Digital Analog Design

Mark A. Horowitz, Metha Jeeradit, Frances Lau, Sabrina Liao, ByongChan Lim, James Mao
Electrical Engineering, Stanford University

Jaeha Kim
Seoul National University
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
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
Circuit Browser

Circuits
- Amplifiers and Comparators
  - Clock and Timing
- Data Converters
  - ADC
    - Delta-Sigma
    - Digital Filter
    - Dual-Slope
    - Flash
    - Modulator
    - Pipeline
  - SAR
- DAC
  - Switches and Multiplexers
- Interface
- Power Management
- Reference
- RF/IF

Data Converters > ADC

Electrical Symbol for Analog to Digital Converter (ADC)

Inputs
- ANALOG_INPUT
- ANALOG_REFERENCE (Optional)
- ENCODE_OR_CLK
- Vdd
- Gnd

Outputs
- DIGITAL_DATA_OUTPUT
Circuit Browser

DATA CONVERTERS > ADC > SAR

Schematic  Regressions  Assertions

LINCOLN

Full Power Bandwidth

- Measure FPB

Results

- FPB, MHz

INHERITED

Single-Tone FFT

- Measure SNR
- Measure SNRFS
- Measure SINAD
- Measure noise figure
- Measure noise floor

Results

- SNR, dB
- SNRFS, dBFS
- SINAD, dB
- NF, dB
- NOISE_FLOOR, dBFS
## Circuit Browser

### Local
- Input common mode range ~ 0.2V - 0.8V

### Inherited
- Input termination ~ 50 ohm
## Test Browser

### Patterns

<table>
<thead>
<tr>
<th>Test Browser</th>
<th>Stimulus: Frequency Sweep</th>
<th>Technique: Frequency Domain</th>
<th>Measurement: Frequency Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Analyzer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectrum Analyzer</td>
<td>Technique: Frequency Domain</td>
<td>Measurement: Frequency Domain</td>
<td></td>
</tr>
<tr>
<td>Step Response</td>
<td>Technique: Large Signal</td>
<td>Stimulus: Voltage Step</td>
<td>Measurement: Time Domain</td>
</tr>
<tr>
<td>Virtual Spectrum Analyzer</td>
<td>Technique: Frequency Domain</td>
<td>Measurement: FFT</td>
<td></td>
</tr>
</tbody>
</table>

### Tests

<table>
<thead>
<tr>
<th>Test Description</th>
<th>Application: DAC</th>
<th>Technique: Frequency Domain</th>
<th>Stimulus: Clock</th>
<th>Stimulus: Digital Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Time</td>
<td></td>
<td>Technique: Large Signal</td>
<td>Stimulus: Voltage Step</td>
<td>Measurement: Time Domain</td>
</tr>
<tr>
<td>Adjacent Channel Leakage Ratio</td>
<td>Application</td>
<td>Technique: Frequency Domain</td>
<td>Stimulus: Clock</td>
<td>Stimulus: Digital Pattern</td>
</tr>
<tr>
<td>Analog Input Full-Scale Range</td>
<td>Application: ADC</td>
<td>Technique: Frequency Domain</td>
<td>Stimulus: Clock</td>
<td>Stimulus: Single Tone</td>
</tr>
<tr>
<td></td>
<td>Stimulus: Voltage Sweep</td>
<td>Measurement: FFT</td>
<td>Measurement: RF Voltmeter</td>
<td></td>
</tr>
</tbody>
</table>
## Test Browser

### Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Application</th>
<th>Technique</th>
<th>Stimulus</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Time</td>
<td>Analog Switch</td>
<td>Large Signal</td>
<td>Voltage Step</td>
<td>Time Domain</td>
</tr>
<tr>
<td>Break-Before-Make-Delay</td>
<td>Analog Switch</td>
<td>Large Signal</td>
<td>Voltage Step</td>
<td>Time Domain</td>
</tr>
<tr>
<td>Charge Injection</td>
<td>Analog Switch</td>
<td>Large Signal</td>
<td>Voltage Step</td>
<td>Time Domain</td>
</tr>
<tr>
<td>Full Power Bandwidth</td>
<td>Amplifier</td>
<td>DC</td>
<td>Large Signal</td>
<td>Feedback</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Voltage Step</td>
<td>Time Domain</td>
</tr>
<tr>
<td>Rise Time, Fall Time and</td>
<td>Amplifier</td>
<td>Large Signal</td>
<td>Feedback</td>
<td>Voltage Step</td>
</tr>
<tr>
<td>Overshoot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setting Time</td>
<td>Analog Switch</td>
<td>Large Signal</td>
<td>Voltage Step</td>
<td>Time Domain</td>
</tr>
</tbody>
</table>
## Tags

<table>
<thead>
<tr>
<th>Unigrams</th>
<th>Bigrams</th>
<th>Trigrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique: Large Signal</td>
<td>Measurement: Time Domain</td>
<td>Measurement: Time Domain</td>
</tr>
<tr>
<td>(3)</td>
<td>Stimulus: Voltage Step (8)</td>
<td>Stimulus: Voltage Step (8)</td>
</tr>
<tr>
<td>Measurement: Time Domain</td>
<td>Technique: Large Signal (8)</td>
<td>Technique: Large Signal (3)</td>
</tr>
<tr>
<td>(3)</td>
<td>Stimulus: Voltage Step (8)</td>
<td>Application: Amplifier</td>
</tr>
<tr>
<td>Stimulus: Voltage Step (8)</td>
<td>Technique: Large Signal (8)</td>
<td>Stimulus: Voltage Step (4)</td>
</tr>
<tr>
<td>Application: Analog Switch</td>
<td>Technique: Feedback (4)</td>
<td>Application: Amplifier (4)</td>
</tr>
<tr>
<td>(4)</td>
<td>Technique: Large Signal (4)</td>
<td>Measurement: Time Domain (4)</td>
</tr>
<tr>
<td>Technique: Feedback (4)</td>
<td>Stimulus: Voltage Step (4)</td>
<td>Technique: Large Signal (4)</td>
</tr>
<tr>
<td>Technique: DC (1)</td>
<td>Technique: Feedback (4)</td>
<td>Technique: Feedback (4)</td>
</tr>
<tr>
<td></td>
<td>Application: Amplifier (4)</td>
<td>Technique: Large Signal (4)</td>
</tr>
<tr>
<td></td>
<td>Stimulus: Voltage Step (4)</td>
<td>Stimulus: Voltage Step (4)</td>
</tr>
<tr>
<td></td>
<td>Technique: Feedback (4)</td>
<td>Technique: Feedback (4)</td>
</tr>
<tr>
<td></td>
<td>Application: Amplifier (4)</td>
<td>Application: Amplifier (4)</td>
</tr>
</tbody>
</table>
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 (Linear)

Digital (Binary)
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

\[ CLK_o = f(CLK_i, V_{ctrl}) \]
Variable Domain Translation

- **Duty-Cycle Adjuster**

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

\[
Duty(CLKo) = \alpha \cdot Duty(CLKi) + \beta \cdot V(Vctrl)
\]
Result Surface

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

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 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

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?

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’b1111;
      4’b1101: out = 4’b1110;
      4’b1110: out = 4’b1010;
      4’b1111: out = 4’b1110;
    endcase
  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
What Constitutes a Good Model?

- Simple, but could capture realistic behavior if necessary
  - Easy to update
  - Intuitive breakdown of analog behavior
- Fast to simulate
- Pin-accurate
What Not To Do

- Write a bad circuit simulator in Verilog:

```verilog
module pll_if (  
    input up, dn,       // inputs from phase detector  
    `output_real Vctrl // control voltage output  
);  
  timemun 1ps;  
timeprecision 1ps;

  `parameter_real(Icp,1e-3); // nominal charge pump current  
  `parameter_real(Cp, 8e-12); // loop capacitor (proportional)

...  
always begin  
    #1;              // 1ps step  
    Vctrl = (Cp*Vctrl + Icp*1e-12*(up-dn)) / Cp;  
end  
...  
endmodule  
```

- Time integration i.e., long simulation time
Example Model #1: Interpolator

How do we get it to be “linear”?
- Large signal in, large signal out
- Lots of (non-linear) digital controls
- … but for non-extreme digital control values, output delay from input should be roughly linear
Example Model #1: Interpolator

- Model for interpolator
  - Delay = f (input phase difference, digital control bits, supply)
  - f is conjectured to be a polynomial of the arguments
    (coefficients come from LS fit of simulation data)
    - Create different model levels

- Simulation space for data extraction is too large
  - At least for non-linear models
    - Have many possible inputs (cross-product is large)
  - Simulate carefully the critical region
  - Assume model is separable for extrapolation to entire space
Example Model #1: Interpolator

- Form of the delay equation:

\[
\text{delay} = \sum \alpha_i (\omega t)^i (\omega n)^m (c a p)^n (d)^k
\]

such that \(0 \leq i + m + n + k \leq 3\)

- Jitter
  - roughly 1% change in Vdd causes 1% change in delay

- Final equation:

\[
\text{total delay} = \text{delay} - \left(\frac{\Delta V_{dd}}{V_{dd}}\right)\text{delay}
\]

- Model returns extrapolated output
  - Raise a flag when inputs are outside carefully simulated region
Example Model #2: Comparator

- Demonstration of different levels of model accuracy

Offset adjustment ($\text{cfg\_offset [m:0]}$)

Small signal behavior modeled as discrete time
filter response with choice of:
1). Ideal comparator – gain only
2). First order system – gain + 1 dominant pole
3). Full response – estimated from circuit step response

- Offset = $f(\text{cfg\_offset})$
  modeled with linear, quadratic and cubic fit, as well as lookup table for maximum accuracy
1). Ideal comparator

2). 1st order AC response

3). Full AC response
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
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

Linear System Inputs

- $V_{IN^+}$
- $V_{CAL^+}$
- $V_{CAL^-}$
- $V_{OUT^+}$
- $V_{OUT^-}$

Control Adjusts system’s properties:
- Gain
- Output Swing
- Bandwidth

Linear System Outputs

pwrdsn

DN[2:0]

I_BIAS

$\text{vN}$

DP[2:0]
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

\[ I_P = \sum \alpha_k \cdot D[k] \]

< 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

- \{\text{calib}\_\text{en}, \text{/pwrdn}\} creates \(2^2\) linear systems
True Digital Port: Example

- \{\text{calib\_en}, /\text{pwr}dn\} creates $2^2$ linear systems
True Digital Port: Example

- `{calib_en, /pwrdn}` creates $2^2$ linear systems
True Digital Port: Example

- `{calib_en, /pwrdsn}` creates $2^2$ linear systems
True Digital Port: Example

- \{\text{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

Diagram:

1. Port labeling
2. Generate multiple circuit configurations
3. Get responses from random vectors
4. Linear regression
5. Check statistics: \(|R^2-1| < \varepsilon_{\text{tol}}\) & \(|C_{\text{INT}}| < \lambda\)
6. Compare \(G_{\text{CIRCUIT}}\) & \(G_{\text{MODEL}}\)
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