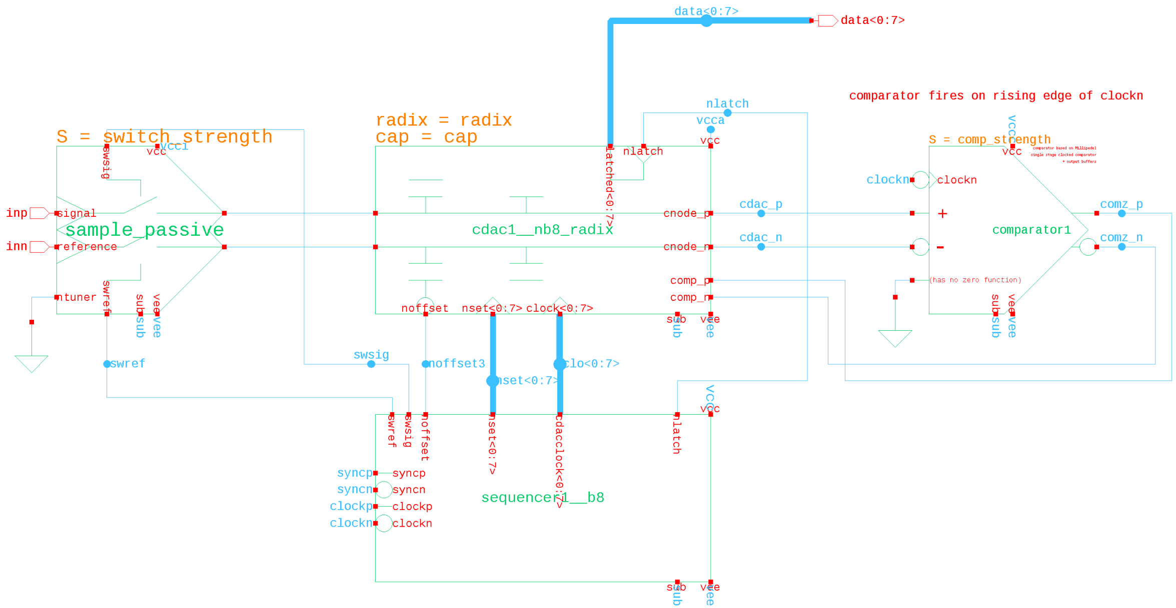
What’s the goal?

Design	EDET DCD	CoRDIA	pre-Helena	Helena
ADC resolution	8-bit	10-bit	8-bit	10-bit
Conversion rate	10 MHz	2.5MHz	5 MHz	10 MHz
Area of one ADC	100x200 µm²	80x330 µm²	60x800 µm²	20x100 µm²
Power of one ADC	1800 µW	30 µW	700 µW	100 µW
FOM_csa (conv/sec/area)	500 Hz/µm²	95 Hz/µm²	105 Hz/µm²	5000 Hz/µm²
FOM_epc (energy/conv)	180 pJ	12 pJ	155 pJ	10 pJ
FOM_ppa (power/area)	9.0 W/cm²	0.11 W/cm²	1.45 W/cm²	5.0 W/cm²
ADC qty Mpix @ 100 KHz	10000	40000	20000	10000
ADCs total pixel rate	100 Gpx/s	100 Gpx/s	100 Gpx/s	100 Gpx/s
ADCs total data rate	800 Gb/s	1 Tb/s	800 Gb/s	1 Tb/s
ADCs total area	2.0 cm²	10.5 cm²	9.6 cm²	0.2 cm²
ADCs total power	35.0 W	1.2 W	14 W	1.0 W



What should we try tuning?

- capacitor array radix ratio & repetitions
- capacitor array unit & repetitions total values
- capacitor array switching scheme? (monotonic is simple)
- comparator architecture and strength
- total ADC nominal resolution
- total comparisons & time per comparison

SPICE @ 100 samples per bin:

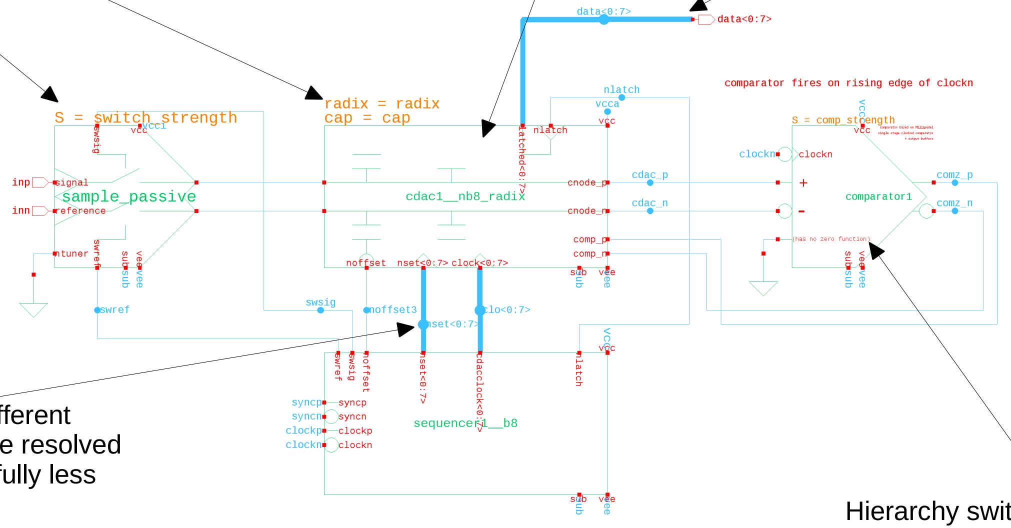
8-bit ADC: ~15 hours
10-bit ADC: ~2.5 days

Schematic setup: Principles

RDAC omitted for now, for simplicity

Output code interpretation done in Python

Several parameters are be passed to each block, from Python testbench



Parameters which mandate different topologies (e.g. bus widths) are resolved by different schematics. Hopefully less than ~20 configurations

Hierarchy switch list used to enable Verilog-A ideal versions of blocks (currently just have comparator)

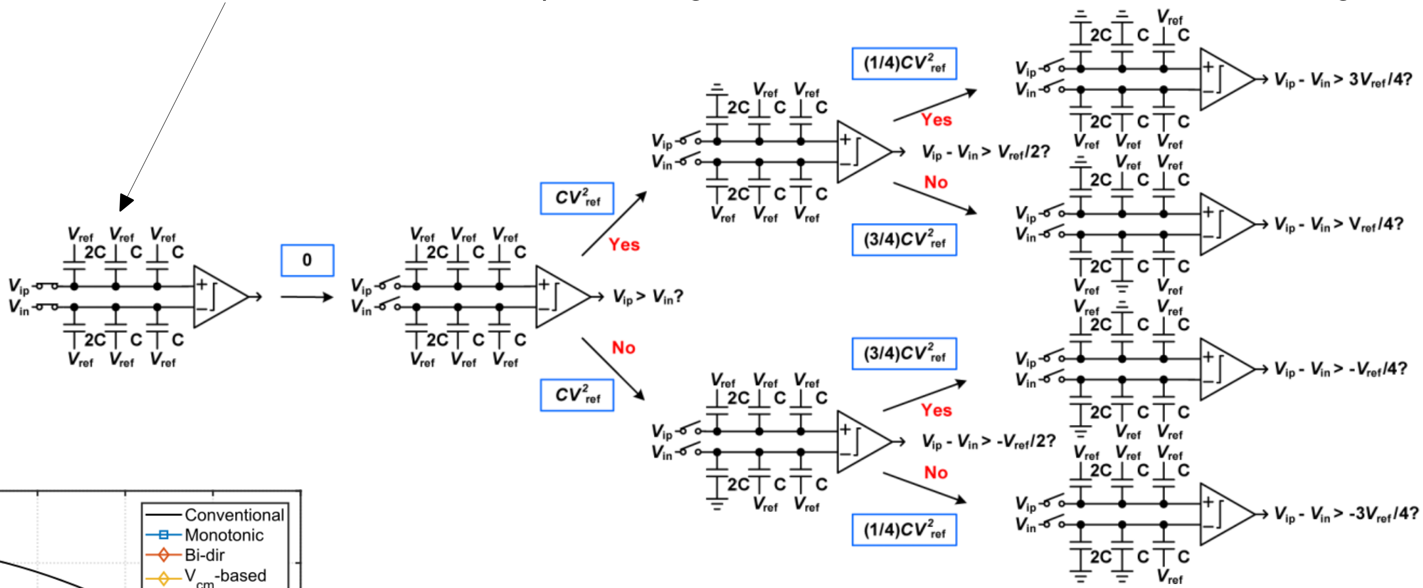
Model parameters

Work in progress

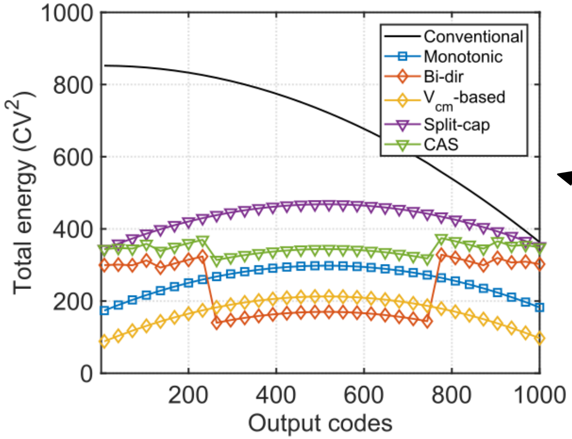
```
params = {
    "ADC": {
        "bit_size": 8,                      # nominal resolution of the ADC (switching between netlists)
        "sampling_frequency": 10.0e6,       # sampling rate in Hz, used to driver clock sources
        "jitter": 0.0e-12,                  # aperture jitter in seconds (TBD)
        "device_noise": False,              # enables basic gaussian noise in behavioral, and tran noise in SPICE
    },
    "TESTBENCH": {
        "positive_input_voltages": [0.2, 1.2, 20e-6],      # start, end (incl.), and step voltage
        "negative_input_voltages": [1.2, 0.2, 20e-6],
        "use_calibration": False,          # account for cap error when calculating Dout (re-analog)
        "pdk_file": "\"~/helena/tech/tsmc65/default_testbench_header_55ulp_linux.lib\" tt",
        "spicedir": None,                  # Use this to write netlist from template
        "rawdir": None,                    # Use this to set SPICE output dir, and to read for parsing
    },
    "SWITCH": {
        "offset_voltage": 0.0e-3,          # offset voltage in Volts
        "common_mode_dependent_offset_gain": 0.0,  # common mode voltage gain
        "threshold_voltage_noise": True,
        "type": "passive",                 # supports active, passive, or ideal
        "strength": 4,
    },
    "COMP": {
        "offset_voltage": 0.0e-3,          # offset voltage in Volts
        "common_mode_dependent_offset_gain": 0.0,  # common mode voltage gain
        "threshold_voltage_noise": True,
        "strength": 4,                     # used to size some active devices (SPICE only)
    },
    "CDAC": {
        "positive_reference_voltage": 1.2,   # reference voltage in Volts
        "negative_reference_voltage": 0.0,   # reference voltage in Volts
        "reference_voltage_noise": 0.0e-3,   # reference voltage noise in Volts (CDAC)
        "switching_strat": "monotonic",      # {monotonic, bss} used to determined initial starting voltages
        "unit_capacitance": 1e-15,           # unit capacitance
        "target_capacitance": None,          # Used for alternative
        "array_size": 8,                     # number of capacitor stages
        "array_N_M_expansion": False,        # Sizing strategy where
        "multiple_conversions": None,        # List bit positions in C array, with number of repetitions at each
        "use_rdac": False,                   # Set bit position which should
        "use_offset_cap": False,             # set to 0 farads, if disabled
        "use_split_cap": True,               # set to 0 farads, if disabled
        "parasitic_capacitance": 5.00e-14,   # estimate of capacitance at output (added to SPICE and ideal)
        "settling_time": 0.0e-9,             # individual settling errors per capacitor?
    },
}
```

CDAC model parameters: Switching scheme

Initial values of bottom-plate voltages differentiate ‘Monotonic’ vs ‘BSS’ switching



[C.C. Liu JSSC 2010]



Other schemes require different topology
but we already have good energy savings

[X. Tang IEEE TCAS 2022]

CDAC model parameters: Capacitor calculation

N_{CDAC} number of capacitor stages (on both N and P plate)

β radix: 2.0, 1.85, 1.8, etc

Approach #1: thinking in terms of LSB capacitor size

C_{unit} typically ~1 fF

$$\text{where } i = 0, 1, 2, \ \dots, \ (N_{CDAC} - 1)$$

$$w_i = \beta^i$$ weights

$$C_i = C_{unit} \cdot w_i$$ capacitor values

```
>>> w_radix1p8 = [2.0**i for i in range(8)]
[1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0]

>>> w_radix2 = [1.8**i for i in range(8)]
[1.0, 1.8, 3.24, 5.83, 10.5, 18.9, 34.01, 61.22]
```

Approach #2: If we care about total capacitance

C_{total} typically ~ 100 fF

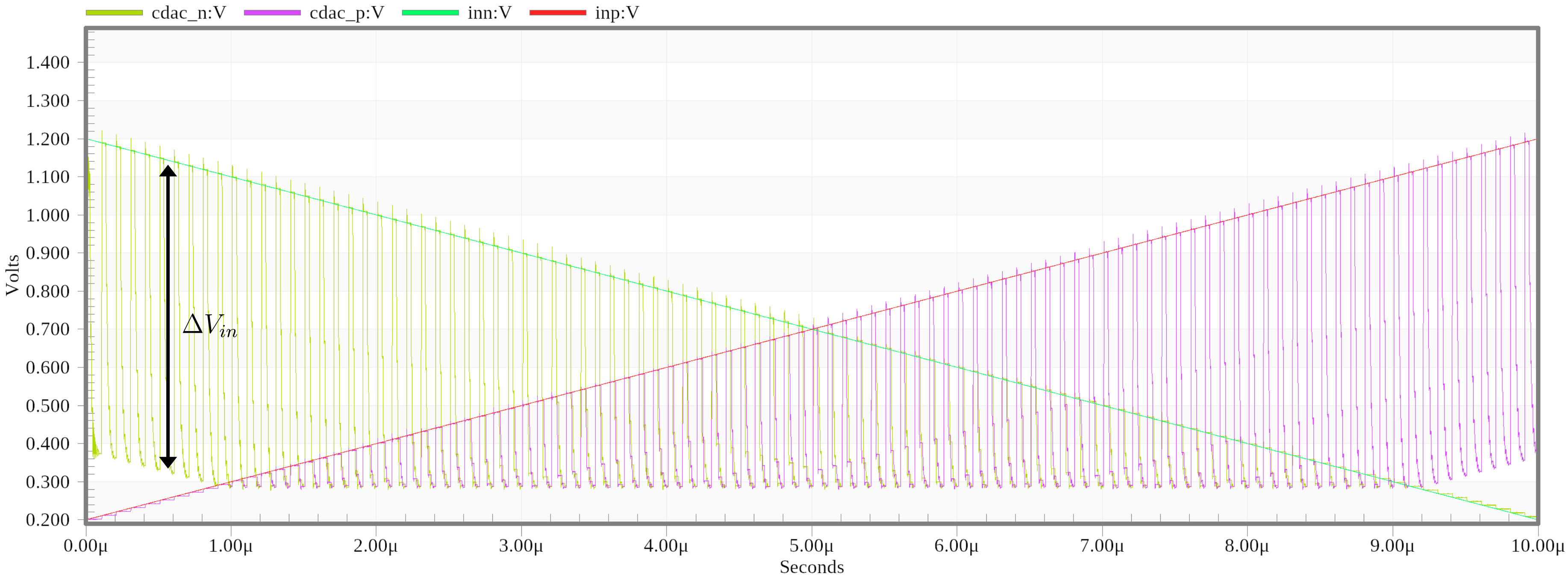
$$\text{where } i = 0, 1, 2, \ \dots, \ (N_{CDAC} - 1)$$

$$C_i = \frac{C_{total}}{2 \cdot \beta^{(N_{CDAC} - 1 - i)}}$$ capacitor values

$$w_i = \frac{C_i}{C_0}$$ weights

Approach #3: If we care about allowable input signal swing, account for split offset and parasitic caps

CDAC model parameters: Input voltage swings



The differential input range is limited by the CDAC and additional + parasitic caps

In this 9-bit 1.2V ref case: $\Delta V_{in} \approx \frac{C_{\text{dac}}}{C_{\text{total}}} \cdot V_{\text{ref}} = \frac{136}{61 + 136} \cdot 1.2 = 0.826$

Also note, non-ideal comparator has common-mode input limitation

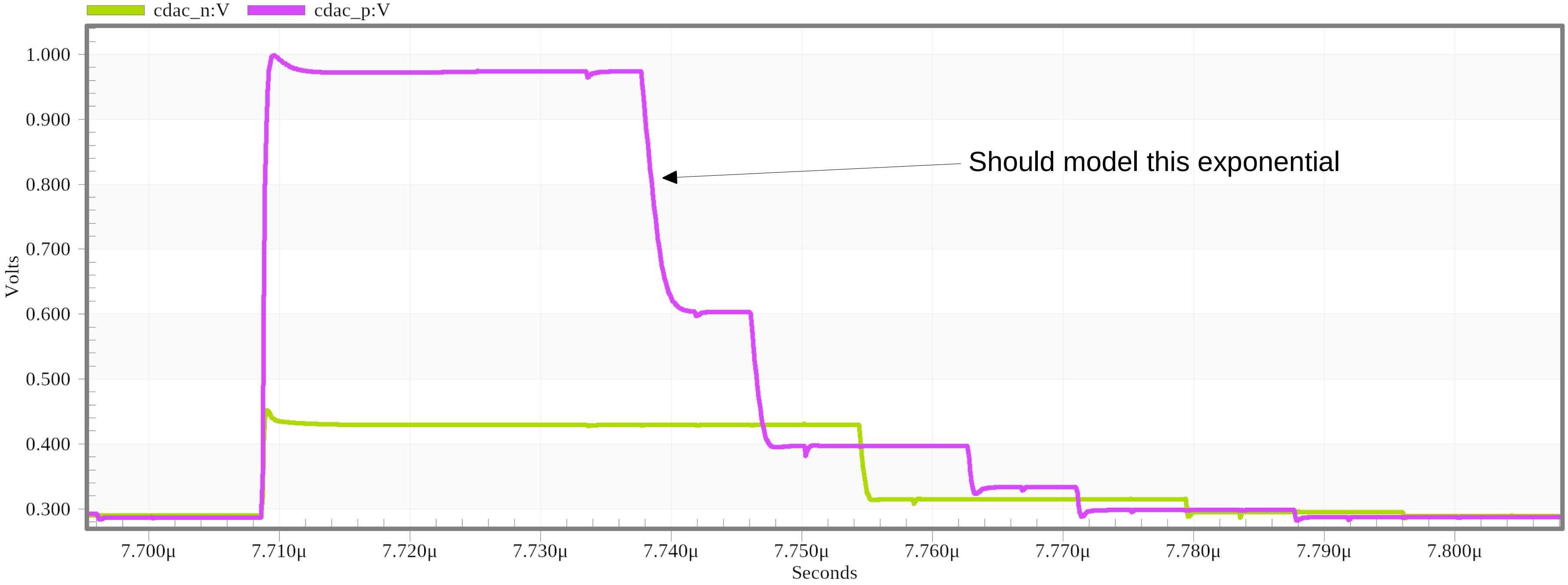
CDAC model parameters: Settling error

Monotonic switching ameliorates RC delay, but it will still manifest as voltage error when:

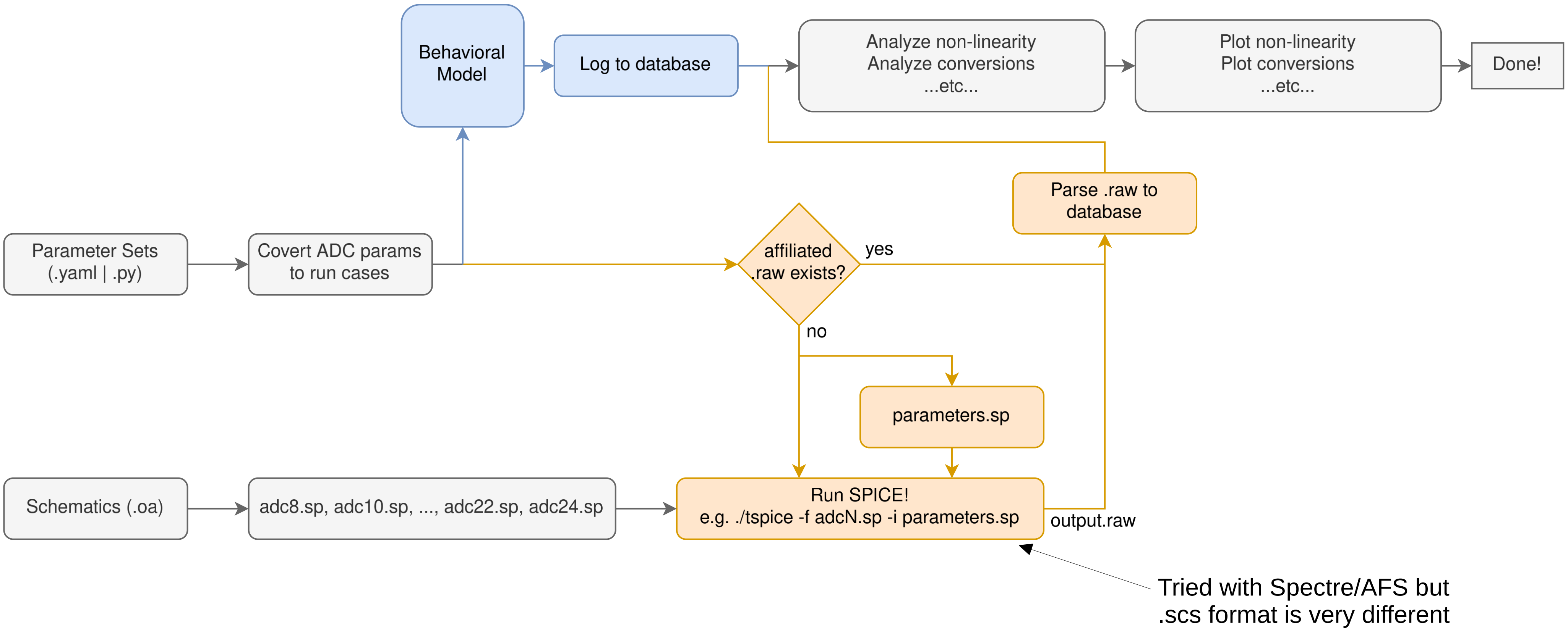
Clock periods are short

Differential input voltage is large

$$\text{settling\_time\_error} = e^{-\frac{1}{\tau_S \cdot f_S \cdot (N+1)}}$$



Workflow: SPICE & behavioral models use same params



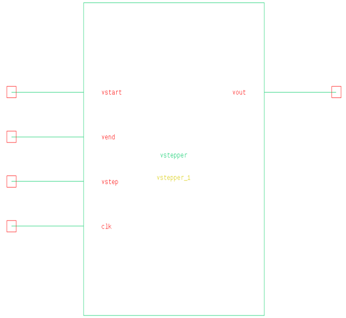
Testbench parameters: Syncing input voltages and clocks

```
-----Behavioral dataframe-----
               Vin
0      -0.59999333
1      -0.59996000
2      -0.59992667
3      -0.59989333
4      -0.59986000
...            ...
35995   0.59986000
35996   0.59989333
35997   0.59992667
35998   0.59996000
35999   0.59999333
```

```
-----SPICE dataframe-----
            Vin
0      -0.59994
1      -0.59991
2      -0.59988
3      -0.59985
4      -0.59982
...         ...
39991   0.59982
39992   0.59985
39993   0.59988
39994   0.59991
39995   0.59994
```

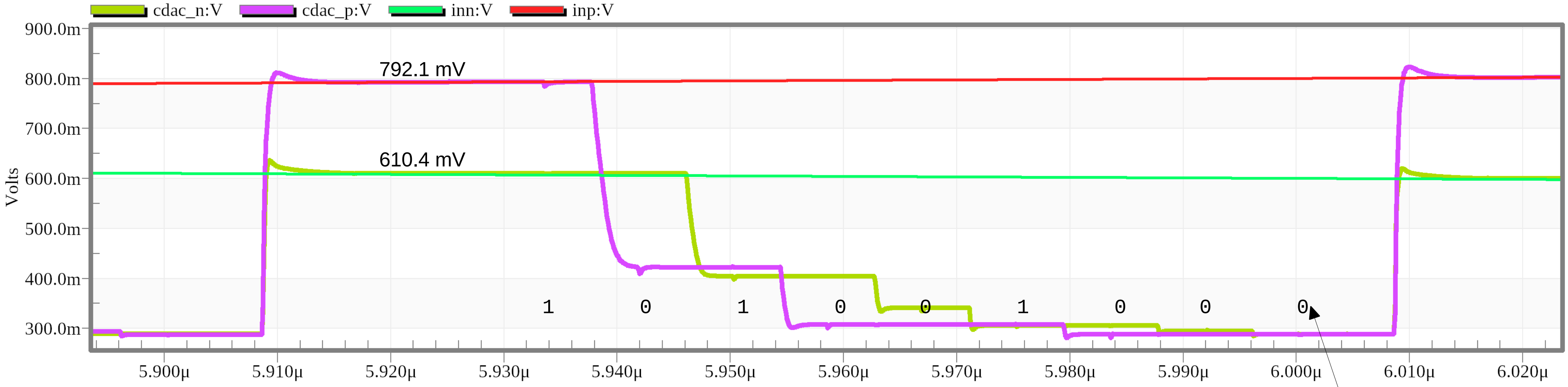
Because we are trying to directly compare SPICE vs behavioral data, it’s best if the timing and voltage references are synced

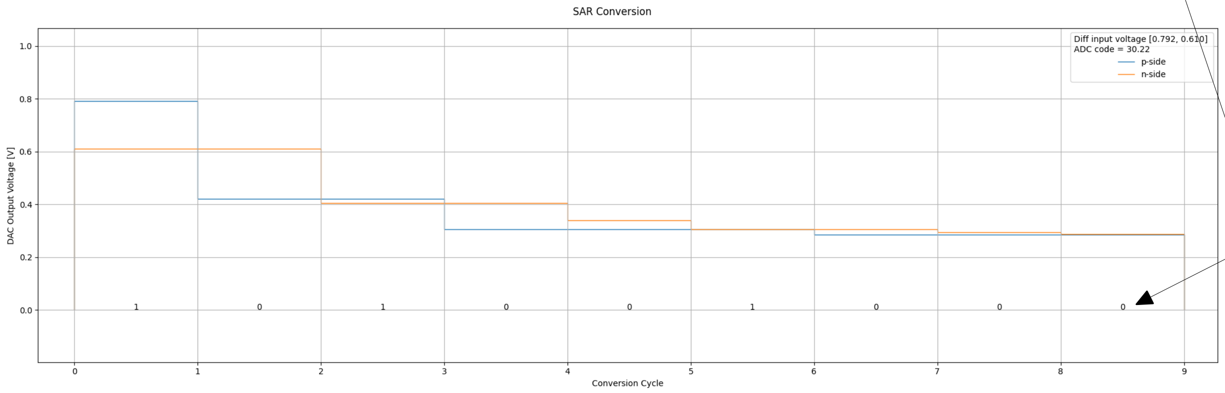
We can correct this with Verilog-A models which are easier to handle than SPICE primitives




```
`include "discipline.h"
`include "constants.h"

module vstepper(vstart, vstep, vend, clk, vout);
    input vstart, vstep, vend, clk;
    output vout;
    electrical vstart, vstep, vend, clk, vout;

    // Internal variable to store the current voltage
    real current_voltage;
    real next_update_time;

    // Initial conditions
    initial begin
        current_voltage = vstart;  // Start voltage is vstart
        next_update_time = $abstime;  // Initialize the update time
    end

    // Voltage update logic
    analog begin
        // Only update the voltage when the clock signal is active (high)
        if (V(clk) > 0) begin
            // If the current voltage is less than vend, increment
            if (current_voltage < V(vend)) begin
                current_voltage = current_voltage + V(vstep);
                if (current_voltage > V(vend)) begin
                    current_voltage = V(vend);  // Ensure voltage doesn't exceed vend
                end
            end
        end

        // Output the current voltage
        V(vout) <+ current_voltage;
    end

endmodule // vstepper
```

Does the behavioral model work?





yep,
for (most)
Vin values

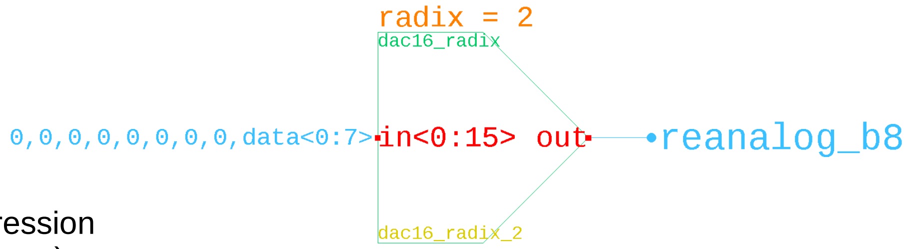
Testbench: Output value calculation (i.e. 're-analog')

$$R = \sum_{i=0}^{N_{CDAC}-1} b_i \cdot w_i$$

$$\text{where } i = 0, 1, 2, \ \ldots\ , \ (N_{CDAC} - 1)$$

$$w_i = \beta^i$$

Equivalent expression
(except 0 → 1 range)



$$\text{cycles} = 1, 2, 3, 4, 5, 6, 7, 8$$

$$\left.\begin{aligned} \text{weights} &= w_i = w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0 \\ \text{SA register} &= r_i = r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0 \\ \text{data out} &= b_i = b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0 \end{aligned}\right\}$$

8-capacitor
CDAC/ADC

```
>>> w_radix2 = [2.0**i for i in range(8)]
[1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0]

>>> sum(w_radix2)
255.0

>>> w_radix1p8 = [1.8**i for i in range(8)]
[1.0, 1.8, 3.24, 5.83, 10.5, 18.9, 34.01, 61.22]

>>> sum(w_radix1p8)
136.4995072
```

```
.subckt dac16_radix in<0> in<1> in<2> in<3> in<4> in<5>
+ in<6> in<7> in<8> in<9> in<10> in<11> in<12> in<13>
+ in<14> in<15> out radix=1.8
```

(default value)

```
eout out gnd vol='
V(in<15>)/pow(radix, 0)+
V(in<14>)/pow(radix, 1)+
V(in<13>)/pow(radix, 2)+
V(in<12>)/pow(radix, 3)+
V(in<11>)/pow(radix, 4)+
V(in<10>)/ow(radix, 5)+
V(in<9>)/pow(radix, 6)+
V(in<8>)/pow(radix, 7)+
V(in<7>)/pow(radix, 8)+
V(in<6>)/pow(radix, 9)+
V(in<5>)/pow(radix, 10)+
V(in<4>)/pow(radix, 11)+
V(in<3>)/pow(radix, 12)+
V(in<2>)/pow(radix, 13)+
V(in<1>)/pow(radix, 14)+
V(in<0>)/pow(radix, 15)'
.ends
```

Testbench: Output value calculation (i.e. 're-analog')

$$R = \sum_{i=0}^{N_{CDAC}-1} b_i \cdot w_i$$

$$w_i = \beta^i$$
$$\text{where } i = 0, 1, 2, \; \dots \;, \; (N_{CDAC} - 1)$$

$$R = \sum_{i=1}^{N_{CDAC}} (2 \cdot b_i - 1) \cdot w_{i-1} + b_0 - 1$$

Can we use the trailing comparator output as an ADC bit?

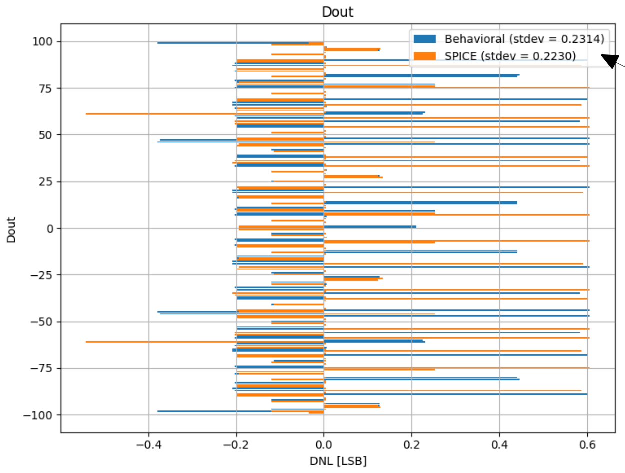
$$\text{cycles} = 1, 2, 3, 4, 5, 6, 7, 8$$
$$\text{weights} = w_i = w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0$$
$$\text{SA register} = r_i = r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0$$
$$\text{data out} = b_i = b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$$

8-capacitor CDAC/ADC

8-cap CDAC 9-bit ADC

$$\text{cycles} = 1, 2, 3, 4, 5, 6, 7, 8, 9$$
$$\text{weights} = w_i = w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0$$
$$\text{SA register} = r_i = r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0$$
$$\text{data out} = b_i = b_8, b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$$

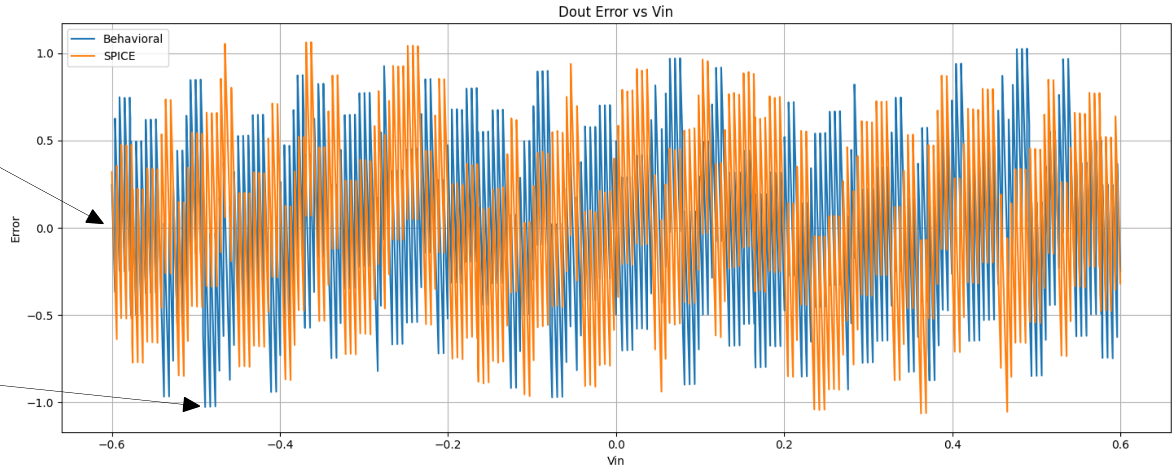
Behavioral vs SPICE: Linearity comparison



DNL stddev is similar

Not exercising full range

9-bit ADC
1.8 radix
-99 to 98 range

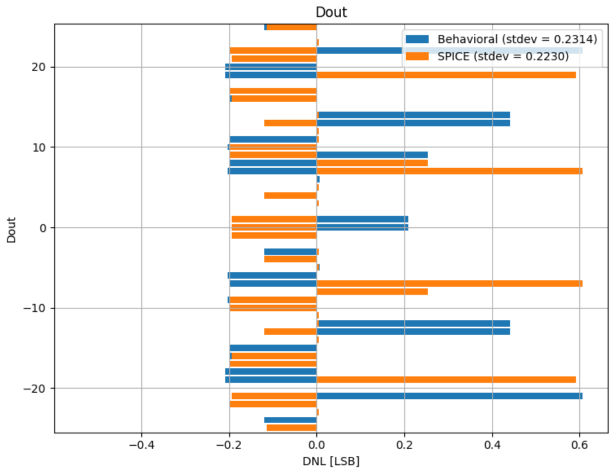


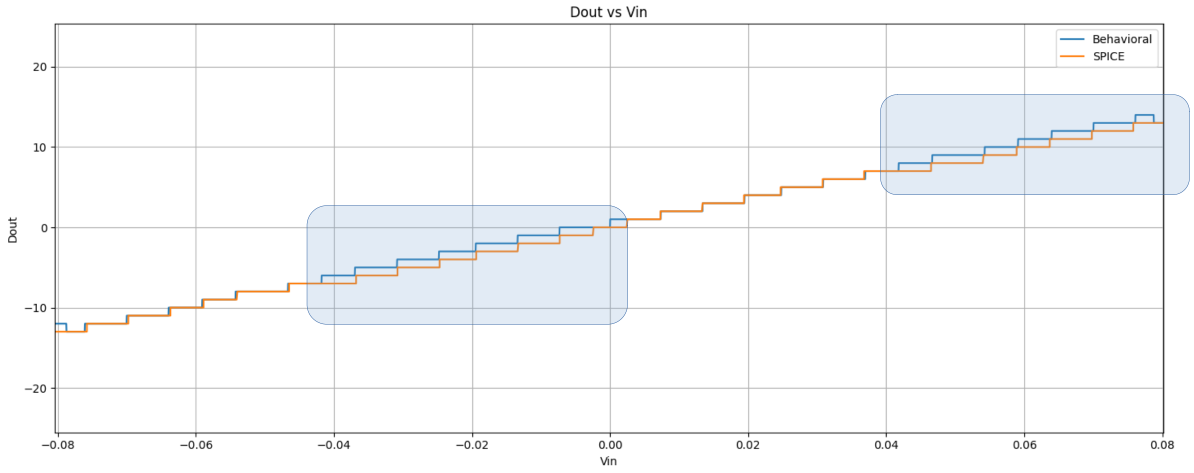
quantization error (RMS) is identical @ 0.278 LSB

But ideal ADC would have 1/sqrt(12) = 0.289 LSB RMS. I think I need to calibrate for Dout range of ~197.

Worst-case INL

Behavioral vs SPICE: Linearity comparison





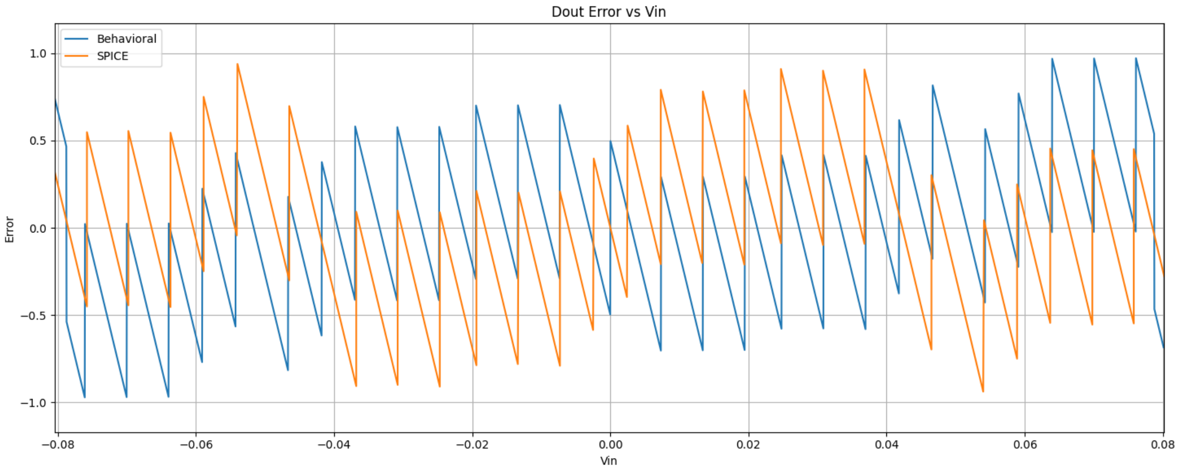
1 LSB errors exist

Is rounding applied at different stages?

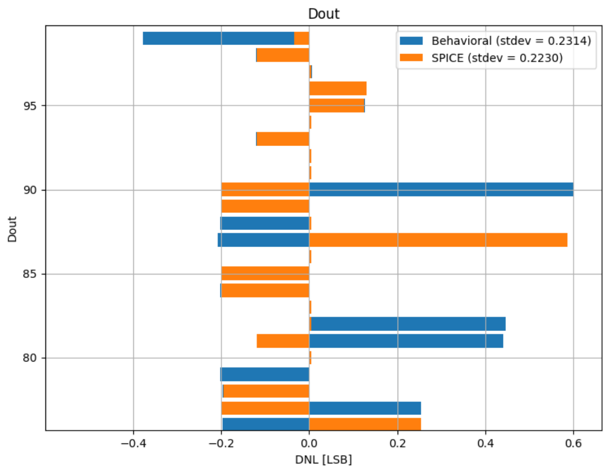
Perhaps parasitic capacitance is throwing us off? I calculated ~50 fF for the parasitics plus

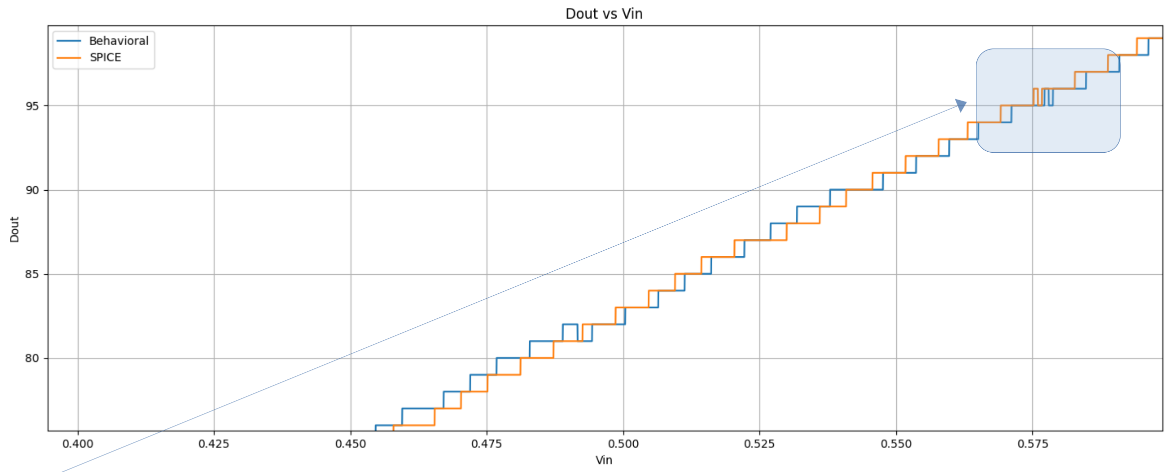
```
-----Behavioral dataframe-----
              Vin          Dout  Dout_rounded
0      -0.59999333  -98.2403072         -98.0
1      -0.59996000  -98.2403072         -98.0
2      -0.59992667  -98.2403072         -98.0
3      -0.59989333  -98.2403072         -98.0
4      -0.59986000  -98.2403072         -98.0
...            ...          ...           ...
35995   0.59986000   99.2403072          99.0
35996   0.59989333   99.2403072          99.0
35997   0.59992667   99.2403072          99.0
35998   0.59996000   99.2403072          99.0
35999   0.59999333   99.2403072          99.0
```

```
-----SPICE dataframe-----
           Vin          Dout  Dout_rounded
0     -0.59994   -98.7403072         -99.0
1     -0.59991   -98.7403072         -99.0
2     -0.59988   -98.7403072         -99.0
3     -0.59985   -98.7403072         -99.0
4     -0.59982   -98.7403072         -99.0
...        ...           ...           ...
39991  0.59982    98.7403072          99.0
39992  0.59985    98.7403072          99.0
39993  0.59988    98.7403072          99.0
39994  0.59991    98.7403072          99.0
39995  0.59994    98.7403072          99.0
```



Behavioral vs SPICE: Linearity comparison

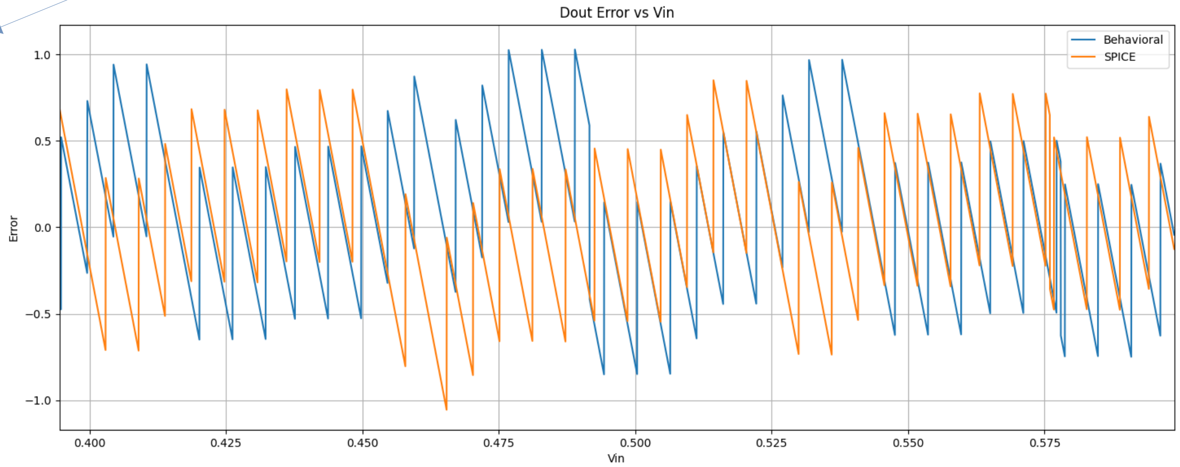




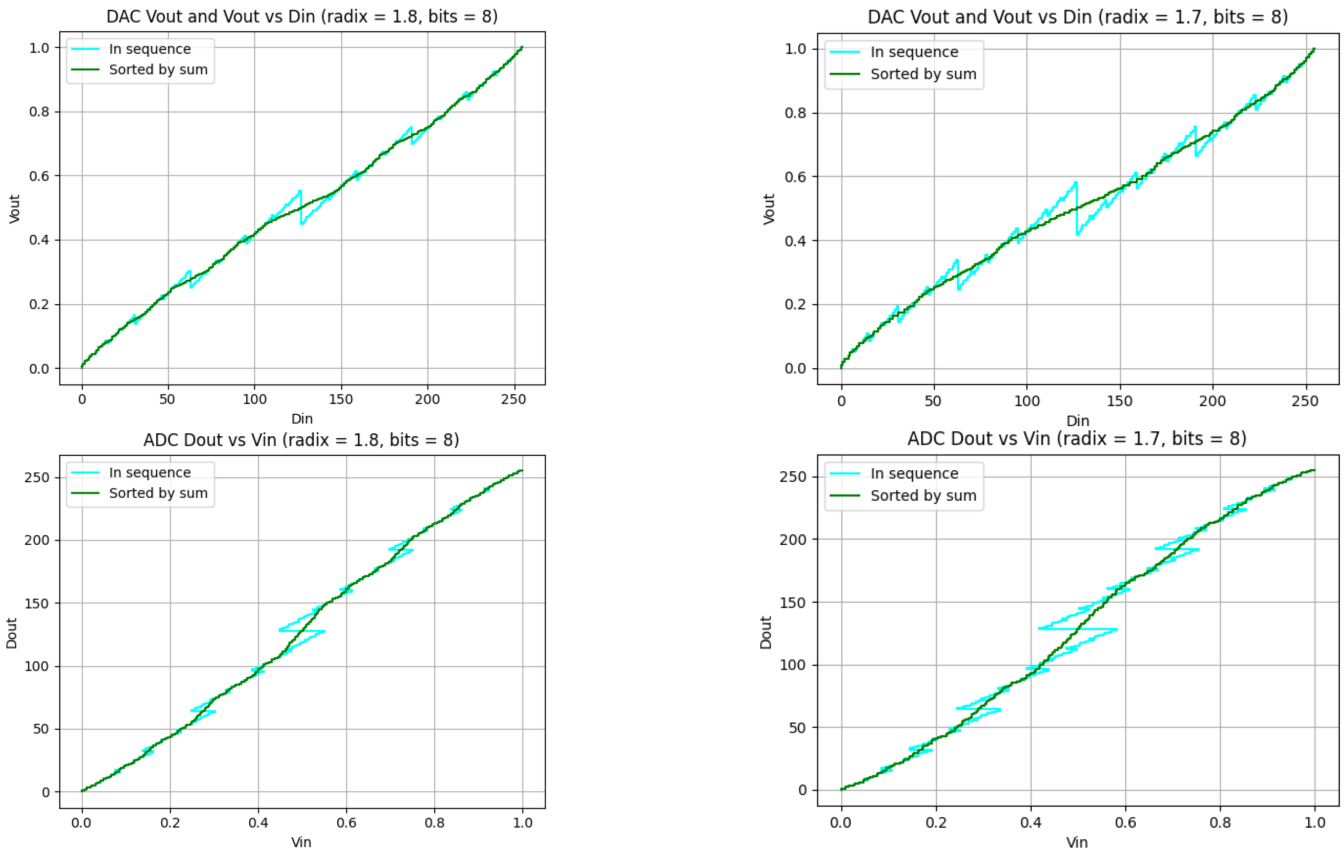
comp	data<0>	data<1>	data<2>	data<3>	data<4>	data<5>	data<6>	data<7>
0	1	1	0	0	1	0	1	1
1	1	1	0	0	1	0	1	1
0	0	0	1	0	1	0	1	1
1	0	0	1	0	1	0	1	1

Vin	Dout
0.48723	79.564147
0.48990	80.564147
0.48993	80.444147
0.49257	81.444147

Some codes are non-monotonic, in both SPICE and behavioral models?



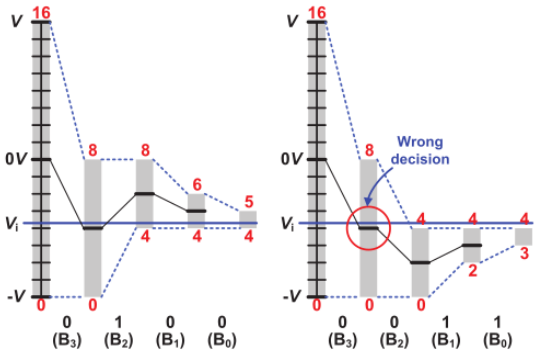
How are non-binary redundant codes distributed?



What about double conversions?

Side note: What type of adder hardware is ideal?

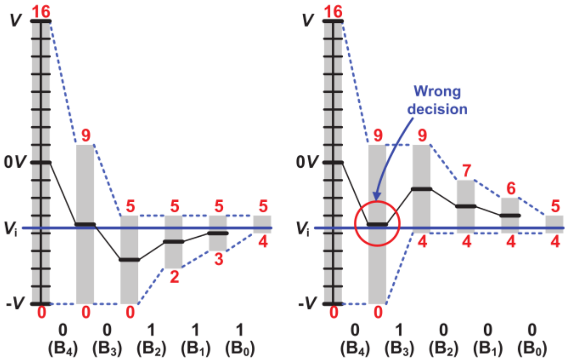
Conventional binary



$D_{out} = 8 \cdot B_3 + 4 \cdot B_2 + 2 \cdot B_1 + 1 \cdot B_0 = 8 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 0 = 4$ (correct)

$D_{out} = 8 \cdot B_3 + 4 \cdot B_2 + 2 \cdot B_1 + 1 \cdot B_0 = 8 \cdot 0 + 4 \cdot 0 + 2 \cdot 1 + 1 \cdot 1 = 3$ (incorrect)

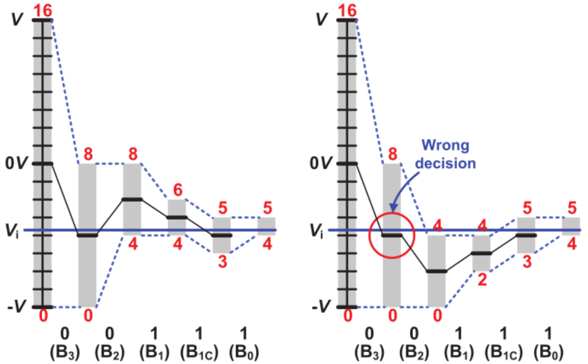
Non-binary (integer rounded weights)



$D_{out} = 7 \cdot B_4 + 4 \cdot B_3 + 2 \cdot B_2 + 1 \cdot B_1 + 1 \cdot B_0 = 7 \cdot 0 + 4 \cdot 0 + 2 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 4$ (correct)

$D_{out} = 7 \cdot B_4 + 4 \cdot B_3 + 2 \cdot B_2 + 1 \cdot B_1 + 1 \cdot B_0 = 7 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 = 4$ (correct)

"Binary-scaled compensation/recombination"



$D_{out} = -1 + 8 \cdot B_3 + 4 \cdot B_2 + 2 \cdot B_1 + 2 \cdot B_{1C} + 1 \cdot B_0 = -1 + 8 \cdot 0 + 4 \cdot 0 + 2 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 = 4$ (correct)

$D_{out} = -1 + 8 \cdot B_3 + 4 \cdot B_2 + 2 \cdot B_1 + 2 \cdot B_{1C} + 1 \cdot B_0 = -1 + 8 \cdot 0 + 4 \cdot 0 + 2 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 = 4$ (correct)

The repeated conversion steps need an offset of half their weight to account for the 'bias' they introduce

[C.C. Liu JSSC 2015]

Next steps?

```python
params = {
    "ADC": {
        "bit_size": 8,                      # nominal resolution of the ADC (switching between netlists)
        "sampling_frequency": 10.0e6,       # sampling rate in Hz, used to driver clock sources
        "jitter": 0.0e-12,                  # aperture jitter in seconds (TBD)
        "device_noise": False,              # enables basic gaussian noise in behavioral, and tran noise in SPICE
    },
    "TESTBENCH": {
        "positive_input_voltages": [0.2, 1.2, 20e-6],      # start, end (incl.), and step voltage
        "negative_input_voltages": [1.2, 0.2, 20e-6],
        "use_calibration": False,          # account for cap error when calculating Dout (re-analog)
        "pdk_file": "\"~/helena/tech/tsmc65/default_testbench_header_55ulp_linux.lib\" tt",
        "spicedir": None,                  # Use this to write netlist from template
        "rawdir": None,                    # Use this to set SPICE output dir, and to read for parsing
    },
    "SWITCH": {
        "offset_voltage": 0.0e-3,          # offset voltage in Volts
        "common_mode_dependent_offset_gain": 0.0,  # common mode voltage gain
        "threshold_voltage_noise": True,
        "type": "passive",                 # supports active, passive, or ideal
        "strength": 4,
    },
    "COMP": {
        "offset_voltage": 0.0e-3,          # offset voltage in Volts
        "common_mode_dependent_offset_gain": 0.0,  # common mode voltage gain
        "threshold_voltage_noise": True,
        "strength": 4,                     # used to size some active devices (SPICE only)
    },
    "CDAC": {
        "positive_reference_voltage": 1.2,   # reference voltage in Volts
        "negative_reference_voltage": 0.0,   # reference voltage in Volts
        "reference_voltage_noise": 0.0e-3,   # reference voltage noise in Volts (CDAC)
        "switching_strat": "monotonic",      # {monotonic, bss} used to determined initial starting voltages
        "unit_capacitance": 1e-15,           # unit capacitance
        "target_capacitance": None,          # Used for alternative
        "array_size": 8,                     # number of capacitor stages
        "array_N_M_expansion": False,        # Sizing strategy where
        "multiple_conversions": None,        # List bit positions in C array, with number of repetitions at each
        "use_rdac": False,                   # Set bit position which should
        "use_offset_cap": False,             # set to 0 farads, if disabled
        "use_split_cap": True,               # set to 0 farads, if disabled
        "parasitic_capacitance": 5.00e-14,   # estimate of capacitance at output (added to SPICE and ideal)
        "settling_time": 0.0e-9,             # individual settling errors per capacitor?
    },
}
```