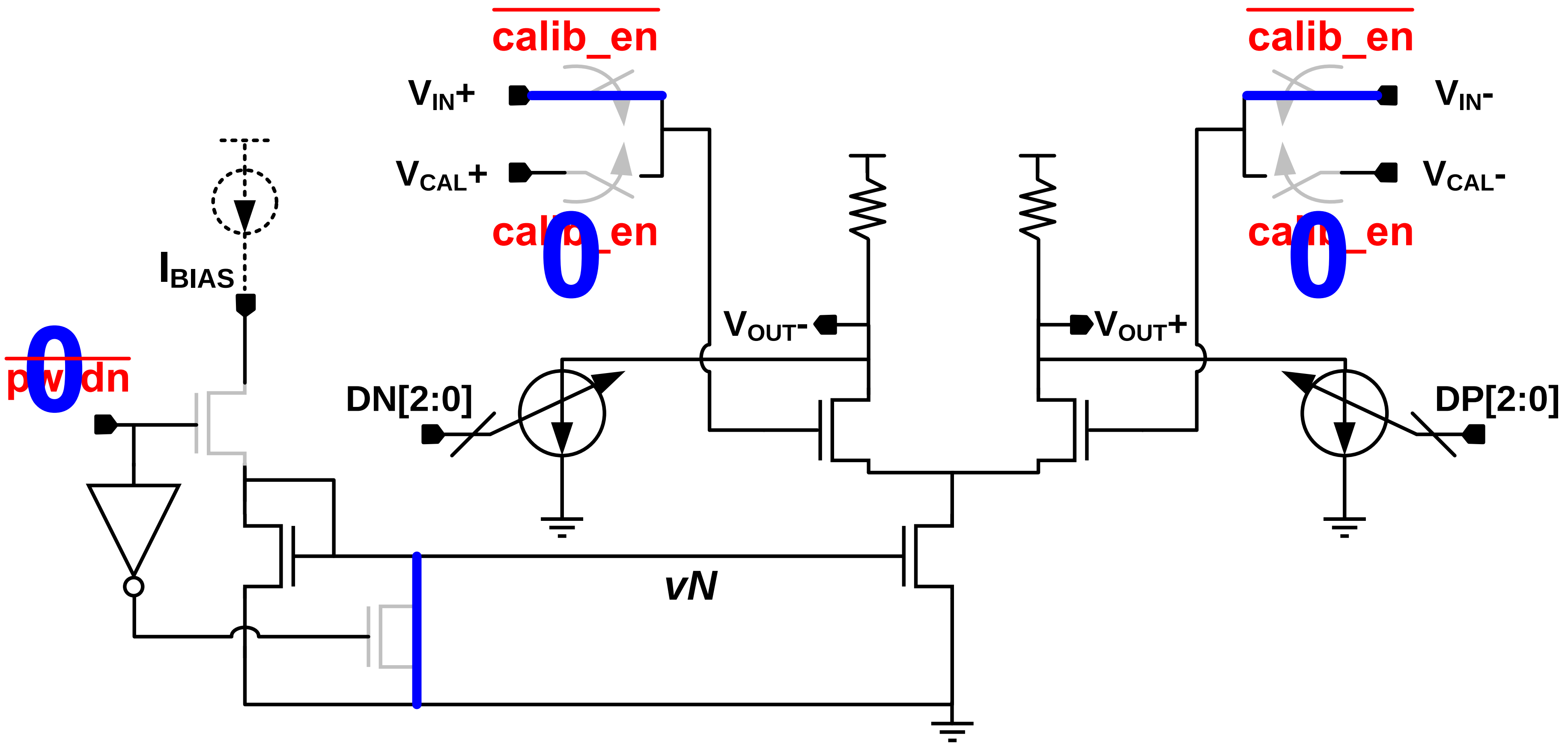
- {calib_en, /pwrdn} creates 2^2 linear systems



: Linear system's analog ports

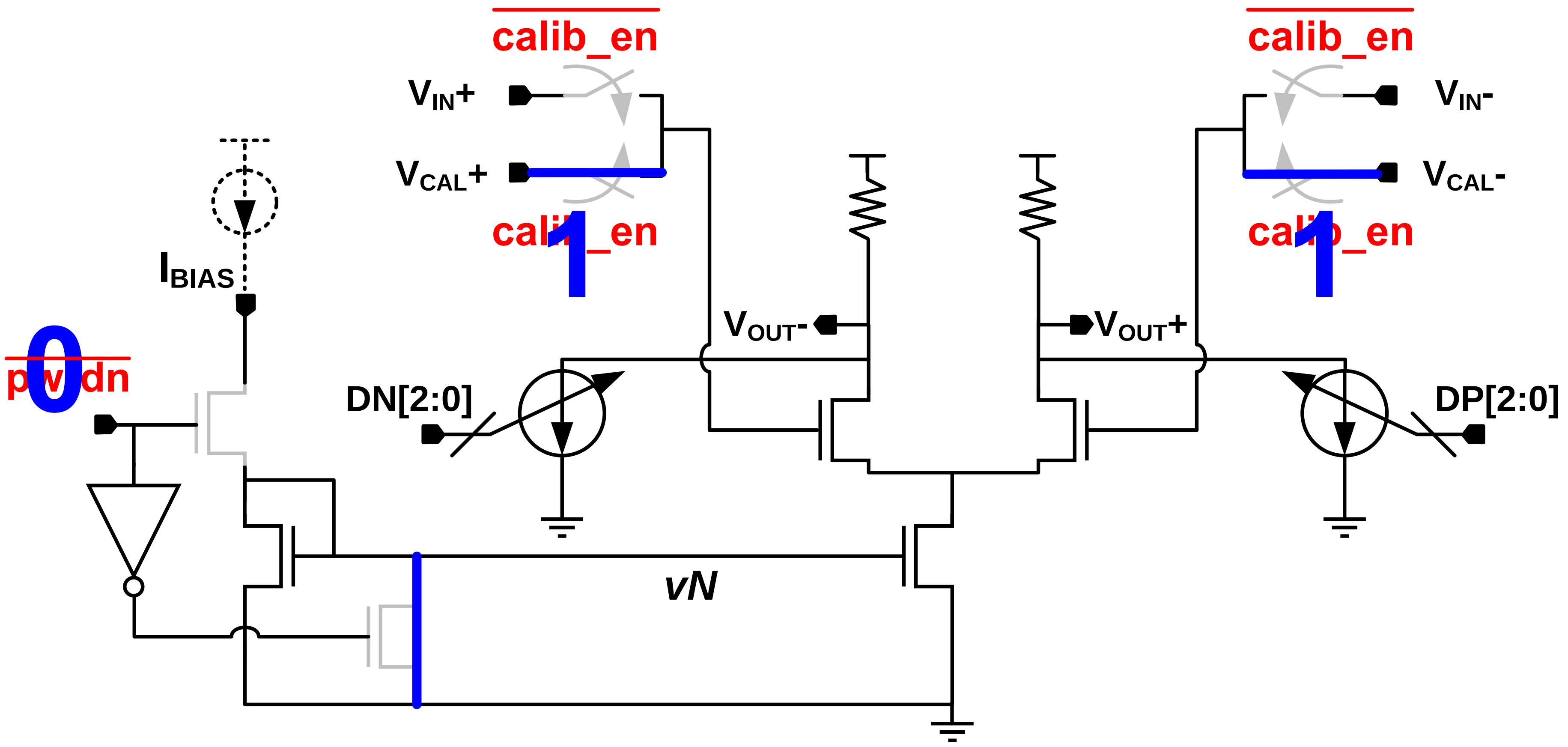
True Digital Port: Example

- {calib_en, /pwrdn} creates 2^2 linear systems



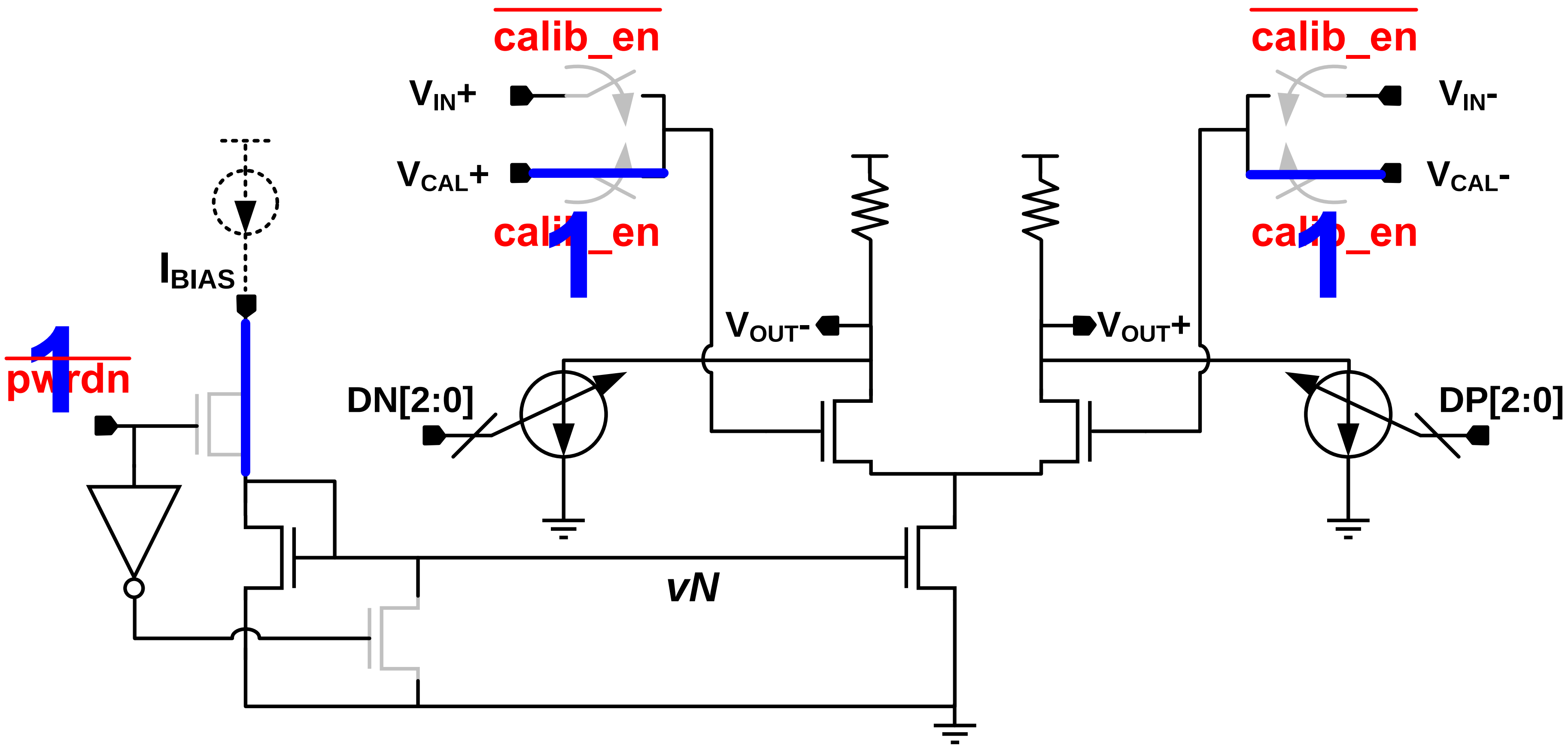
True Digital Port: Example

- {calib_en, /pwrdn} creates 2^2 linear systems



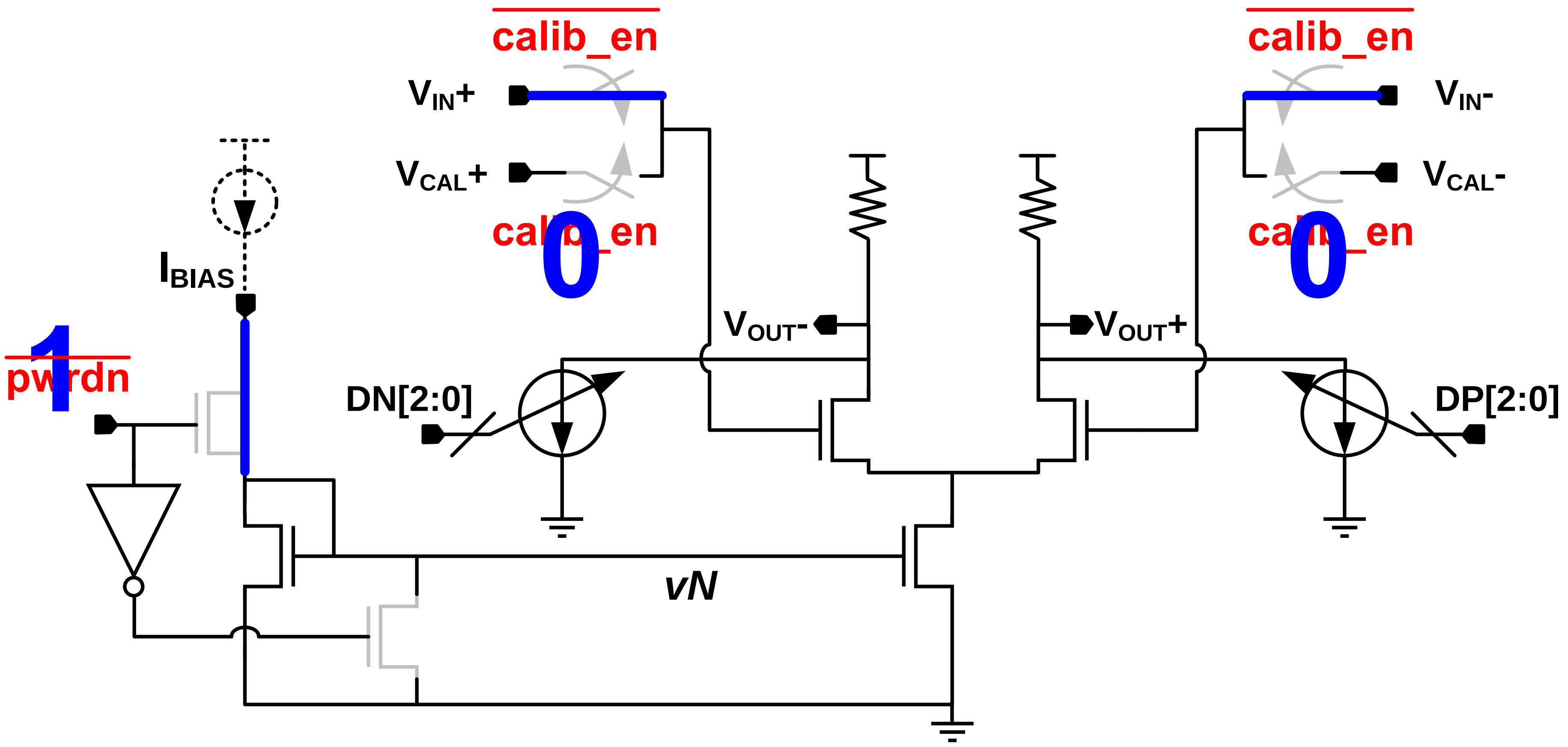
True Digital Port: Example

- {calib_en, /pwrdn} creates 2^2 linear systems



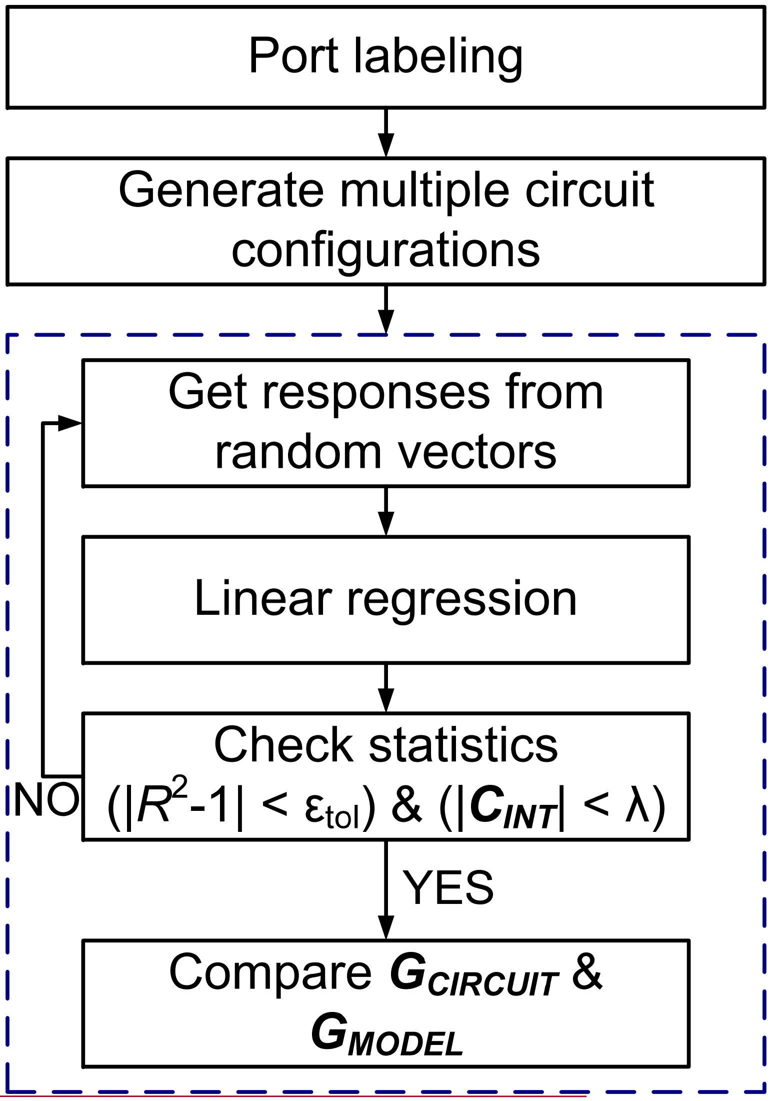
True Digital Port: Example

- {calib_en, /pwrdn} creates 2^2 linear systems



Checking Procedure

- Generate circuits to check
  - True digital inputs cause the linear circuit to change, and each needs to be checked
- Generate input stimulus
  - Using domain converter if needed
- Check to ensure circuit is linear
  - If not complain to user
- Check equivalence
  - Comparing gain matrices



Port labeling
Generate multiple circuit configurations
Get responses from random vectors
Linear regression
Check statistics
$(|R^2-1| < \varepsilon_{tol})$ & $(|\boldsymbol{C}_{INT}| < \lambda)$
NO
YES
Compare $\boldsymbol{G}_{CIRCUIT}$ & $\boldsymbol{G}_{MODEL}$

Size of Analog Blocks

- It will be easier to validate smaller blocks
  - Less inputs/outputs
  - Less true digital inputs

- Digital functional models are unidirectional
  - Can’t easily model tightly coupled systems

- Tear circuit into the smallest unidirectional blocks
  - Need to account for output load in model
  - Easiest method is to extract transfer matrix for each instance
    - Extract when simulated in proper environment

Analog Fault Detection/Coverage

- If a circuit is defined by transfer matrix
  - One can find all faults by measuring that matrix
- Measuring that matrix is not hard
  - Since the number of required inputs is small
  - Even when the matrix is a function of control inputs
- Problem is determining what is a fault
  - Since no two matrices will ever be exactly the same
  - Need to set a tolerance
    - Is it absolute error? Relative error?
  - Unlike digital, generating the stimulus is the easy part.

Conclusions

- Analog circuits are not linear but
  - A linear model is a great abstraction for their operation
- Extensions allow most circuits to be modeled this way
  - Domain transformation
  - Controlled linear system
- This abstraction makes it possible to:
  - Formally define a functional model
  - Formally define fault coverage
- There is no excuse for not using this approach