Floating Point ADC Converter

Nathan Kim

July 2025

Project Summary

This project designed a **current-mode floating-point ADC** using translinear circuits to achieve **96 dB dynamic range** (3.63 pA to 238 nA) with only **10 bits** (4-bit exponent + 6-bit mantissa). Key innovations:

- **Translinear Circuits**: The binary search is performed on the exponent using logarithmic properties, modeled in code by (`e_cell`).
- **Modeling Circuit in C**: Verified that the ADC architecture was capable of extracting exponent bits and scaling the signal for proper mantissa bit calculation, and that the results could be put into IEEE-754 encoding by modeling the circuit's computations in C. All input arrays resulted in expected outputs.
- **Area/Power Savings**: 4-bit exponent covers 16 decades, reducing comparator count vs. fixed-point ADCs. Ideal for biomedical/audio applications with wide dynamic ranges.

Project Constraints:

- **Positive Currents Only** The sign bit in the IEEE-754 32 bit representation for floating numbers is always 0.

Definitions used throughout the report: $I_{ref} = 2^{-30} =$ `REF`, $I_{mid} = 2^{-21}$ $=$ `MID`, MAX $= 2^{-22} =$ `MAX`, MIN $= 2^{-38} =$ `MIN`.

Definitions used in code: `REF` $= 2^{-30}$, `MID` $= 2^{-21}$, `MAX` $= 2^{-22}$, `MIN` $= 2^{-38}$.

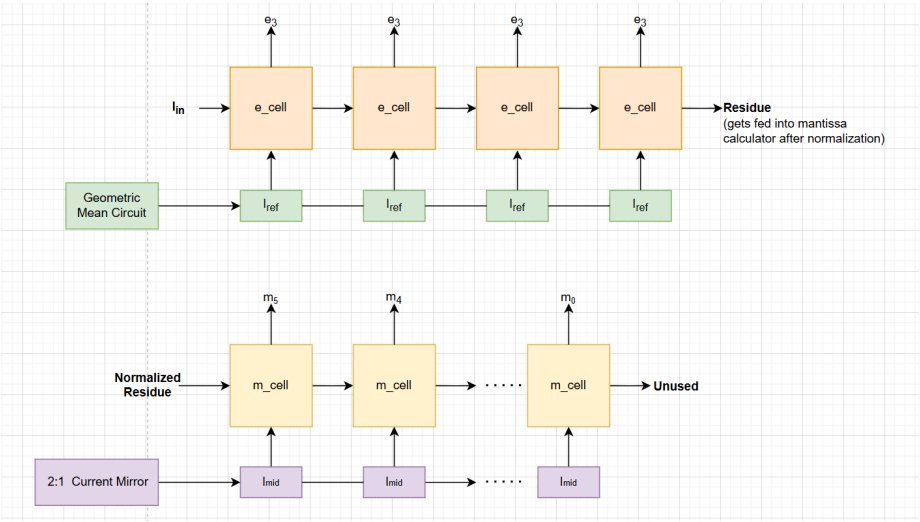

Figure 1: The output of the exponent cell array goes to the mantissa cell array

1 Circuit Stages and Overview

The block diagram of the circuit is in Figure 1

The input current first flows through the exponent cell array, where it is repeatedly compared to the geometric mean, I_{ref}, produced by a translinear circuit.

Then the result is scaled to a normalized range from 0 to 2^{-22} amps. In that range, the midpoint is 2^{-21} amps. At this point, the current is ready for mantissa processing and is passed along to the mantissa cell array.

Quick note on midpoints vs. geometric means: `MID` is simply `MAX/2`. Halfway between the maximum and zero. `REF`, on the other hand, is the logarithmic halfway point. So between 2^{-38} and 2^{-22}, the geometric midpoint is 2^{-30}, because -30 is equidistant from -38 and -22.

The code that models the circuit and block diagram is uploaded to my github here. It models the computations of my circuit. It verifies that it is capable of outputting bits in IEEE 754 format, as seen in Figure 3 and Figure 4.

An in-depth explanation of the computations and theory behind the floating-point conversion is in the section ADC Computations Explained.

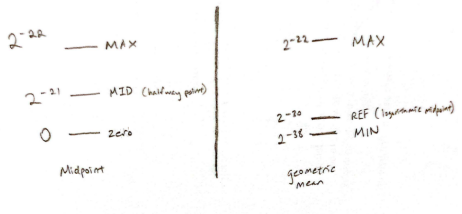

Figure 2: `MID` is used in mantissa bit calculations, `REF` is used in exponent bit calculations

```
root@DESKTOP-EBV98JS:~/AVLSI_2025/floating_point_ADC# ./floating_point_ADC
0 01101000 1.00000000000000000000000 1.19209e-07
0 01101000 1.00000

0 01101000 1.01000000000000000000000 1.49012e-07
0 01101000 1.01000

0 01101000 1.10000000000000000000000 1.78814e-07
0 01101000 1.10000

0 01101000 1.11000000000000000000000 2.08616e-07
0 01101000 1.11000

0 01101000 1.11111111111111110101100 2.38417e-07
0 01101000 1.11111
```

Figure 3: The first line is the expected output, the computed exponent bits and mantissa is the second line

```
0 01101000 1.11000000000000000000000 2.08616e-07
0 01101000 1.11000

0 01101000 1.11111111111111110101100 2.38417e-07
0 01101000 1.11111

0 01101001 1.00000000000000000000000 2.38419e-07
0 01101000 1.11111

0 01100001 1.00110000101000111101100 1.10827e-09
0 01100001 1.00110

0 01100010 1.00011110101110000101001 2.08616e-09
0 01100010 1.00011

0 01100011 1.01010100011110101110001 4.95464e-09
0 01100011 1.01010

0 01100000 1.00111011001000101101000 5.73229e-10
0 01100000 1.00111
```

Figure 4: Sign bit (1 bit), Exponent bits (8 bits), Calculated Mantissa (6 bits)

Table 1: Floating Point Encoding Results

Input Current	Expected (Binary + Float)	Output (Binary)
$1.00 \cdot 2^{-30} = 9.31 \cdot 10^{-10}$	0 01101000 1.00000000000000000000000 = $1.19209 \cdot 10^{-7}$	0 01101000 1.00000
$1.25 \cdot 2^{-30} = 1.16 \cdot 10^{-9}$	0 01101000 1.01000000000000000000000 = $1.49012 \cdot 10^{-7}$	0 01101000 1.01000
$1.50 \cdot 2^{-30} = 1.40 \cdot 10^{-9}$	0 01101000 1.10000000000000000000000 = $1.78814 \cdot 10^{-7}$	0 01101000 1.10000
$1.75 \cdot 2^{-30} = 1.63 \cdot 10^{-9}$	0 01101000 1.11000000000000000000000 = $2.08616 \cdot 10^{-7}$	0 01101000 1.11000
$1.99999 \cdot 2^{-30}$	0 01101000 1.11111111111111110101100 = $2.38417 \cdot 10^{-7}$	0 01101000 1.11111
$1.00 \cdot 2^{-29}$	0 01101001 1.00000000000000000000000 = $2.38419 \cdot 10^{-7}$	0 01101000 1.11111
$1.19 \cdot 2^{-38}$	0 01100001 1.00110000101000111101100 = $1.10827 \cdot 10^{-9}$	0 01100001 1.00110
$1.12 \cdot 2^{-37}$	0 01100010 1.00011110101110000101001 = $2.08616 \cdot 10^{-9}$	0 01100010 1.00011
$1.33 \cdot 2^{-38}$	0 01100011 1.01010100011110101110001 = $4.95464 \cdot 10^{-9}$	0 01100011 1.01010
$1.231 \cdot 2^{-31}$	0 01100000 1.00111011001000101101000 = $5.73229 \cdot 10^{-10}$	0 01100000 1.00111
$1.20 \cdot 2^{-32}$	0 01011111 1.00110011001100110011010 = $2.79397 \cdot 10^{-10}$	0 01011111 1.00110
$1.01 \cdot 2^{-23}$	0 01101000 1.00000010100011110101110 = $1.20401 \cdot 10^{-7}$	0 01101000 1.00000
$1.01 \cdot 2^{-38}$	0 01011010 1.00000010100011110101110 = $7.34872 \cdot 10^{-12}$	0 01011010 1.00000

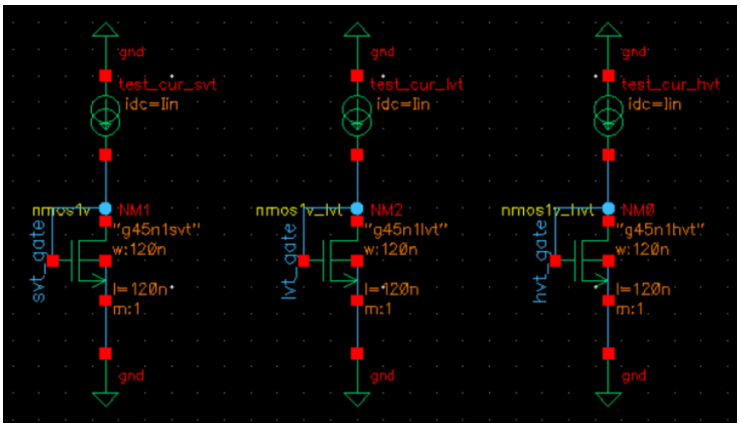

Figure 5: Sweeping design variable `Iin`

2 Subthreshold Circuit Design Steps

2.1 Minimum and Maximum Values

So what gate to source voltages are needed to produce MIN and MAX? Figure 5 shows my setup for testing.

The results of Figure 7 and Figure 6 show that the corresponding voltages are 100 and 480 millivolts.

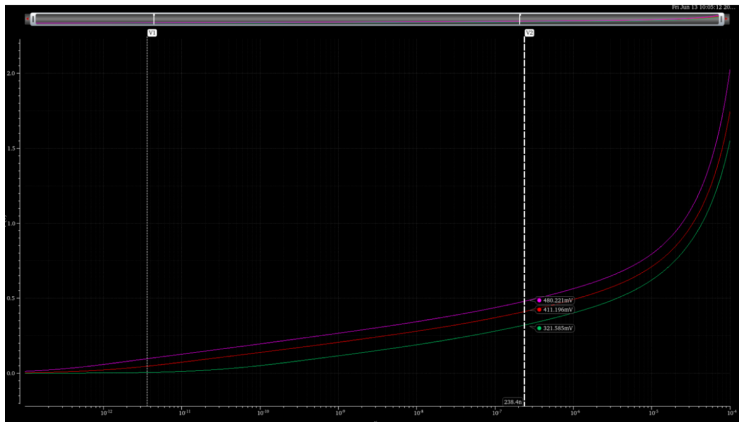

Figure 6: For High Threshold Voltage transistors, about 480 mV produces MAX current

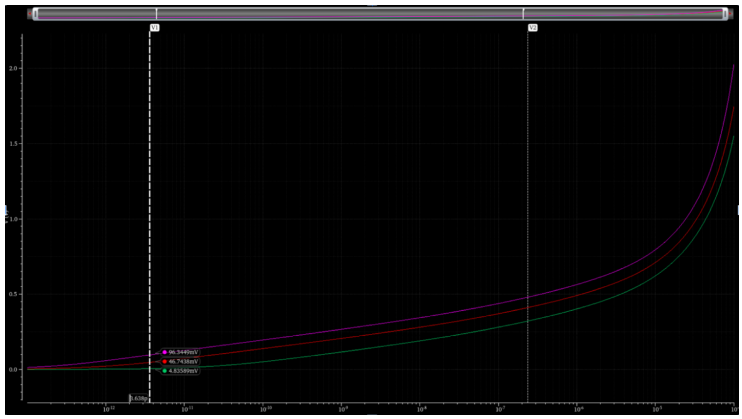

Figure 7: For HVT transistors, about 100 mV produces the MIN current

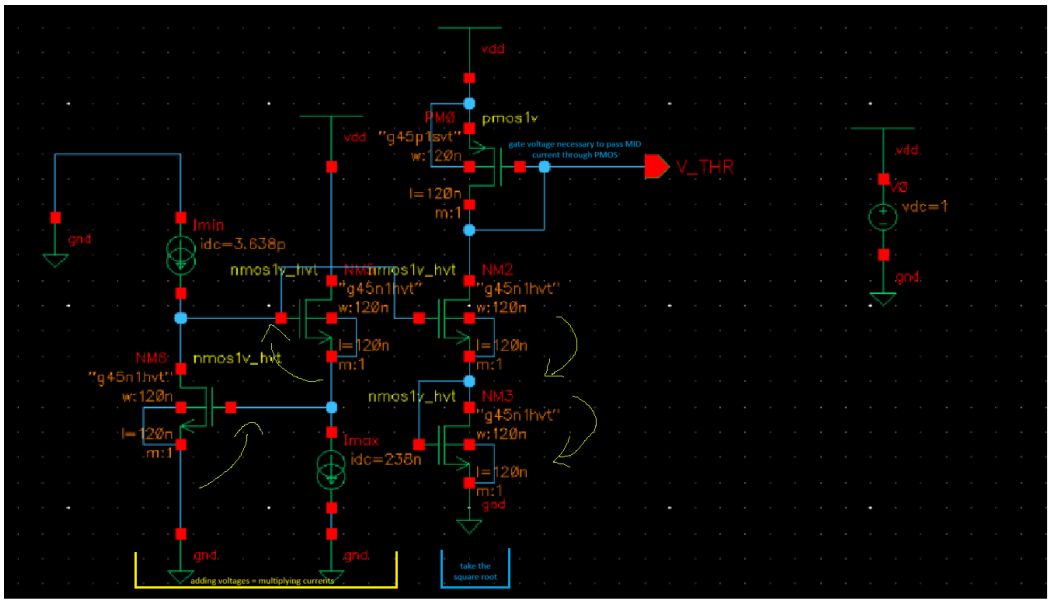

Figure 8: Outputs the correct gate voltage for generating the mid value current

2.2 Circuit Stages and Explanations

2.2.1 Generating Reference Current and Voltage

First, there must be a circuit capable of generating 2^{-30} amps, or I_{ref}.

The translinear circuit in Figure 8 exploits the logarithmic voltage-current relationship in subthreshold operation: voltage addition becomes current multiplication, and subtraction becomes division.

The PMOS transistor in this circuit serves a critical role - it converts the target current (I_{ref}) into its corresponding gate voltage. This works because:

- In subthreshold, PMOS gate voltage relates logarithmically to drain current: $V_{gs} \propto \ln(I_D)$
- The circuit automatically adjusts V_{gs} until the drain current matches I_{ref}

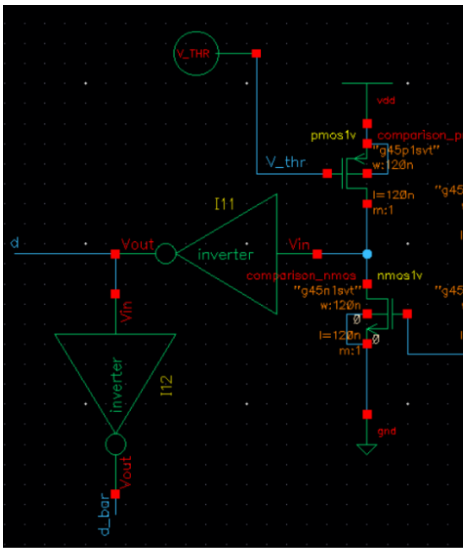

Figure 9: Input current gets compared to reference current I_{ref}

- This voltage then propagates to subsequent stages for current division

2.2.2 Comparison Circuit

Figure 9 shows that top `comparison_pmos` will provide the current I_{ref}. The bottom `comparison_nmos` will provide the input current I_{in}.

- If I_{in} is below I_{ref}, the node connected to V_{in} will be pulled down to ground.
- If I_{in} is above I_{ref}, the node connected to V_{in} will be pulled up to V_{dd}.

The voltage at `Vin` is processed through two inverters to generate complementary digital outputs:

- $d = 1$ when $I_{in} > I_{ref}$
- $d = 0$ when $I_{in} < I_{ref}$

These signals control the current division switches (shown in Figure 10):

- `min_cur_transistor` (activated when $d = 0$)

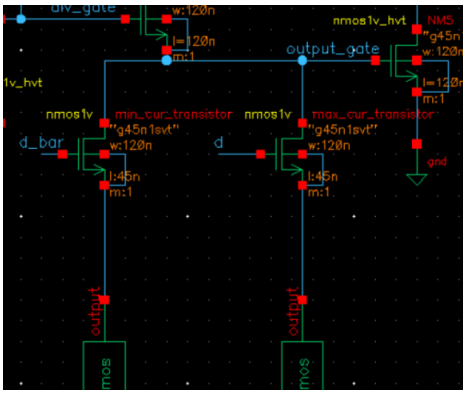

Figure 10: turning on `min_cur_transistor` selects MIN; turning on `max_cur_transistor` selects MAX

- `max_cur_transistor` (activated when $d = 1$)

The switches select the appropriate current path to compute I_{in}^2/K, where K is the division factor determined by the selected current mirror branch.

2.2.3 Exponent Cell

In Figure 11 the exponent cell is annotated. Note that from NM1 to NM2, the gate to source voltage doubles, squaring the current. Then it steps down by NM4's gate to source voltage which varies depending on what current `d` selects.

2.3 Mantissa Cell

For this circuit, I_{ref} is equal to MAX, not MID.

In mantissa_circuit is the mantissa cell. This design basically follows the standard ADC schematic:

$$d = \begin{cases} 0 & \text{if } 2 \cdot I_{\text{in}} < I_{\text{ref}} \\ 1 & \text{if } 2 \cdot I_{\text{in}} > I_{\text{ref}} \end{cases} \quad \Rightarrow \quad I_{\text{out}} = \begin{cases} 2 \cdot I_{\text{in}} & \text{if } d = 0 \\ 2 \cdot I_{\text{in}} - I_{\text{ref}} & \text{if } d = 1 \end{cases}$$

Also, note: comparing $2 \cdot I_{\text{in}}$ to a threshold like I_{ref} is mathematically equivalent to comparing I_{in} to $I_{\text{ref}}/2$.

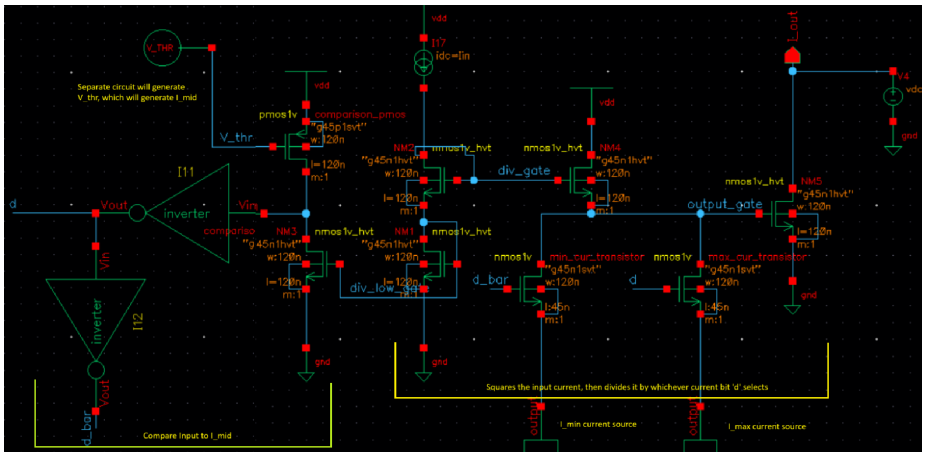

Figure 11: This circuit does the equivalent of what is defined in my code `e_cell`

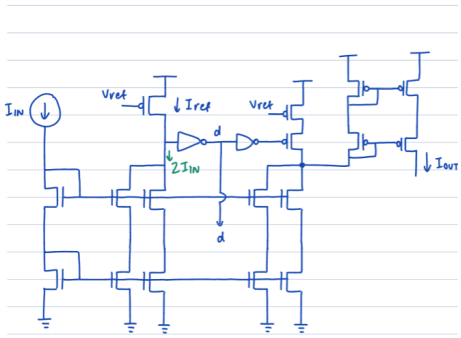

Figure 12: Mantissa Cell Circuit Schematic

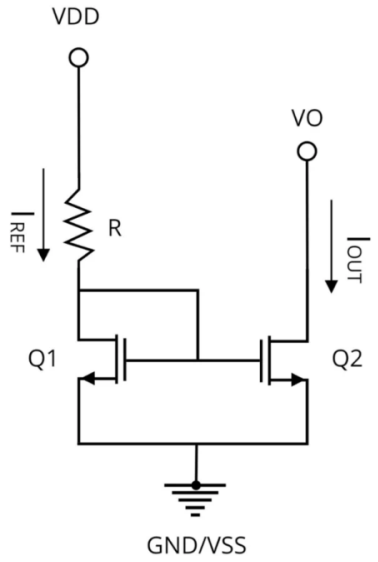

Figure 13: Q2 will fall out of saturation as soon as the drain drops below the largest overdrive voltage

2.4 Design Decisions: Accounting for Voltage Swing

This section refers to Figure 14, which really describes what goes on at the node `output_gate` seen in Figure 10.

2.4.1 High Threshold Transistors over Regular Threshold Transistors

Due to the extremely small currents and voltages being dealt with in the circuit, the output transistors of the current mirrors and squaring circuits were falling out of saturation.

The solution was to switch to high-threshold transistors, which had a larger threshold voltage value.

$$V_{ds} \geq V_{gs} - V_{th}$$

If V_{th} is larger, V_{ds} can be lower while staying in saturation.

2.4.2 Minimum Drain Voltage of Current Mirrors

What is the lowest possible value that the drain of transistor Q2 in Figure 13 could take on for this circuit while maintaining expected operation?

When the current mirror is asked to output the maximum current 2^{-22} amps, it requires gate-source voltages of 480 mV. The drain can drop down to

$$480 \ - V_{\text{th}} \text{ mV}$$

before the output transistor falls out of saturation. For a g45n1hvt operating in subthreshold, the V_{th} value is about 300 mV, so the lowest voltage the drain can swing to is about **180 mV** (this is an estimate).

2.4.3 Voltage Biasing

The voltage at `output_gate` must be sufficient to turn on the output transistor. The minimum voltage occurs when $I_{in} = 2^{-30}$ A:

$$V_{output} = 2 \cdot V_{gs,in} - V_{gs,current}$$

where:

- $V_{gs,in}$ = gate-source voltage for input current (264 mV at 2^{-30} A)
- $V_{gs,current}$ = gate-source voltage for divider current (480 mV)

This yields $V_{output} = 48$ mV - insufficient to turn on the output transistor or maintain saturation, as seen in Figure 14 The solution is to add bias voltages as shown in Fig. 15.

2.5 Accounting for Finite Conductance

The weakness of my circuit is that it assumes that if the right voltage is applied to the gate, then the exact expected current will flow. It assumes that no matter what the drain voltages at each current source is, the current will be the same.

Of course, this is not true, as seen in Fig. 16

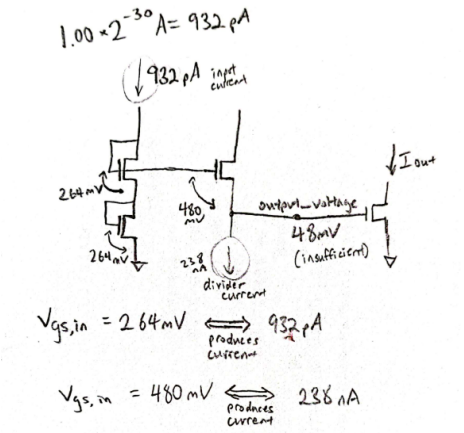

Figure 14: An input of 932 pA makes the voltage step up twice by 264 mV, then down once by 480 mV, causing `output_gate`'s voltage be unable to turn on the output transistor and pass the output current. Furthermore, it's connected to the current source `divider_current`, which is really just a current mirror that will fall out of saturation if `output_gate` is too low.

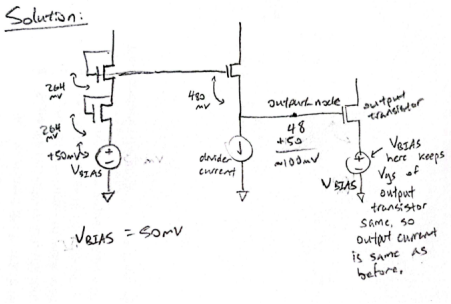

Figure 15: Adding a boost of 50 mV puts sufficient voltage at textttoutput_node

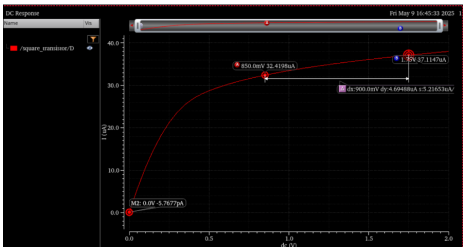

Figure 16: Finite Conductance

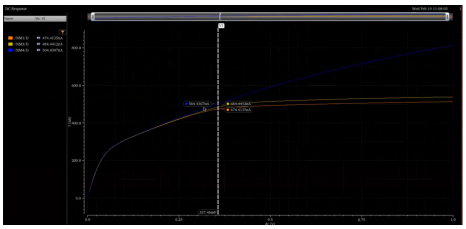

Figure 17: Cascoded Transistors have near infinite conductance

This is improved by adding cascodes to all my current sources. Every single current mirror in the circuit has gotten a cascode added to it. Adding a cascode greatly improves the conductances as seen in Figure 17

3 ADC Computations Explained

3.1 Binary Search on Exponent

We perform binary search on the exponent e in $x \cdot 2^e$, to determine the power-of-two scaling relative to MIN (2^{-38} amps). Observe the table below for examples:

Table 2: Exponent Search Results

Input (amps)	**Expected Exponent**	**Expected Output**
$x \cdot 2^{-38}$	0	0000
$x \cdot 2^{-37}$	1	0001
$x \cdot 2^{-36}$	2	0010
$x \cdot 2^{-35}$	3	0011
$x \cdot 2^{-34}$	4	0100
$x \cdot 2^{-33}$	5	0101
$x \cdot 2^{-32}$	6	0110
$x \cdot 2^{-31}$	7	0111
$x \cdot 2^{-30}$	8	1000
$x \cdot 2^{-29}$	9	1001
$x \cdot 2^{-28}$	10	1010
$x \cdot 2^{-27}$	11	1011
$x \cdot 2^{-26}$	12	1100
$x \cdot 2^{-25}$	13	1101
$x \cdot 2^{-24}$	14	1110
$x \cdot 2^{-23}$	15	1111

The exponent e represents how many steps above 2^{-38} the input is.

For example, $x \cdot 2^{-35}$ (exponent 3) means the current is $2^3 = 8$ times stronger than the minimum 2^{-38} amps.

The search dynamically scales the bounds using an ADC-inspired approach:

3.1.1 Rescaling Principle:

Let's consider regular ADC conversion. At each step, the range is halved, but mathematically rescaled to the original bounds (0–MAX) for consistent comparison logic.

- **Upper Half Search** (input > midpoint):

  $$\text{Next value} = 2 \cdot \text{in} - \text{MAX}$$

  Example: For the input 9 (range is 0–16)

  - The input is larger than the midpoint 8, so the upper half (8–16) becomes the new search range
  - Rescale the range to (0–16): $2 \cdot 9 - 16 = 2 \rightarrow 9$ becomes 2
  - Because being 1 above 8 is equivalent to being 2 above 0 (now we treat 8–16 as 0–16)

- **Lower Half Search** (input < midpoint):

  $$\text{Next value} = 2 \cdot \text{in}$$

  Example: For input the input 3 (range is 0–16)

  - The input is smaller than the midpoint 8, so the lower half (0–8) becomes the new search range
  - Rescale the range to (0–16): $2 \cdot 3 = 6 \rightarrow 3$ becomes 6
  - Equivalent to treating (8–16) as (0–16)

3.2 Translation to Searching the Exponent

Key Insight: The floating point ADC maintains binary search efficiency by mathematically transforming each sub-range to the original scale, allowing identical comparison logic at each step. It just logarithmic current steps rather than linear voltage ranges.

- **Doubling the exponent**: Scaling $e \to 2e$ corresponds to squaring the value:
$$2^e \to 2^{2e} = (2^e)^2 .$$
- **Subtracting a reference**: To compute 2^{2e-r} (for reference r), we divide:
$$2^{2e-r} = \frac{2^{2e}}{2^r}.$$

This mimics the ADC operation $\text{out} = 2 \cdot \text{in} - \text{ref}$, but adapted for exponential scaling.

3.2.1 New Minimums and Maximums: Why Do We Divide By A Different Value Depending On What Half We're In?

In the same way the range (8–16) or (0–8) is scaled to (0–16) as in the previous example, the range for the exponents (-38, -30) or (-30, -22) must be scaled to (-38, -22).

- Say the input is between 2^{-38} and 2^{-30}. Then to scale, 2^{-30} must become the new maximum. It must be scaled to 2^{-22}.
- What about if the input is between 2^{-30} and 2^{-22} (MAX)? The midpoint 2^{-30} (MID) must be scaled to 2^{-38} (MIN).

-30 is the midpoint (MID), equidistant from -22 (MAX) and -38 (MIN)

$$\text{MID} \cdot 2 = \text{MIN} + \text{MAX}$$
$$2 \cdot \text{MID} - \text{MAX} = \text{MIN} \quad \text{and} \quad 2 \cdot \text{MID} - \text{MIN} = \text{MAX}$$

This gives rise to the formula:

$$\text{out} = \left(\frac{\text{in}^2}{(\text{MAX}/\text{MIN})^d} \right) \cdot \text{min}, \quad d = \begin{cases} 0 & \text{if in} < \text{ref} \\ 1 & \text{if in} > \text{ref} \end{cases}$$

This is the exact logic written in my code that represents a single exponent cell's processing function, seen in Figure 18.

```
void e_cell(float iin,float *iout,int *dout)
{
  if(iin<REF){
    *dout=0;
    *iout=iin*iin/MIN;
  }else{
    *dout=1;
    *iout=iin*iin/MAX;
  }
}
```

Figure 18: Calculates current bit and next output current into the next cell