



# Design of a Switch-Level Analog model for Verilog

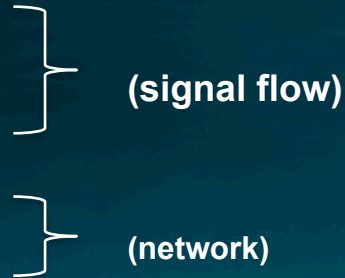
September 25, 2008

Tom Sheffler  
Rambus Inc.

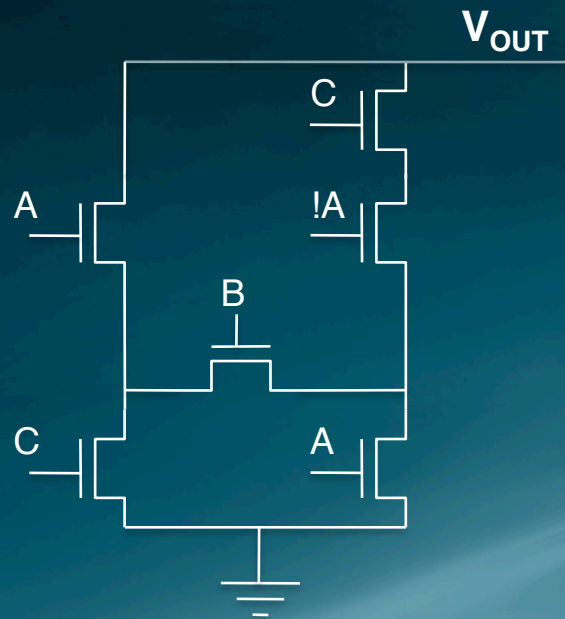
**Rambus**<sup>®</sup>

# Verilog Review

- **A variety of logic evaluation mechanisms**
  - **Gates, UDPs**
  - **Behavioral**
  - **Switch-Level**
- **State**
  - **Stored in registers**
- **Wires/Nets**
  - **Resolve values (signal flow)**
  - **Connect nodes (network) / Store Charge (triereg)**



# A switch-level logic network



# Previous Analog extensions

- **Previous Analog event-driven Extensions**
  - “analog wires”
  - “real number” modeling (wreal)
- **Both are signal-flow**
- ***Coercing Kirchoff networks into signal-flow can introduce errors***

# Idea: Extend Switch-Level

- **Value Domain**
  - Logic => Reals
- **Components**
  - Switches => Pseudo-Linear (2-terminal) devices
- **Time Domain**
  - Discrete time: Verilog Event queue
- **Keep fundamental notions of switch-level**
  - Subnetwork-based
  - Relaxation algorithm => solve linear system
  - X-values

# Talk Roadmap

- Explain modeling components and simulation algorithm.
- Explain a mixed-signal subsystem.
- Show a way to model subsystem, suitable for detecting D/A boundary errors.
- Discuss implementation and performance.

# Register Controlled Resistor



```
wire w1, w2;  
real rval;  
  
initial begin  
    rval = 100; // ohms  
    $resistor(w1, w2, rval);  
end
```

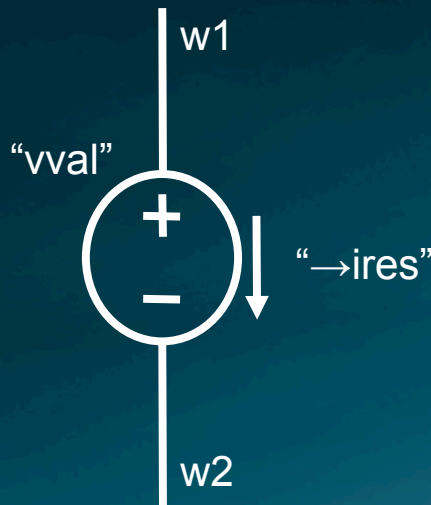
- Register defines magnitude
- Wires name nodes

# Register-Controlled Current Source



```
wire w1, w2;  
real cval;  
  
initial begin  
    cval = 0.1;  
    $csrc(w1, w2, cval);  
end
```

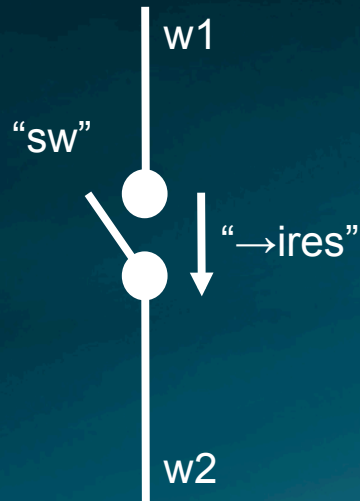
# Register-Controlled Voltage Source



```
wire w1, w2;  
real vval;  
real ires;  
  
initial begin  
    vval = 2.0;  
    $vsrc(w1, w2, vval, ires);  
end
```

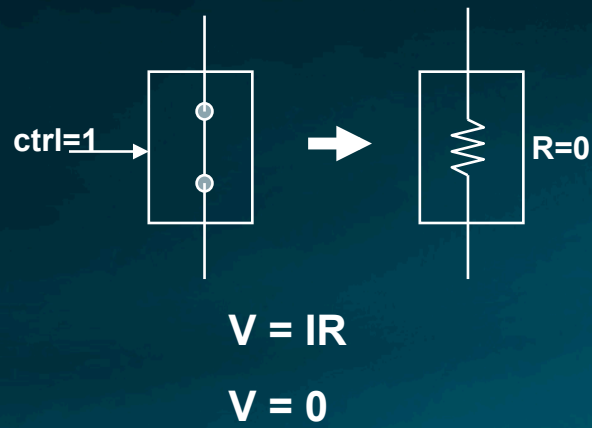
- Register defines magnitude
- Register accepts output

# Register-Controlled Switch

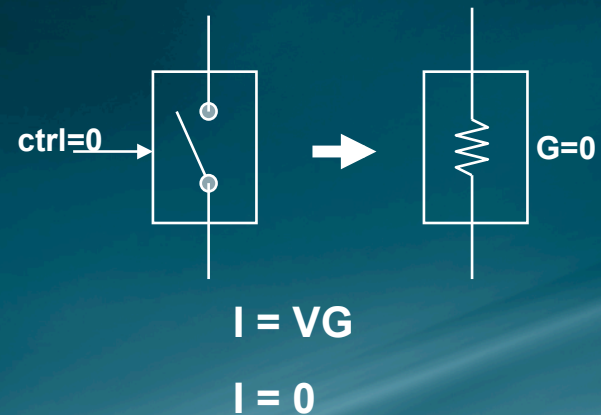


```
wire w1, w2;  
integer sw  
real ires;  
  
initial begin  
    sw = 0; // off  
    $switch(w1, w2, sw, ires);  
end
```

# Modeling a Switch

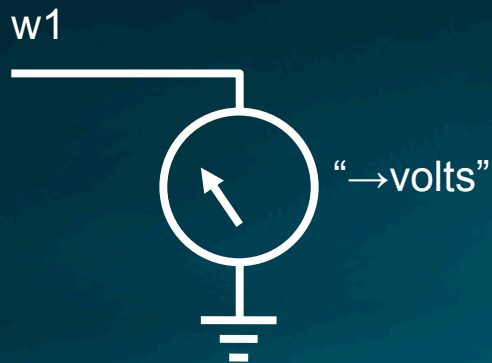


- A constraint on voltage, no constraint on  $I$



- A constraint on current, no constraint on  $V$

# Voltage into a Register



```
wire w1;  
real volts;  
  
initial begin  
    $vprobe(w1, volts);  
end
```

# Simulation Algorithm

- **Start-of-Simulation**

- **Resolve wire names referenced to “node” objects.**
  - Use VPI to traverse database hierarchy.
- **Build subnetworks.**
  - Connected-components of electrical elements.
- **Build matrices using modified nodal analysis.**

- **On Value Change**

- **Update matrix of subnetwork affected.**
- **Schedule matrix solve and new output values of subnetwork at end-of-timestep.**

# A note about explicit Subnetworks

- Gives useful event semantics.
- Potential for “fast” simulation.
  - Solution of each subnetwork should be fast.
  - Good for multirate, multiphase designs.
  - Subnetwork evaluation comparable to UDP?
- Basis for X values and X propagation
  - Pessimistic: If any inputs to a subnetwork are X, then outputs are X.
  - Less pessimistic algorithms may be useful, too.



# A Verification Example

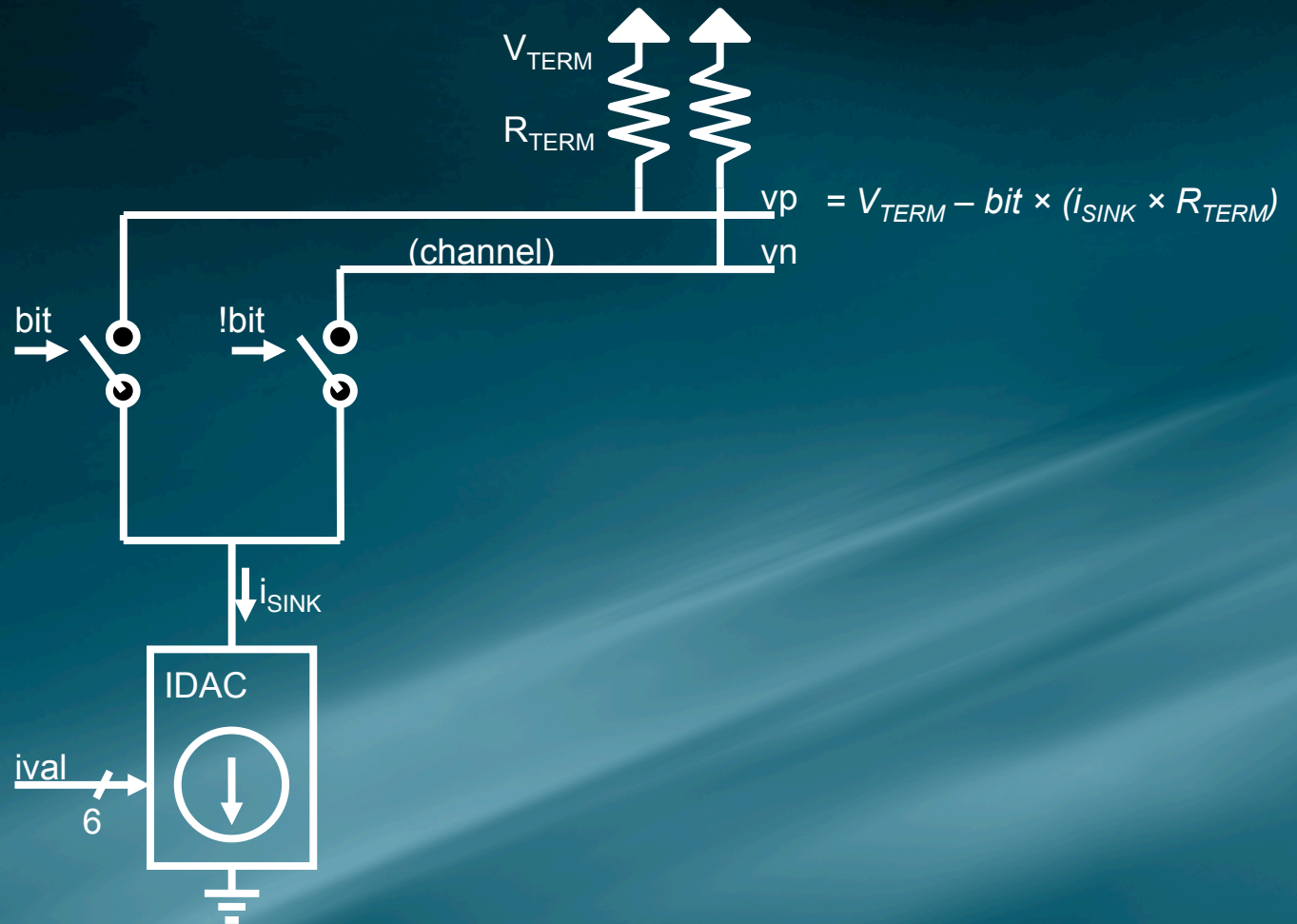
# The Problem

- **Equalization:**
  - implements equation
$$\text{out}_i = \alpha_0 x_i + \alpha_1 x_{i-1} + \alpha_2 x_{i-2}$$
  - $\alpha_0, \alpha_1, \alpha_2$  – programmed by registers (real values)
  - $x_i$  – data bit to be sent this cycle (0 or 1)
  - addition implemented by current summing
- **Verification Question:** can we check the implementation of this equation in the context of its system?

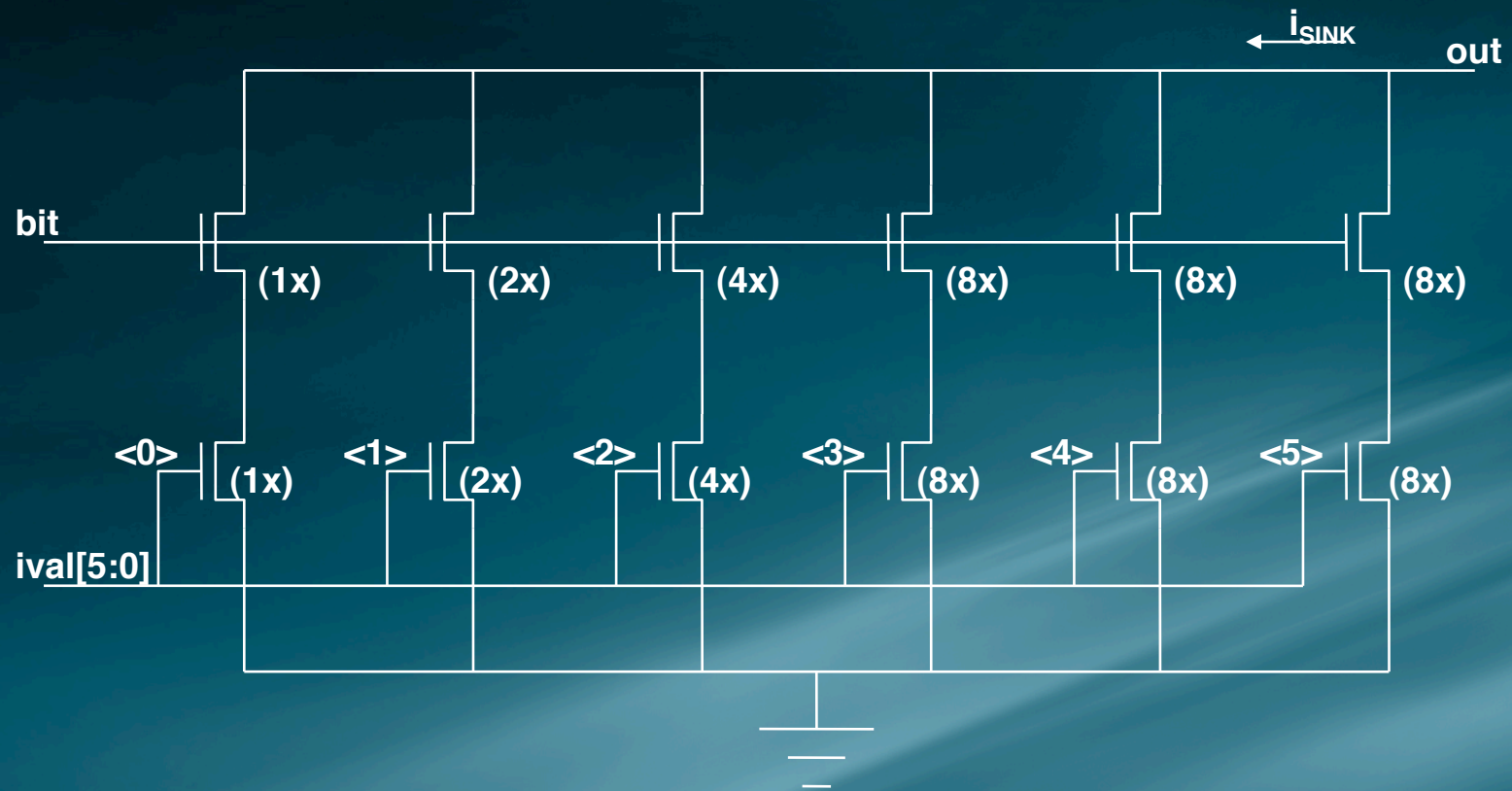
## Errors we are looking for

- Are the  $\alpha$  registers connected up to their associated circuits correctly? (bus swaps, bit inversions)
- Is the map from register value to  $\alpha$  value as intended? (encoding/decoding mismatch, control bit to Xstr leg mapping)
- What is the reset value of these components? [not examined in this presentation]
- Are the interconnections of digital and analog components as intended? (connectivity to device pins)

# Differential Output Driver

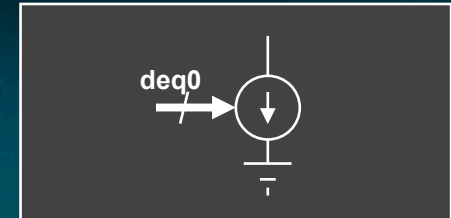


# IDAC Schematic



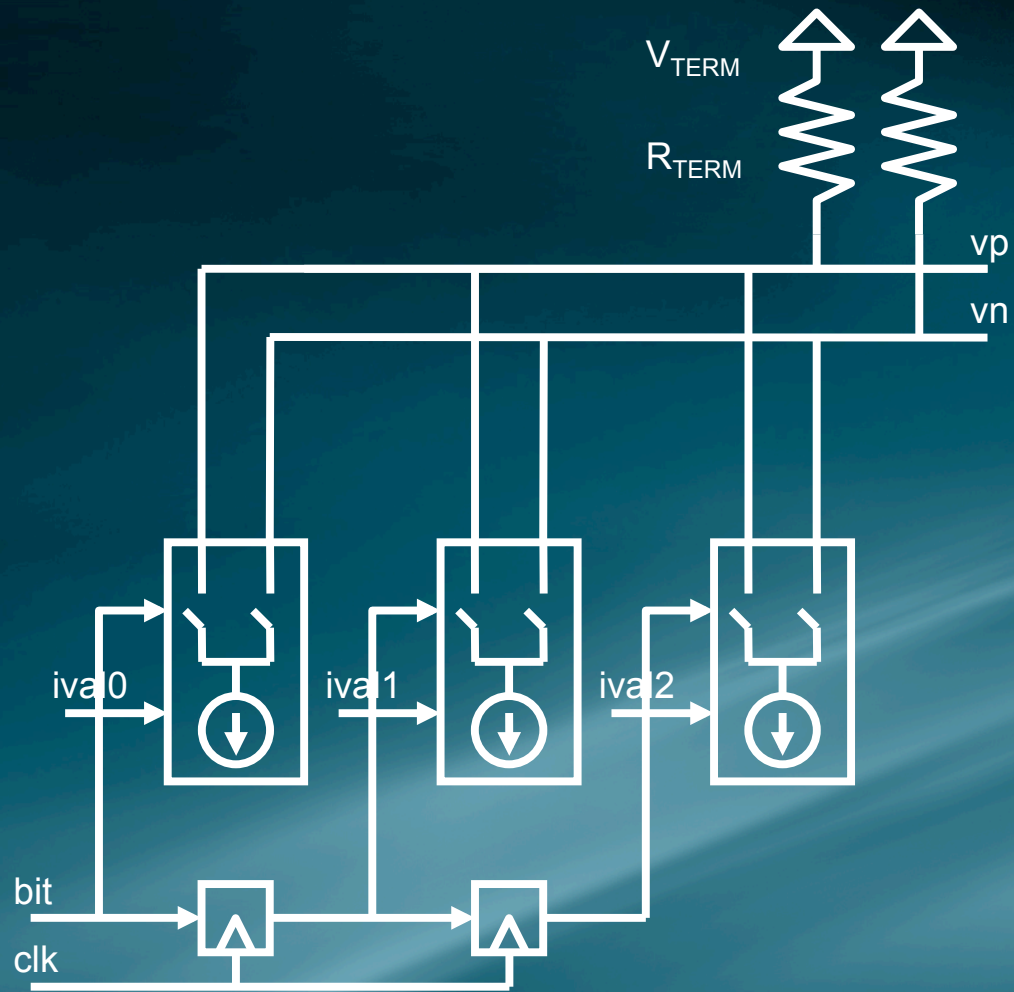
# Behavioral Verilog modeling the iDAC

```
module idac0 (t1, code);  
  
    inout t1;  
    input [3:0] code;  
  
    real ival; initial ival=0.0;  
    integer    status;  
  
    initial  
        $csrc(t1, top.GND, ival);  
  
    always @(code) begin  
        case (code)  
            4'b0000: ival = 0.01;  
            4'b0001: ival = 0.02;  
            4'b0010: ival = 0.03;  
            4'b0011: ival = 0.04;  
            ...  
            ...  
            4'b1110: ival = 0.15;  
            4'b1111: ival = 0.16;  
        endcase  
    end  
  
endmodule // idac0
```



- induce a current with value “ival”
- Designer’s record of his understanding of decoding implemented by the analog block.
- A place to record
  - ones complement
  - twos complement
  - thermometer bits
  - illegal encodings

# Three-Stage Output Driver with Equalization



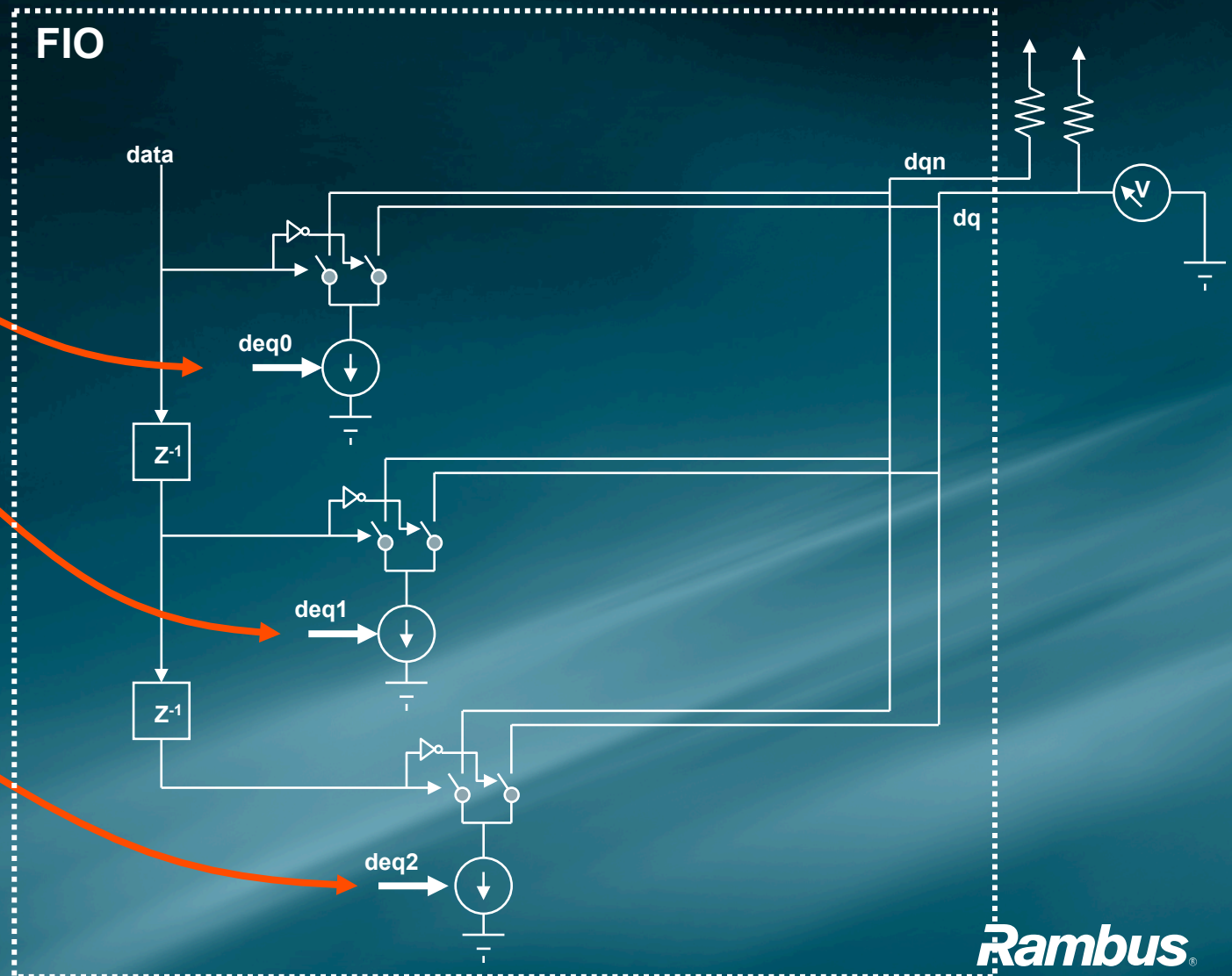
# Test Setup for each Pin

Figure 129 DQ\_TX\_EQ0 Register

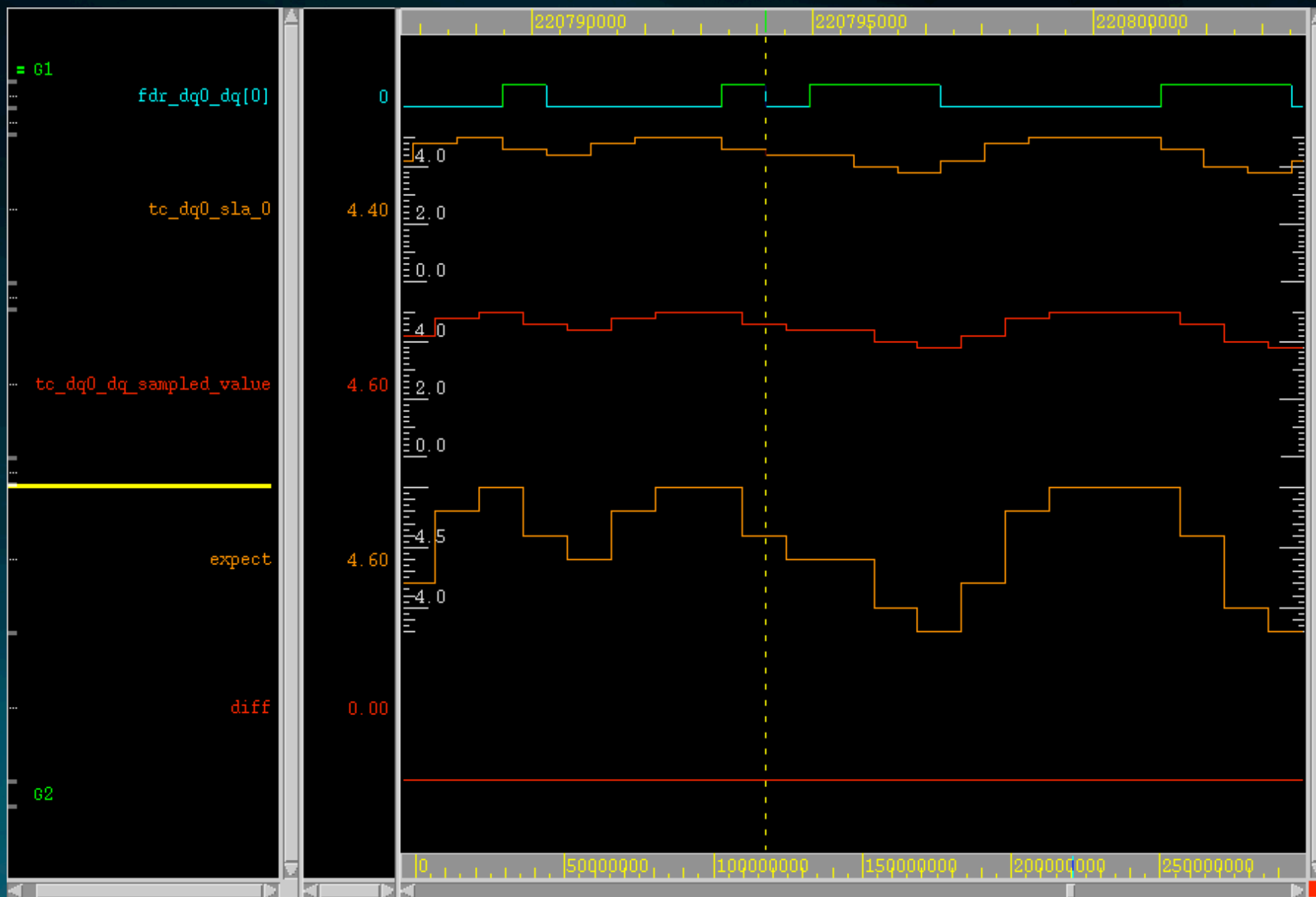
Addr: DQb-1-DQp-1100		RW	Res
0			
1	DEQ0	RW	TE
2	[3:0]		
3			
4	DSIGNOE	RW	Q
5			
6	DEQ1	RW	TE
7	[3:0]		
8			

Figure 130 DQ\_TX\_EQ1 Register

Addr: DQb-1-DQp-1101		RW	Res
0			
1	DEQ2	RW	TE
2	[3:0]		
3			



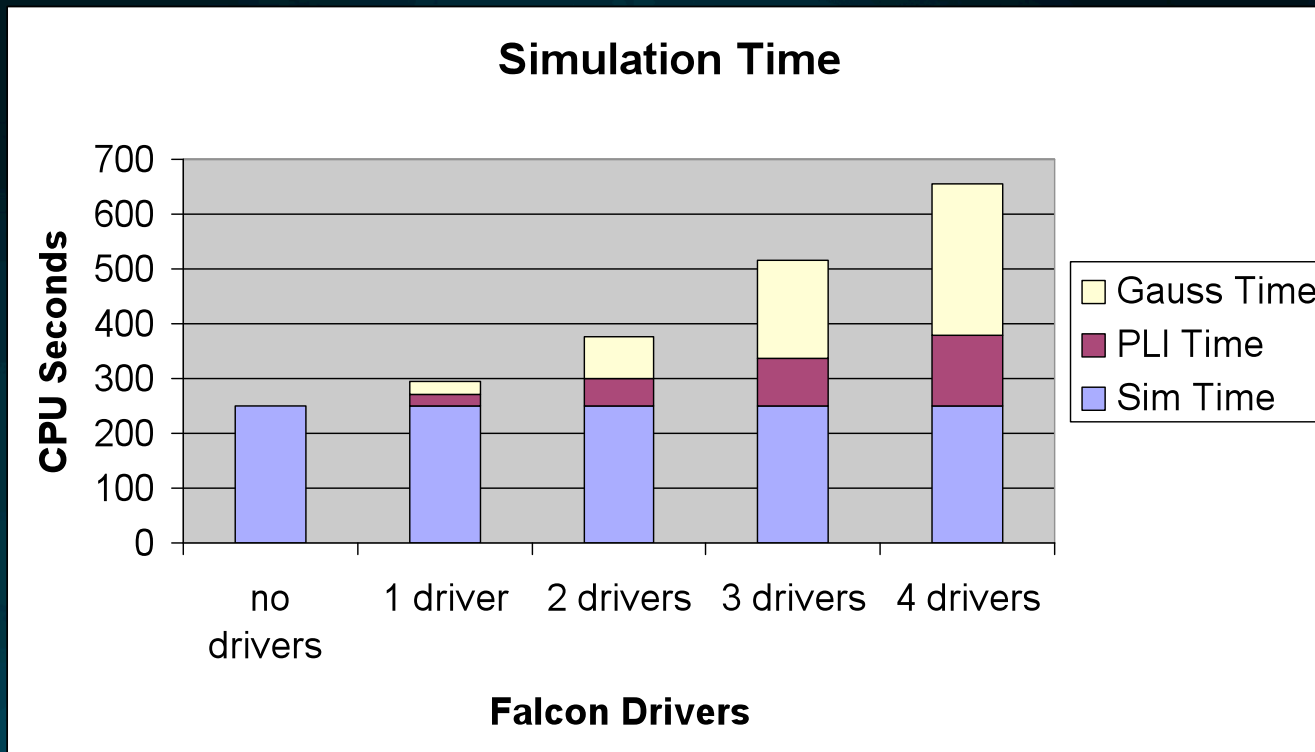
# Verification of RTL w/ 3-stage Equalizer



# Run Times

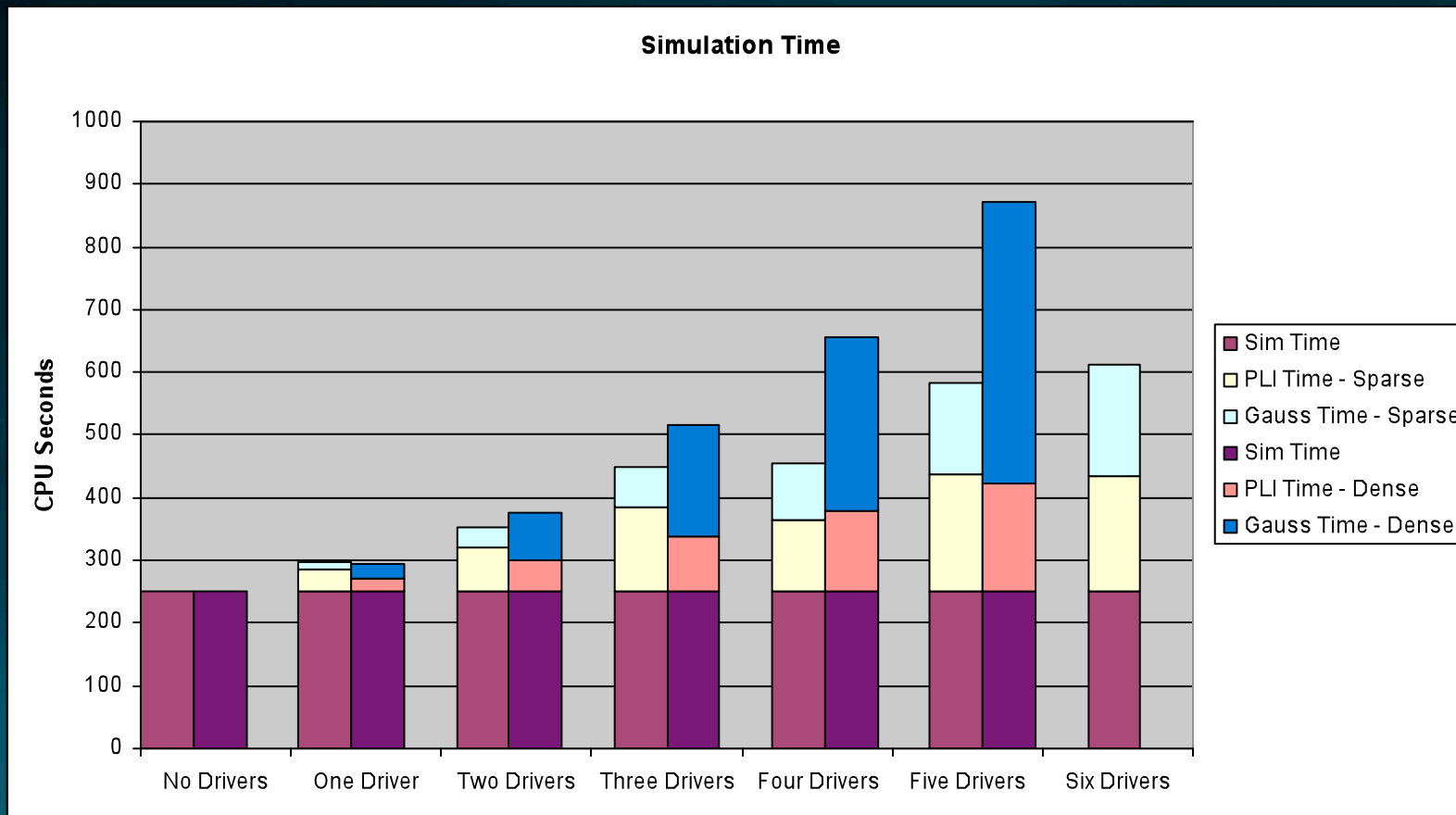
- **Test: ber\_short1**
  - **user: 251 secs**
  
- **With “analog”**
  - **36 three-stage output drivers**
    - **432 devices, 182 nets, 36 subnetworks**
  - **32,000 data transitions**
  - **1.3M 13x13 matrix solves**
  - **9.9M switch toggles**
  
  - **user: 664 secs (Gauss: 254 secs)**

# Where is the time spent?



	no drivers	1 driver	2 drivers	3 drivers	4 drivers
devices	0	216	345	432	540
order	0	7 (x36)	10 (x36)	13 (x36)	16 (x36)
switch toggles	0	1.1M	2.2M	3.4M	4.7M
solves	0	573K	852K	998K	1.0M

# Comparing Dense and Sparse



# Summary

- **Have shown an extension to Verilog useful for**
  - **output drivers**
  - **regulated supplies**
  - **bias networks**
- **System level modeling and verification requires functional abstraction of analog blocks.**
  - **Compatible with logic designers, verifiers.**
  - **Vision: automate the functional abstraction step.**
- **Functional abstraction of other types of analog blocks?**



# Backup

# Modeling

- **Basic Model Requirement**
  - Analog output values reflect digital control.
  - Analog value can be checked.
- **Event-Driven**
  - Transition phenomena are of little interest when checking steady-state values.
- **Building blocks for digitally controlled electrical networks.**

# Behavioral Verilog modeling driver

```
module driver0 (dq, dqn, tdata, code);
  inout dq;
  inout dqn;

  input tdata;
  input [3:0] code;

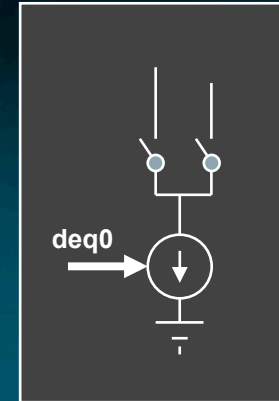
  wire      node;          // internal node
  integer   dqsw; initial dqsw = 0;
  integer   dqns; initial dqns = 1;

  initial begin
    // Instantiate the switches (procedurally)
    $switch(dq, node, dqsw);
    $switch(dqn, node, dqns);
  end

  // Instantiate the DAC (declaratively)
  idac0 i_idac0 (node, code);

  // Connect 'tdata' to the switch
  always @(tdata)
    dqsw = tdata;

  // A behavioral 'inverter'
  always @(dqsw)
    dqns = !dqsw;
endmodule
```



# Instantiate three drivers

```
driver0 i_driver0 (dq, dqn, tx_ena_io ? tdata_out : 1'b0, tx_eq[3:0]);  
driver0 i_driver1 (dq, dqn, tx_ena_io ? tdata1 : 1'b0, tx_eq[8:5]);  
driver0 i_driver2 (dq, dqn, tx_ena_io ? tdata2 : 1'b0, tx_eq[19:16]);
```



- Implementation of register map is tied to observable “analog” output values

# The checker

```
real tc_dq0_sla_0;

initial
  $vprobe(tc_dq0_dq[0], tc_dq0_sla_0);

always @(negedge top.i_falcon_clkgen.dq_sample_clock) begin

  // sample the current real value
  x0 = tc_dq0_sla_0;

  // collect up history of the bit values
  b2 = b1;
  b1 = b0;
  b0 = tc_dq0_dq[0];

  // compute the expect - fixed coefficient values here
  expect = 5.0 - ((b0 * 0.4) + (b1 * 0.6) + (b2 * 0.2));
  diff = expect - x0;

end
```

# Error Detect – Bus Inversion to tap 0

## tx\_eq[3:0] vs tx\_eq[0:3]

