ADC Design

Report of Project 3

Zonghua Ouyang

Abhik Kumar

Department of Electrical and System Engineering

Mixed Signal Circuit Design and Modeling

GITHUB LINK

2025-12-25

ADC Design

Report of Project 3

Zonghua Ouyang

Abhik Kumar

Department of Electrical and System Engineering

Mixed Signal Circuit Design and Modeling

GITHUB LINK

2025-12-25

Contents

List of Figures

List of Tables

1

Descrption

1.1 Description of Pipeline ADC of 1-Bit Design

We tested out both 1.5 Bit and 1 Bit ADC. For 1.5 Bit we refereed lecture 16 ESE6680 slides to implement the ADC Stages.

1.1.1 Timing of Stages

Figure 1.11 shows the time diagram of each stage, and Figure 1.2 shows the time diagram of the last 3 stages and data alignment flip-flops.

As shown in Figure 1.1 and Figure 1.11, the clock polarity of stages are interleaved, so that each stage will be in the track stage when the previous stage is in the hold mode. The clock for comparators is slightly delayed to allow d flip-flops sampling the correct value.

The clock polarity of data alignment flip-flops is also interleaved, so that the d flip-flops always sample at the end of the track period of the stage. However, this will cause a problem that the data of even numbered stages will come at the rising edge while the data of odd numbered stages will come at the previous falling edge. To solve this problem, one more d flip-flop is added to every stages, and this d flip-flop is always triggered at the falling edge of the clock.

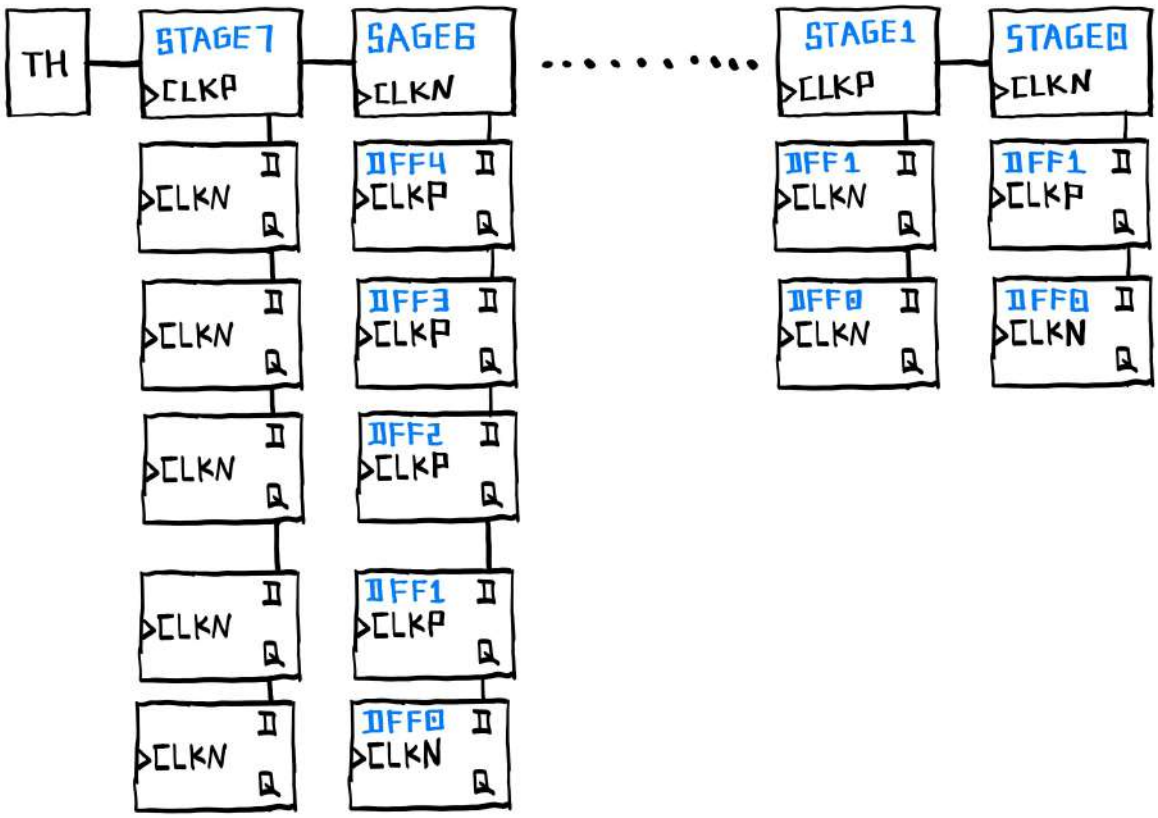


Figure 1.1: *Block Diagram of the ADC*

1.2 Description of Subcompoents for 1.5-Bit Stage

Pipeline analog-to-digital converters (ADCs) achieve high speed and resolution by breaking the conversion process into multiple stages, each resolving a small number of bits and passing the residue to the next stage. The 1.5-bit per stage architecture is a widely used design that balances performance, complexity, and error resilience.

Why 1.5 Bits?

Each 1.5-bit stage resolves two digital bits, but only three output codes are valid:

- `00` if $V_i < -V_R4$ for DAC 1V
- `01` if $-V_R4 < V_i < V_R4$ for DAC 2.5V
- `10` if $V_i > V_R4$ for DAC 4V

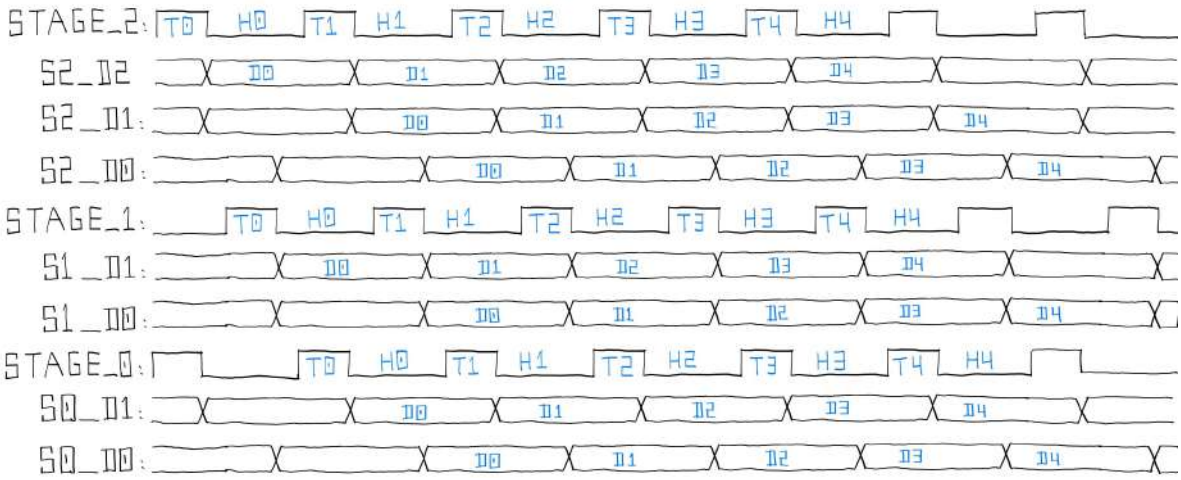


Figure 1.2: *Time Diagram of the Last 3 Stages and Data Alignment Shift-Registers*

The fourth code **11** is unused and serves as redundancy, which enables digital error correction and makes the architecture tolerant to comparator offsets and nonidealities.

Stage Operation

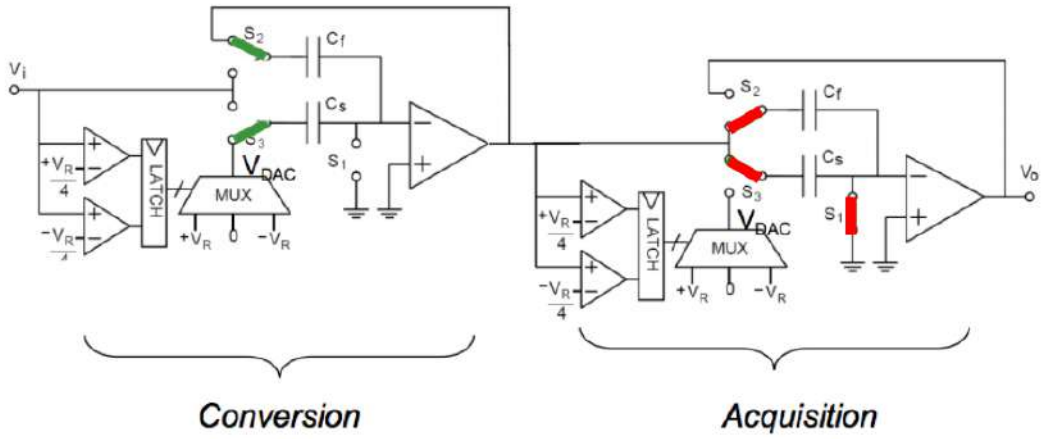


Figure 1.3: *Stage Operation*

Each stage operates in two non-overlapping clock phases: **acquisition** and **conversion**.

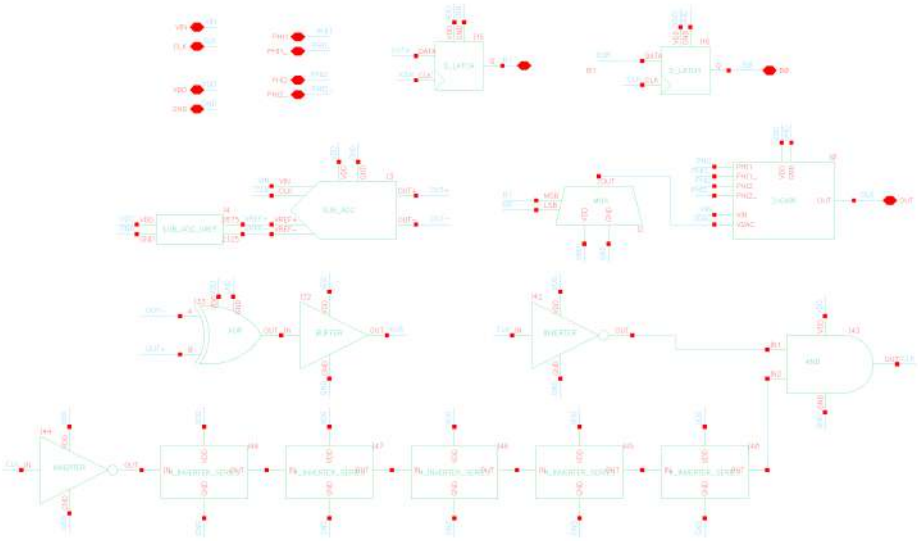


Figure 1.4: *1.5 Bit Stage circuit*

In the top of schematics we have D-Latches to hold the bits output help high so that it can be read by SIPO. The we have sub ADC figure: 1.6 which give the output based on previous stage, In the sub-adc we are using dynamic latch comparator to kep the operation quick and we sized the dynamic latch comparator to have a quick operation 1.5. The output of the add goes to the mux Figure: 1.7 which gives 3 voltage level based on adc output value, we are using XOR logic to convert thermometer code to binary code and to control the switches.

MDAC converts digital output to analog for the next stage with the gain of 2. 1.8, MDAC uses a Op-Amp to boost the signal back for the next stage.

In the bottom row of the fig: 1.4 we used a delayed clock to remove the out ripple from the bits which are getting carried forwarded to SIPO.

Phase 1: Acquisition

During the acquisition phase:

- Switches S_1, S_2, and S_3 are closed.
- The sampling capacitor C_s acquires the input voltage V_i.
- A 2-level sub-ADC (based on two comparators) evaluates V_i against thresholds at $\pm V_R4$ and produces a 2-bit digital output D_1, D_0.

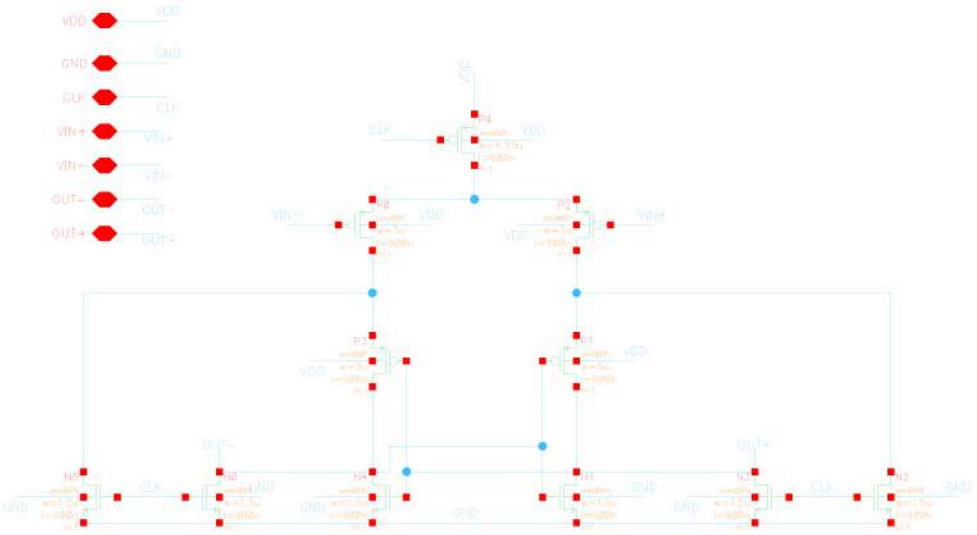


Figure 1.5: *Dynamic Latch Comparator*

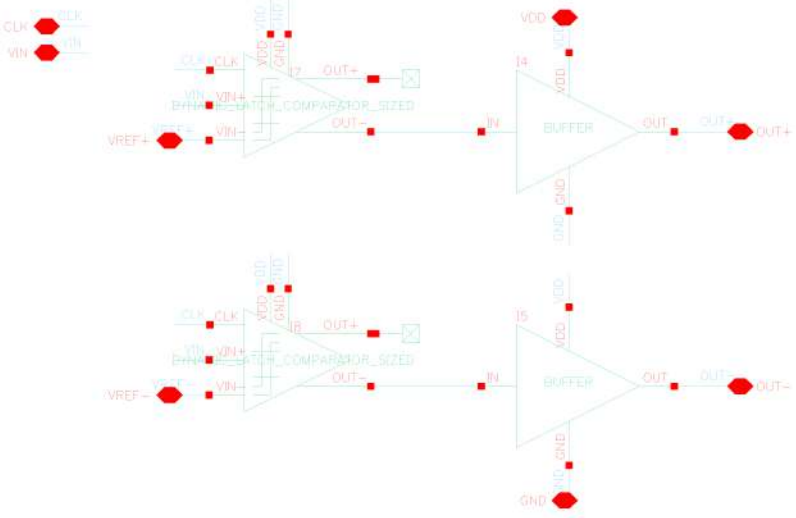


Figure 1.6: *Sub ADC*

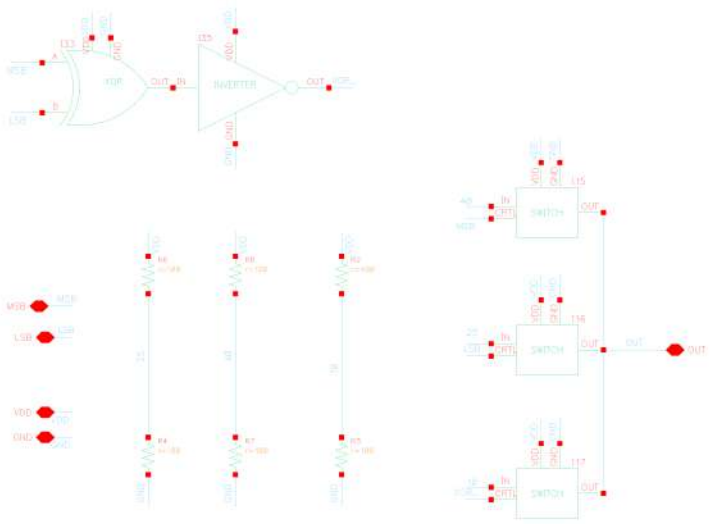


Figure 1.7: *MUX*

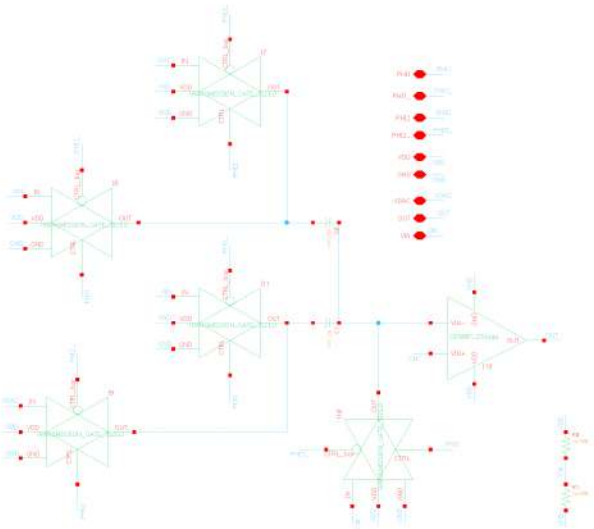


Figure 1.8: *MDAC*

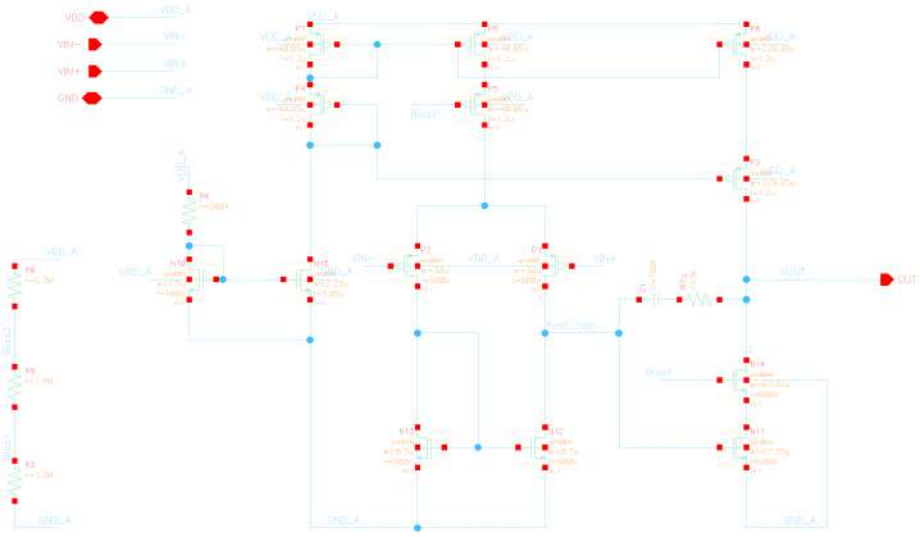


Figure 1.9: *OP AMP*

- This output is latched and sent to the DAC to generate a corresponding analog voltage V_{DAC}.

Phase 2: Conversion

During the conversion phase:

- Switches S_2 and S_3 open to form the amplifier's feedback network.
- The amplifier performs subtraction and amplification:

$$V_o = 2 \cdot V_i - V_{\text{DAC}}$$

- The output V_o becomes the residue that is passed to the next pipeline stage.

Functional Components

Sub-ADC Two comparators determine which of the three ranges the input voltage lies in and output the corresponding code D_1, D_0. For this the reference voltages are $2.875V$ and $2.125V$.

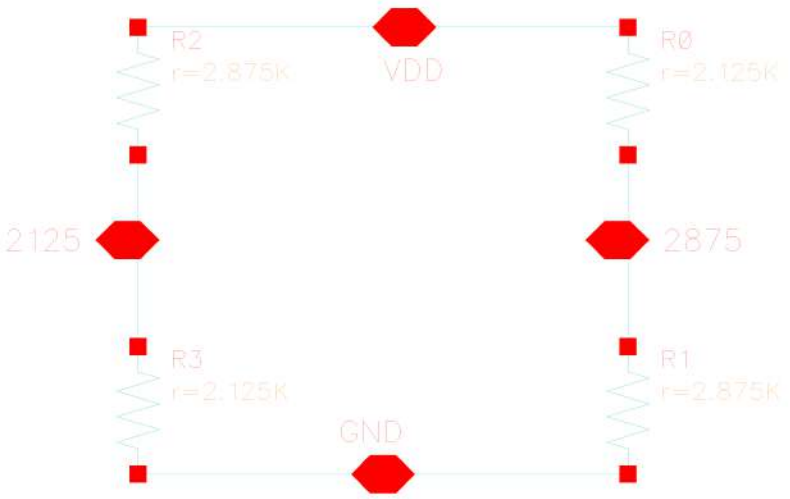


Figure 1.10: *Vref Resistive Divider*

DAC The digital output is converted back to one of three analog levels: V_R (4V), 0 (2.5V), or $-V_R$ (1V), via a simple 3-level DAC implemented with a multiplexer. (XOR operation)

Residue Amplifier The amplifier, with a gain of 2, subtracts V_{DAC} from the sampled input and amplifies the residue for the next stage:

$$V_{\text{residue}} = 2 \cdot V_i - V_{\text{DAC}}$$

Pipelining Concept

Multiple such 1.5-bit stages are cascaded. Each stage performs:

1. Sub-ADC operation to produce rough bits.
2. DAC correction and subtraction.
3. Residue amplification.

The final digital output is obtained by digitally correcting and combining the outputs of all stages.

1.3 Description of Components for Testing

1.3.1 Ideal DAC

The ideal dac used in test-bench shown in Figure 3.1 is implemented using VerilogA. The VerilogA code is below.

```
// VerilogA for ESE6680_PJ3, 10BIT_IDEAL_DAC, veriloga
`include "constants.vams"
`include "disciplines.vams"
module IDEAL_DAC_10BIT(IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8,
   IN9, CLK, OUT);
input IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8, IN9, CLK;
output OUT;
electrical IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8, IN9, CLK;
electrical OUT;
parameter real Analog_Reference_Voltage = 4;
parameter real Digital_Reference_Voltage = 2.5;
integer Din;
analog function integer DIN;
      input IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8, IN9;
      real IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8, IN9;
      begin
            DIN = 0;
            DIN = DIN + (IN0 > Digital_Reference_Voltage ? 1 << 0 :
               0);
            DIN = DIN + (IN1 > Digital_Reference_Voltage ? 1 << 1 :
               0);
            DIN = DIN + (IN2 > Digital_Reference_Voltage ? 1 << 2 :
               0);
            DIN = DIN + (IN3 > Digital_Reference_Voltage ? 1 << 3 :
               0);
            DIN = DIN + (IN4 > Digital_Reference_Voltage ? 1 << 4 :
               0);
```

```
            DIN = DIN + (IN5 > Digital_Reference_Voltage ? 1 << 5 :
               0);
            DIN = DIN + (IN6 > Digital_Reference_Voltage ? 1 << 6 :
               0);
            DIN = DIN + (IN7 > Digital_Reference_Voltage ? 1 << 7 :
               0);
            DIN = DIN + (IN8 > Digital_Reference_Voltage ? 1 << 8 :
               0);
            DIN = DIN + (IN9 > Digital_Reference_Voltage ? 1 << 9 :
               0);
      end
endfunction
analog begin
      V(OUT) <+ slew(Analog_Reference_Voltage * (1.0*Din/1023.0));
      @ (initial_step) begin
            Din = DIN(V(IN0), V(IN1), V(IN2), V(IN3), V(IN4), V(IN5)
               , V(IN6), V(IN7), V(IN8), V(IN9));
      end
      @ (cross(V(CLK) - Digital_Reference_Voltage, 1)) begin
            Din = DIN(V(IN0), V(IN1), V(IN2), V(IN3), V(IN4), V(IN5)
               , V(IN6), V(IN7), V(IN8), V(IN9));
            //$display(">>>Din = %d<<<", Din);
      end
      @ (final_step) begin
            $display(">>>Din = %d<<<", Din);
      end
end
endmodule
```

1.4 Description of Subcompoents for 1-Bit Stage

1.4.1 Track and Hold Circuit

As shown in Figure 2.3. The track and hold circuit used is the same in Project 2. It is a bottom-plate sampling track-and-hold circuit using charge redistribution.

1.4.2 Comparator

Shown in Figure 2.6, the comparator used is also the same in Project 2. It is a dynamic latch comparator. The dynamic latch is used because: 1) consumes zero static power, 2) it directly produces rail-to-rail outputs, and 3) its input-referred offset arises from primarily one differential pair mismatch. The comparator will compare the voltage of VIN+ and VIN- when CLK is low.

1.4.3 DAC

This is only a one-bit DAC with two possible output voltages. It is implemented using an MUX.

1.4.4 Gain Stage

As shown in Figure 2.7, the gain stage is very similar to the track and hold circuit except the positive input of the opamp is now connected to IN- instead of a fixed bias voltage. $C_1 = 2C_0$ is used to obtain a gain of 2. As you may noticed, this circuit can't perform the subtraction function of IN - IN-. Indeed, it can't, so we played some tricks on the DAC, which will be explained in the next section.

1.4.5 A Single Stage

Figure 2.5 shows the circuit of a single stage and Figure 1.11 shows the timing of a single stage. The comparator is driven by a slightly delayed clock to allow the d flip-flop and the gain stage to hold the correct value.

As mentioned above, the output of the gain stage is not $2IN - IN-$, instead, it is $2IN - V_{off}$, with V_{off} determined by IN-. After a lot of simulation and variable analysis, we decide to use $V_1 = 1.3703$ V and $V_0 = 2.3385$ V for the DAC output.

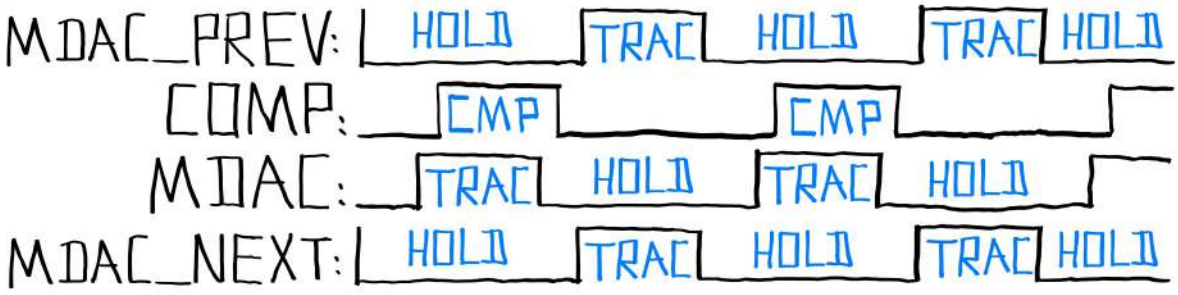


Figure 1.11: *Time Diagram of a Single Stage*

1.5 Hand Calculations and Analysis Leading to the Design

1.5.1 Target Specifications

We aim to design an 8-bit pipeline ADC with:

- Sampling frequency $f_s = 2$ MHz
- Input bandwidth = DC to 1 MHz
- Input range: 0 V to 4 V (single-ended)
- SNDR > 40 dB
- DNL/INL < 1 LSB
- Energy-efficient operation

1.5.2 Comparison of 1.5-bit and 1-bit per Stage Architectures

We began with an analysis of the 1.5-bit-per-stage architecture, which is commonly used in pipeline ADCs due to its redundancy and tolerance to comparator offset errors. A 1.5-bit stage resolves 3 output levels and passes the residue forward, allowing for digital correction of small errors between stages. The key benefit of this approach is that each comparator only needs to be accurate within ± 0.25 LSB , and errors can be corrected digitally using overlapping bits.

To resolve 8 bits with 1.5-bit stages: $(8 + 1)/1 = 9$

(The extra bit allows for overlap and digital correction.)

However, during simulation, we observed bit-flipping near decision boundaries, especially where metastability or mismatch affected comparator behavior. These artifacts were harder to isolate and correct under our time constraints. Therefore, we opted

to implement the final design using 1-bit-per-stage, whichwhile requiring more analog precision and more stagesoffered:

- Cleaner bit alignment and residue scaling
- Simpler comparator logic (single threshold)
- Easier debugging and verification

1.5.3 1-bit Stage Operation and Scaling

Each 1-bit stage includes:

- A comparator that compares V_{in} to a threshold (V_{ref})
- A 2-level DAC (outputs 0 or V_{ref})
- A residue amplifier with gain of 2

Residue calculation:

$$V_{\text{res}} = V_{\text{in}} - D_i \cdot V_{\text{ref}}, \quad V_{\text{next}} = 2 \cdot V_{\text{res}}$$

To resolve 8 bits using 1-bit per stage:

$$N_{\text{stages}} = \frac{8}{1} = 8$$

1.5.4 LSB and Comparator Offset Tolerance

For a full-scale range of 4V:

$$\text{LSB} = \frac{4\ \text{V}}{2^8} = 15.625\ \text{mV}$$

Allowing comparator offset $\leq \frac{1}{4}$ LSB:

$$V_{\text{offset}}^{\text{max}} \approx 3.91\ \text{mV}$$

1.5.5 Capacitor Sizing Based on Thermal Noise

Thermal noise (from kT/C) should be:

$$v_{\text{n, rms}} < \frac{1}{4} \cdot \text{LSB} \approx 3.91\ \text{mV}$$

$$C \geq \frac{kT}{v_{\mathrm{n}}^2} \approx \frac{1.38 \times 10^{-23} \cdot 300}{3.91 \times 10^{-32}} \approx 2.7 \text{ fF}$$

In practice, we used capacitor sizes of $\geq$ 50 fF to tolerate mismatch and ensure settling.

1.5.6 SNDR, ENOB, and FOM

Ideal SNDR:

$$\mathrm{SNDR}_{\mathrm{ideal}} = 6.02 \cdot 8 \ \ 1.76 = 49.92 \text{ dB} \tag{1.1}$$

Measured SNDR:

$$\mathrm{SNDR} = 43.38 \text{ dB}, \quad \mathrm{ENOB} = \frac{43.38 - 1.76}{6.02} \approx 6.91$$

Figure of Merit:

$$\mathrm{FOM} = \frac{\mathrm{Power}}{f_s \cdot 2^8}$$

Where power is measured in simulations (Section **??**).

1.5.7 Stage Timing and OTA Bandwidth

Each stage must settle within half a clock cycle:

$$T_{\mathrm{settle}} = \frac{1}{2f_s} = 250 \text{ ns}$$

$$\tau \leq \frac{250 \text{ ns}}{\ln 100} \approx 54.3 \text{ ns} \Rightarrow \mathrm{Bandwidth} \geq \frac{1}{2\pi\tau} \approx 2.93 \text{ MHz}$$

This dictated OTA gain-bandwidth design targets.

2

Schematics

2.1 Schematics of ADC of 1-Bit per Stage Design

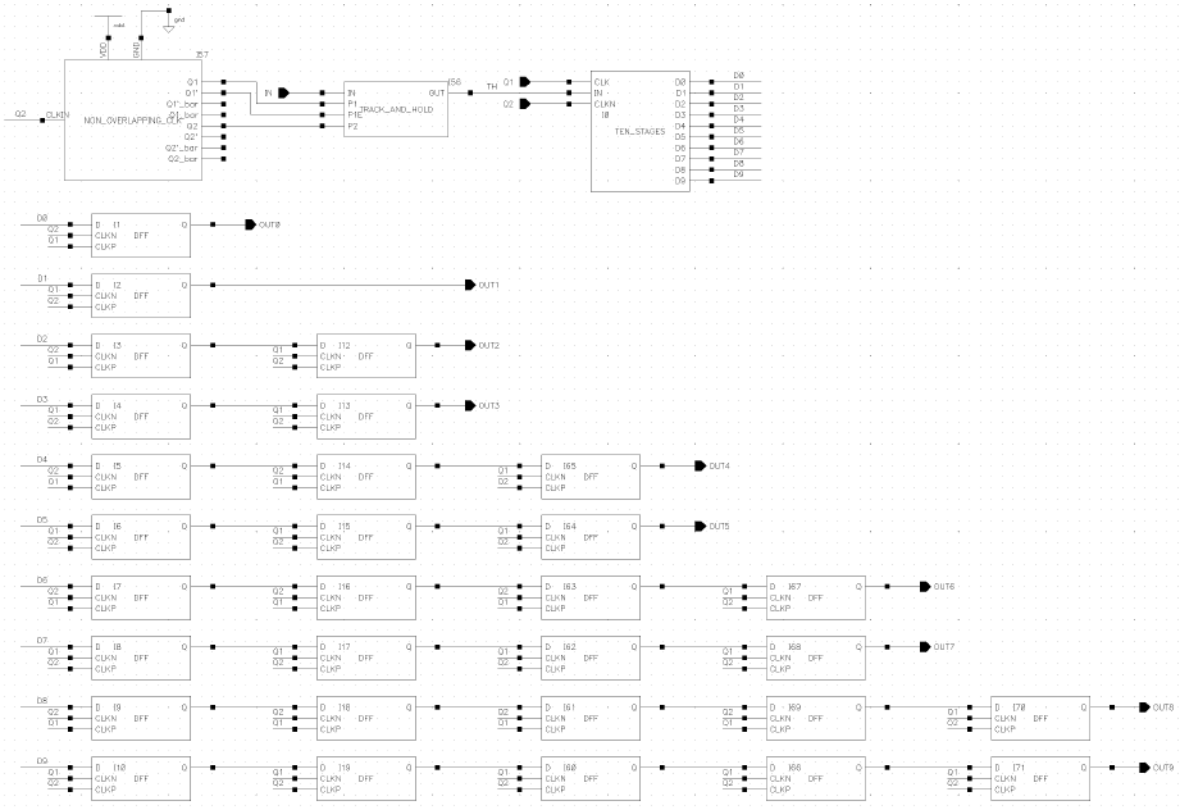


Figure 2.1: *Schematics of ADC*

The schematic of the ADC is shown in Figure 2.1.

2.2 Schematics of Subcomponents of 1-Bit per Stage Design

2.2.1 Schematics of Non-Overlapping Clock Generator

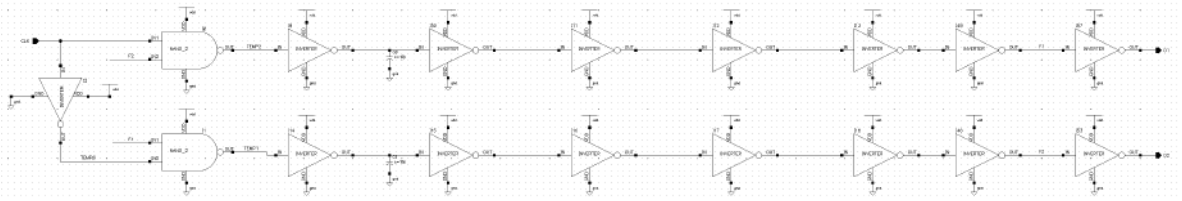


Figure 2.2: *Schematics of Non-Overlapping Clock Generator*

The schematic of the non-overlapping clock generator is shown in Figure 2.2.

2.2.2 Schematics of Track and Hold Circuit

The schematic of the non-overlapping clock generator is shown in Figure 2.3.

2.2.3 Schematic of 10 Stages

The schematic of 10 stages cascaded together is shown in Figure 2.4.

2.2.4 Schematic of a Single Stage

The schematic of a single stage is shown in Figure 2.5.

2.2.5 Schematic of Comparator

The schematic of the comparator is shown in Figure 2.6.

2.2.6 Schematic of Gain Stage

The schematic of the gain stage is shown in Figure 2.7.

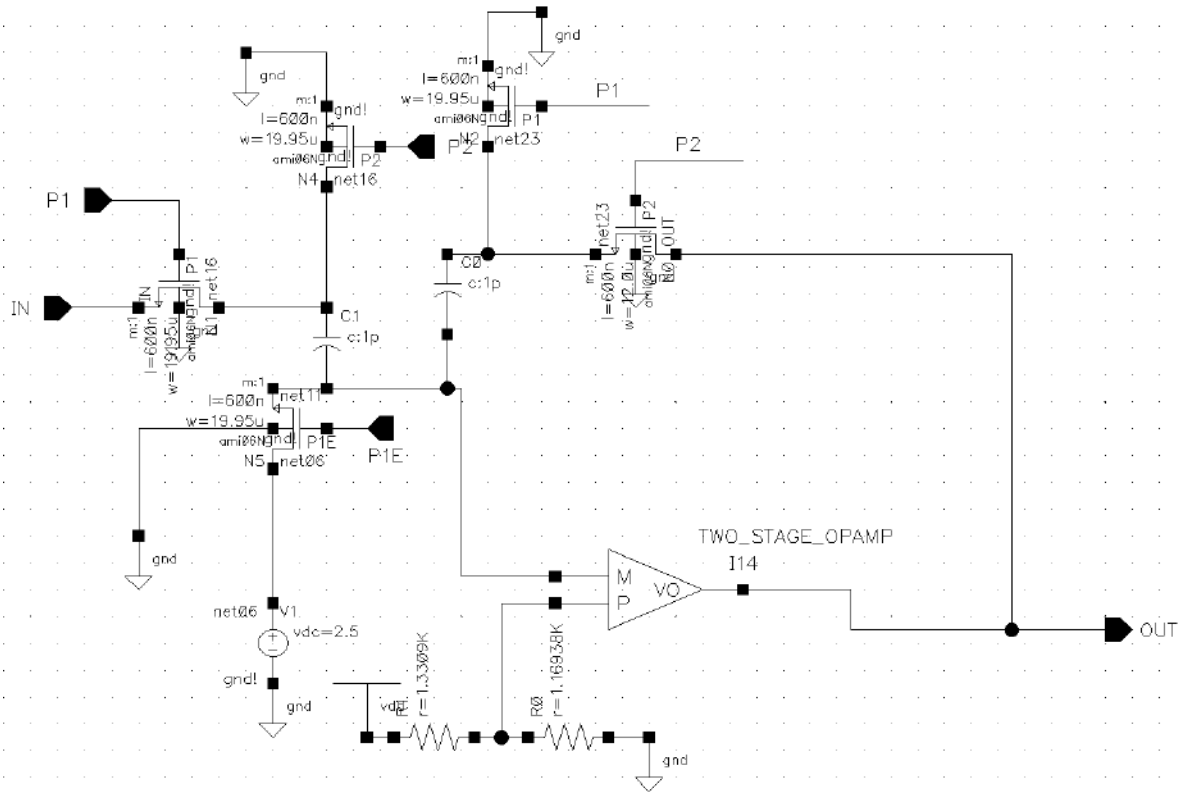


Figure 2.3: *Schematics of Track and Hold Circuit*

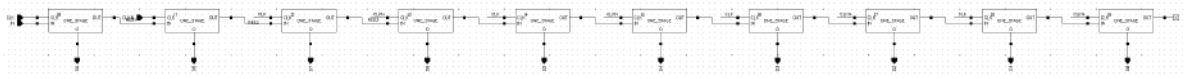


Figure 2.4: *Schematic of 10 Stages*

2.3 Schematics of Subcomponents of 1.5-Bit per Stage Design

2.3.1 Bootstrap Sample and Hold

The schematic shown in figure 2.8 is bootstrap implementation of which is used in 1.5 bit adc for data input.

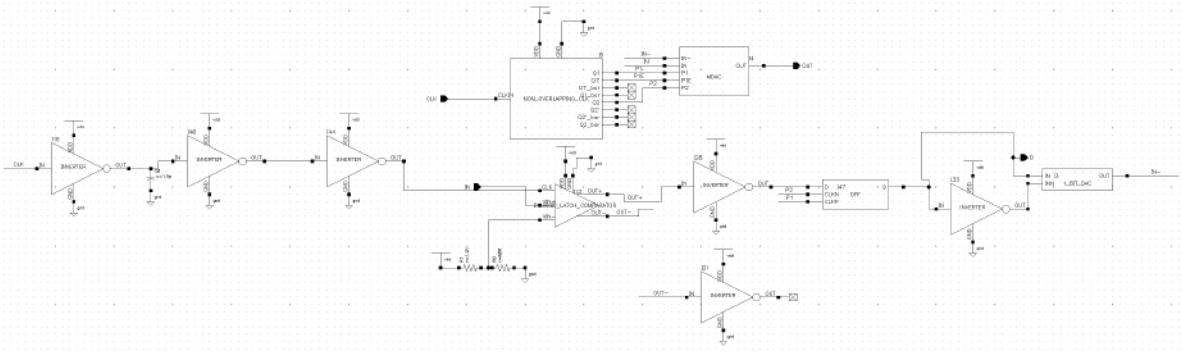


Figure 2.5: *Schematic of a Single Stage*

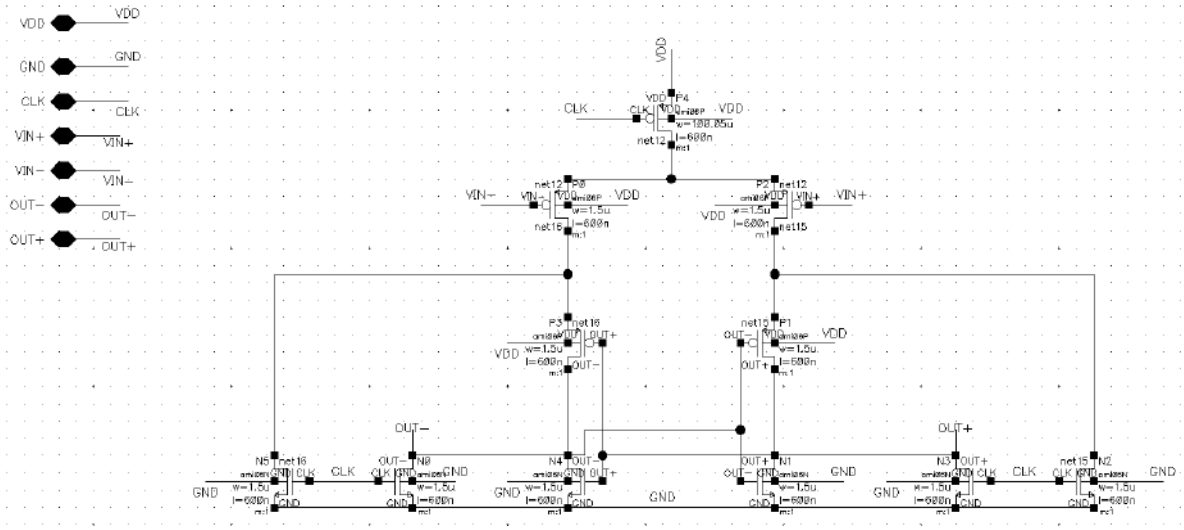


Figure 2.6: *Schematic of Comparator*

2.3.2 1.5 Bit cascaded stages

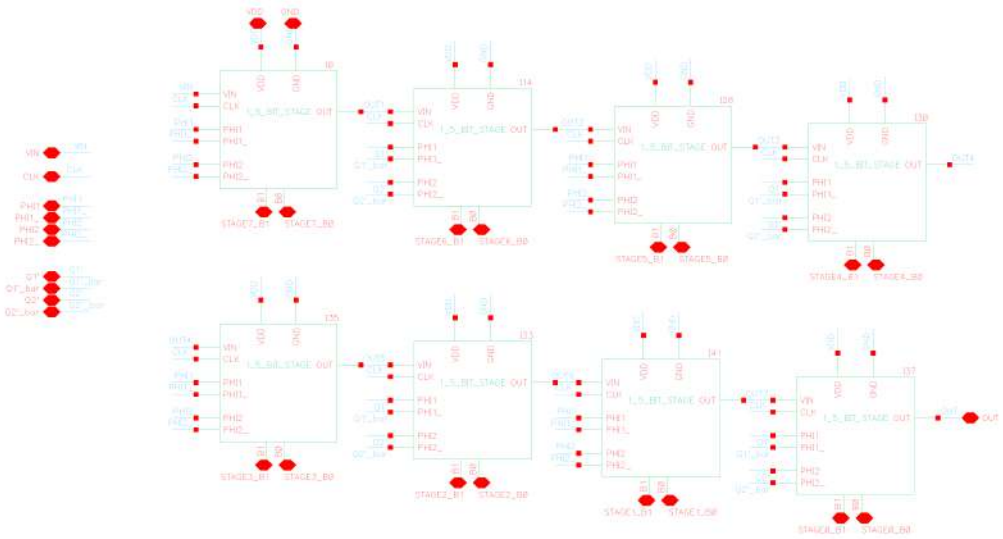


Figure 2.9: *8 Cascaded stages of 1.5 bit*

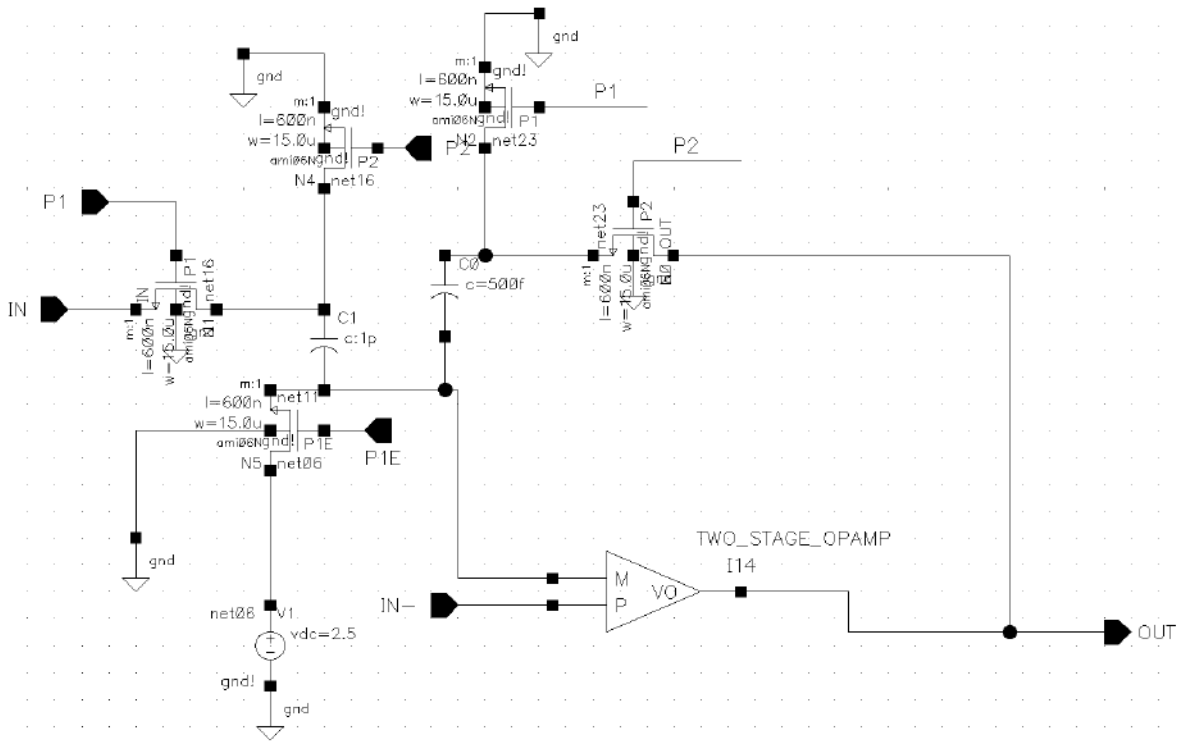


Figure 2.7: *Schematic of Gain Stage*

The schematic in figure 2.9 is the cascaded 8 stage for every bits with 0.5 redundant bit.

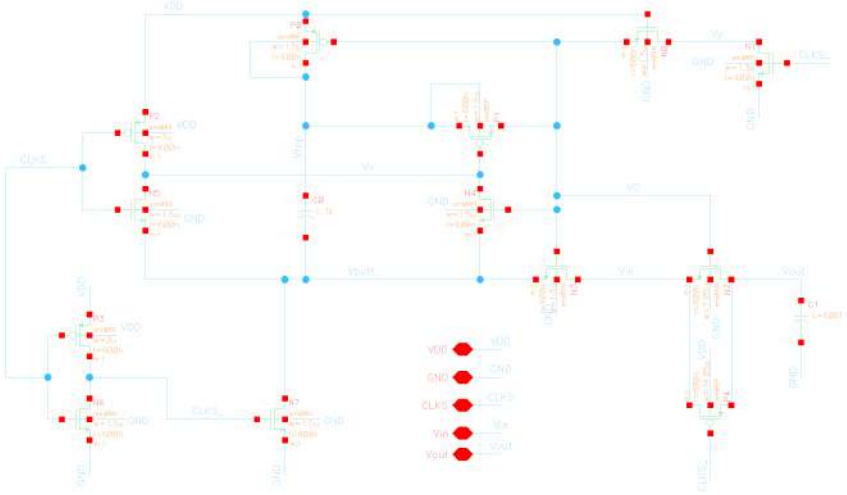


Figure 2.8: *Sample and Hold Bootstrap*

2.3.3 SIPO- Serial in Parallel Out

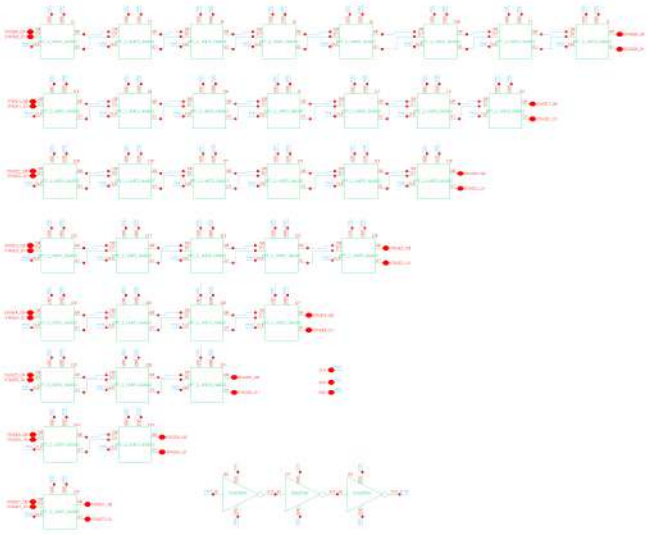


Figure 2.10: *SIPO*

The schematic in figure 2.10 is used to shift the bits of the ADC for the adder.

Figure 2.11: *Adder Schematic*

2.3.4 Adder

The Adder is used to sum and carry the bits of all the stages so that it can be sent to DAC. 2.11

The AND, OR and XOR implementation of individual adder element is shown in figure 2.12

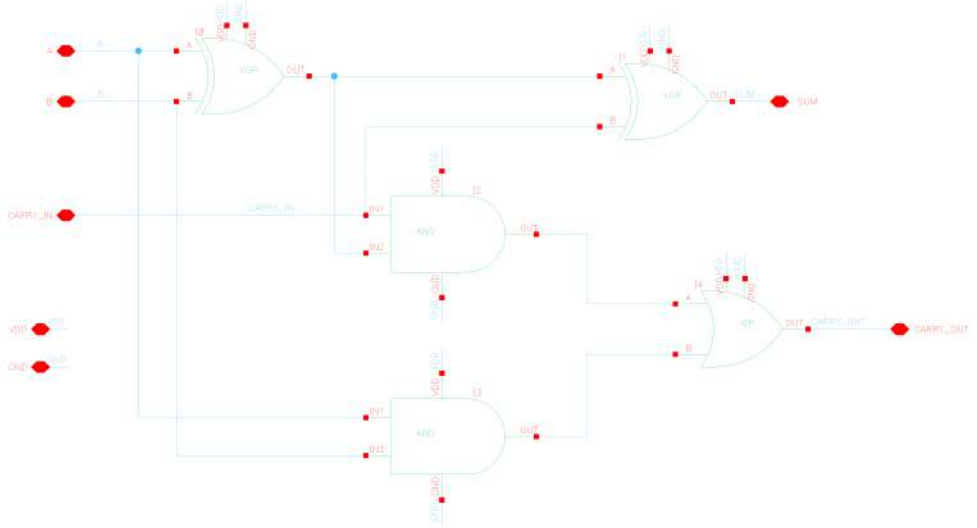


Figure 2.12: *Adder Unit Element*

3

Design Validation and Verification

3.1 Simulation Results Showing Functional Operation of ADC of 1-Bit per Stage Design

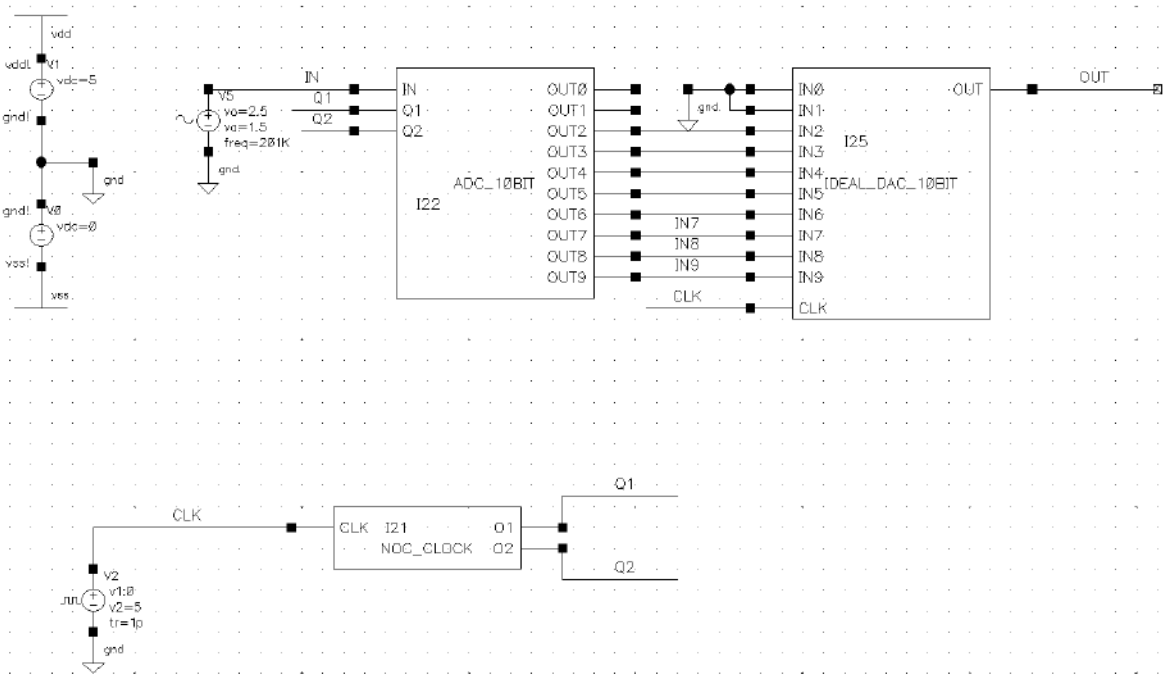


Figure 3.1: *Testbench for Verifying the Function of ADC*

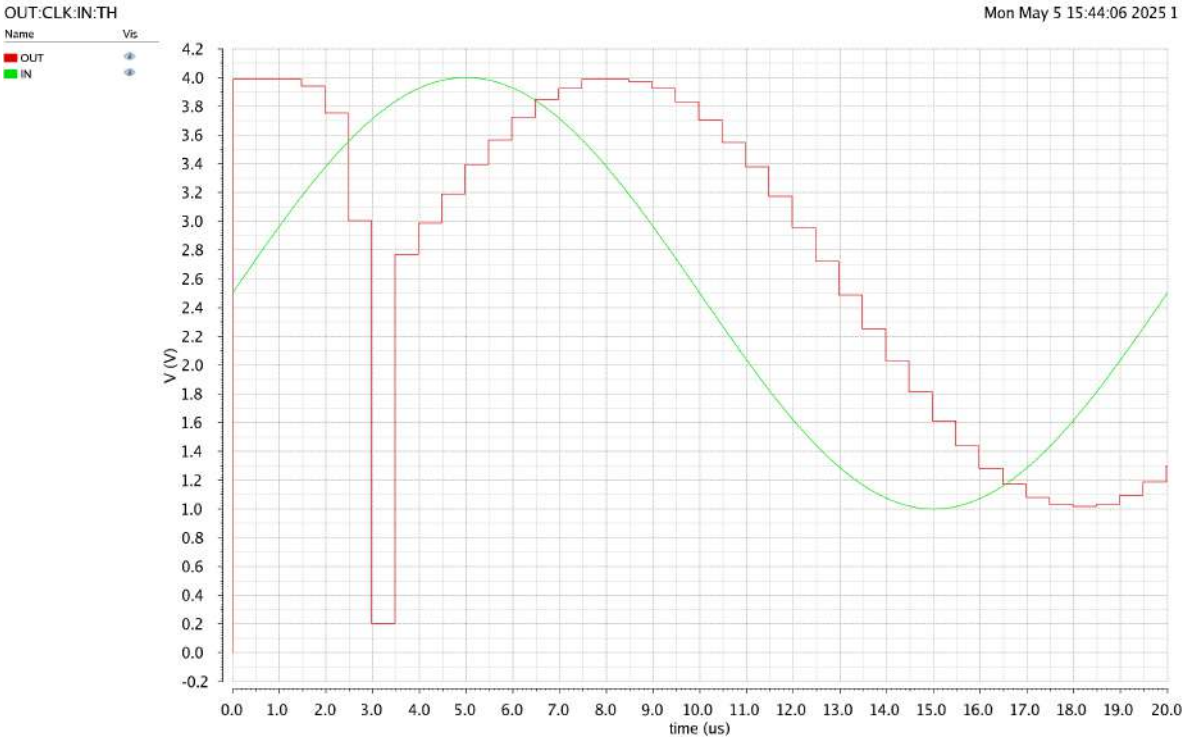


Figure 3.2: *Waveform Showing the ADC Functions Properly*

Figure 3.2 shows the pipeline adc with 1 bit per stage works properly. Figure 3.3 is the testbench setup to get the waveform.

3.2 Simulation Results Showing Functional Operation of Subcomponents in 1-Bit per Stage Design

3.2.1 A Single Stage

Figure 3.4 shows each stage works properly. Figure 3.3 is the testbench setup to get the waveform.

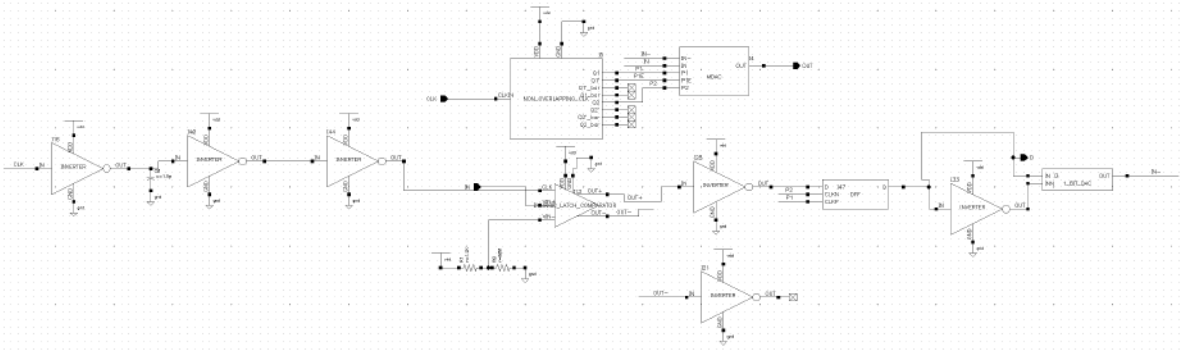


Figure 3.3: *Testbench for Verifying a Single Stage*

3.3 Verification for 1.5-Bit per Stage Design

3.3.1 Sample and Hold Bootstrap

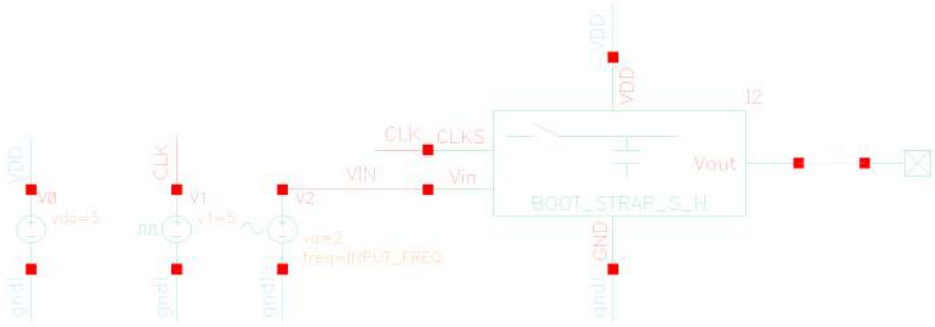


Figure 3.5: *Sample and Hold test*

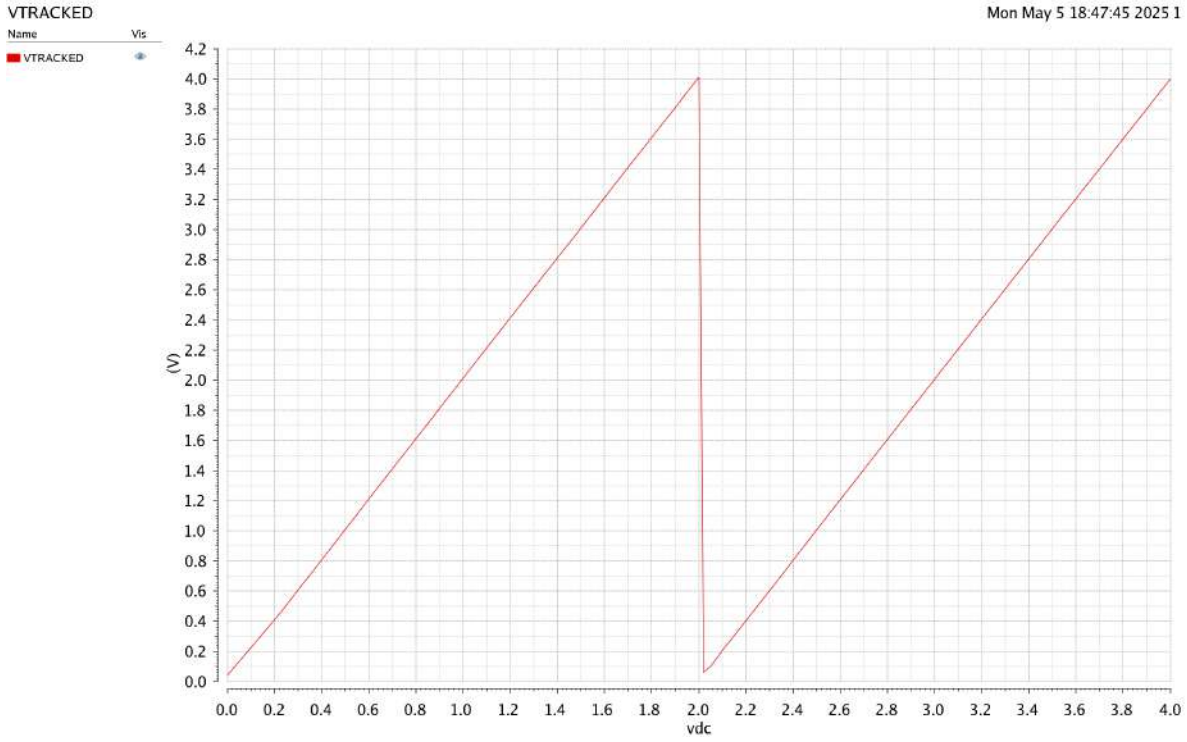


Figure 3.4: *Waveform Showing Each Stage Functions Properly*

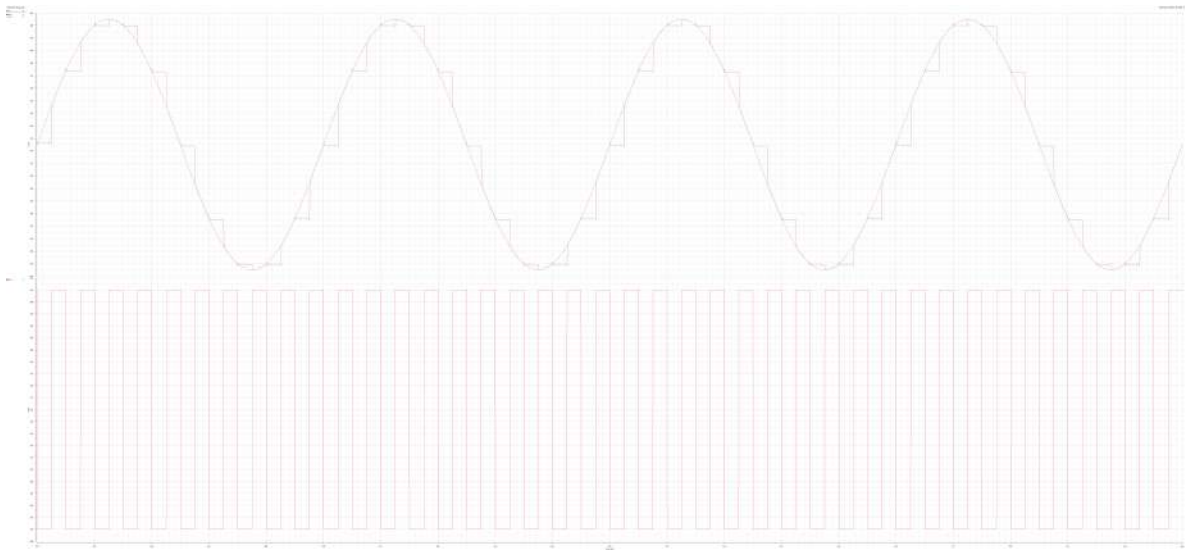


Figure 3.6: *Sample and Hold Result*

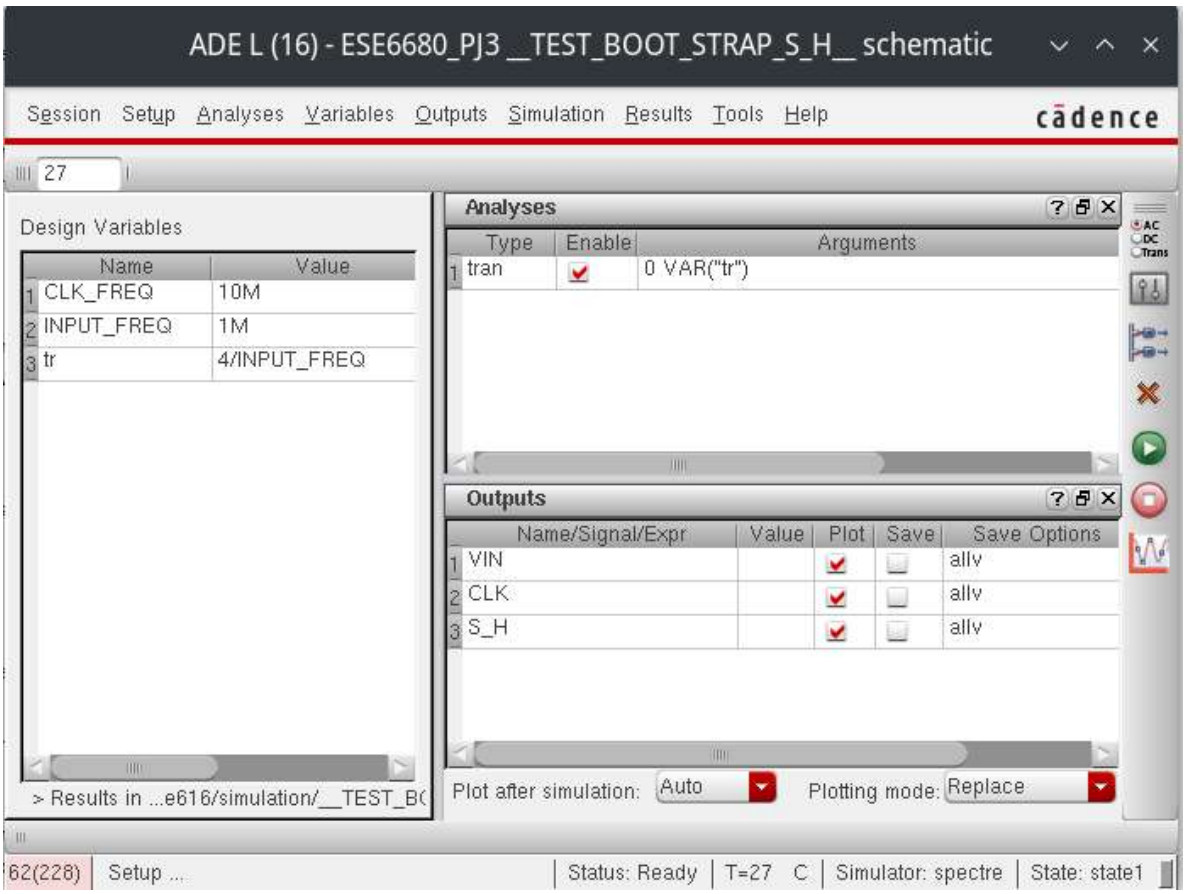


Figure 3.7: *ADL Sample and Hold*

The figure 3.5 shows testbench of bootstrap sample and hold with the result shown in figure 3.6. From the graph we can see the output is tracking the input waveform.

3.3.2 XOR

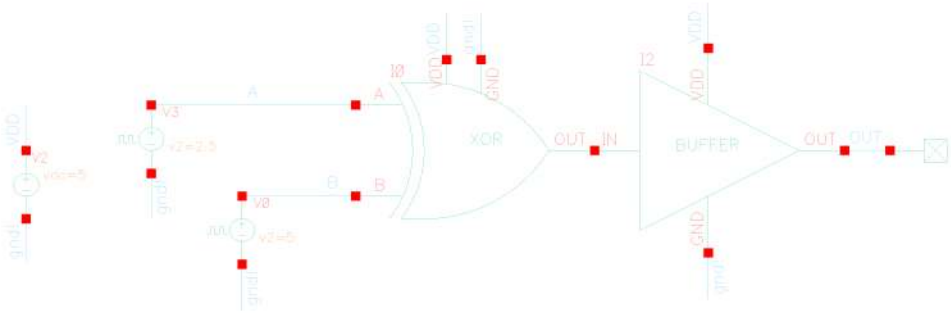


Figure 3.8: *XOR Test*

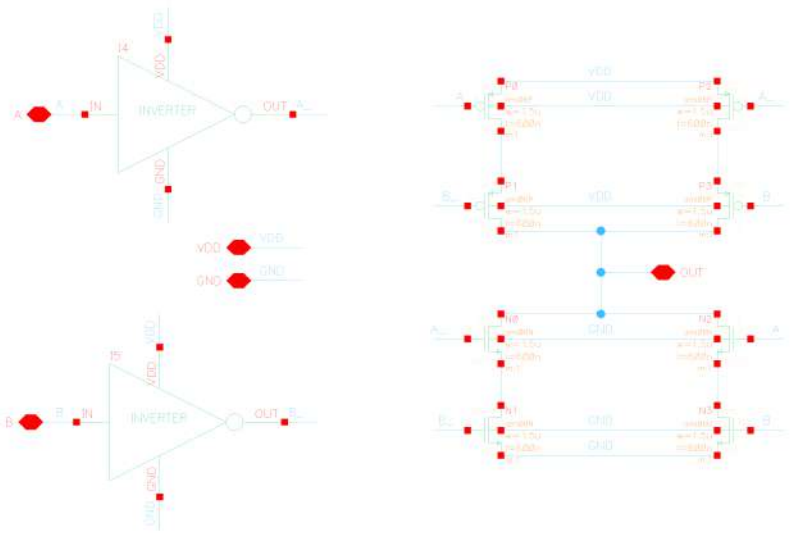


Figure 3.9: *XOR*

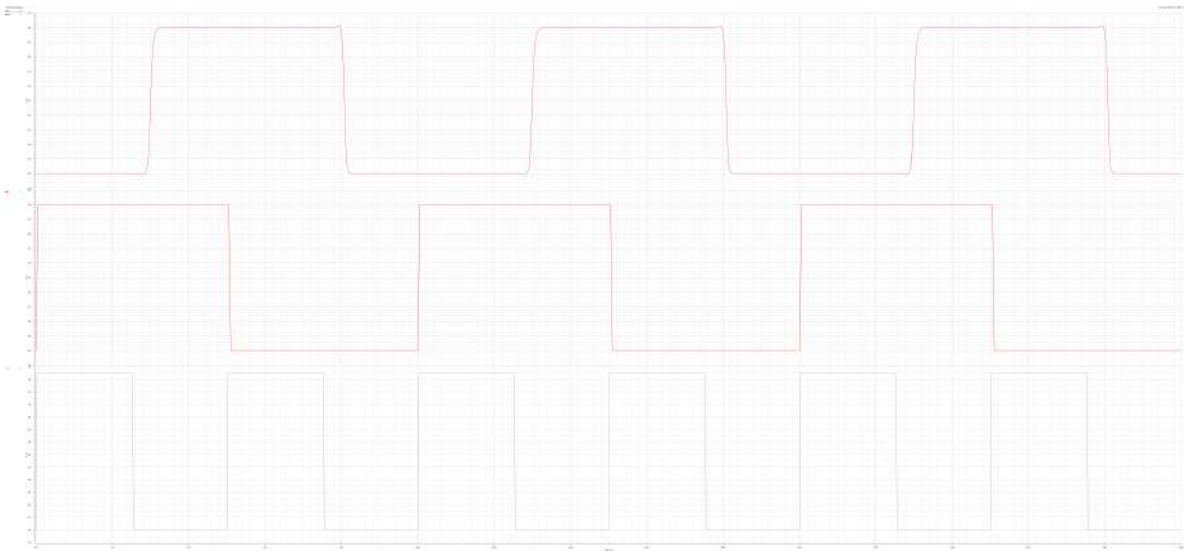


Figure 3.10: *XOR Result*

The functionality of XOR can be verified from the figure 3.10

3.3.3 XNOR

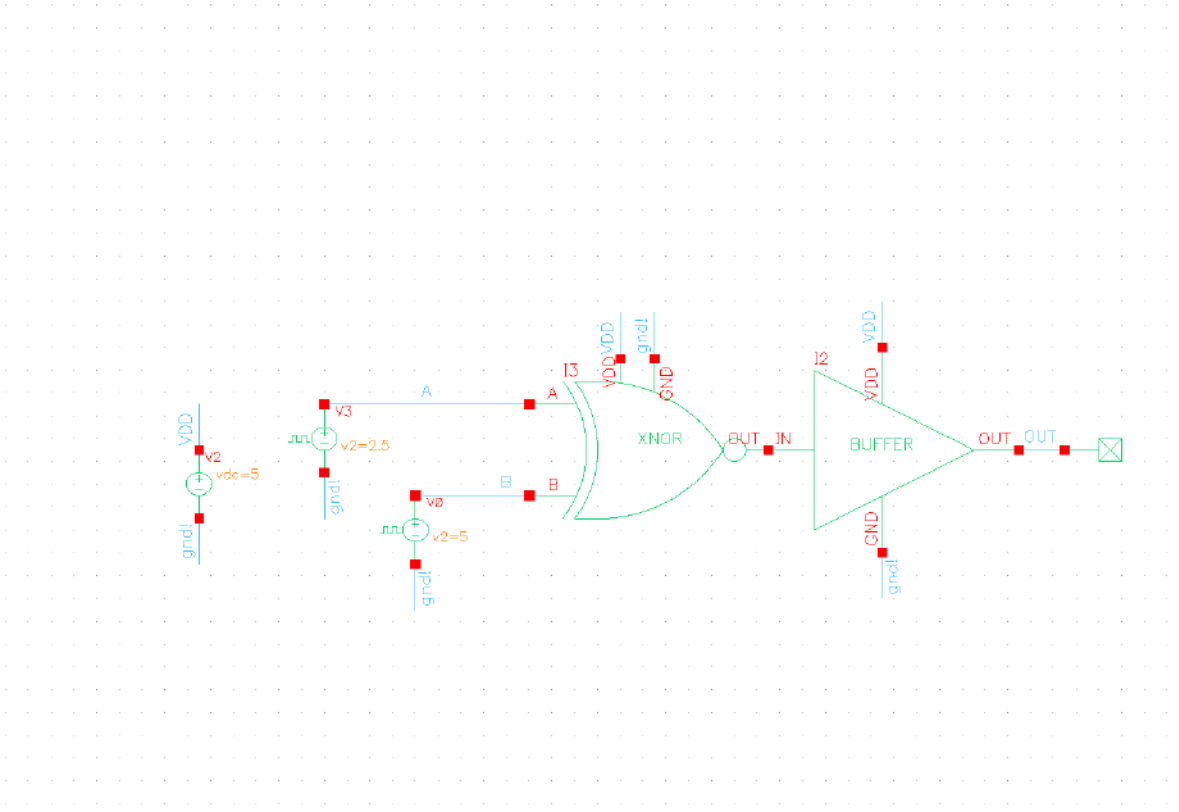


Figure 3.11: *XNOR TEST*

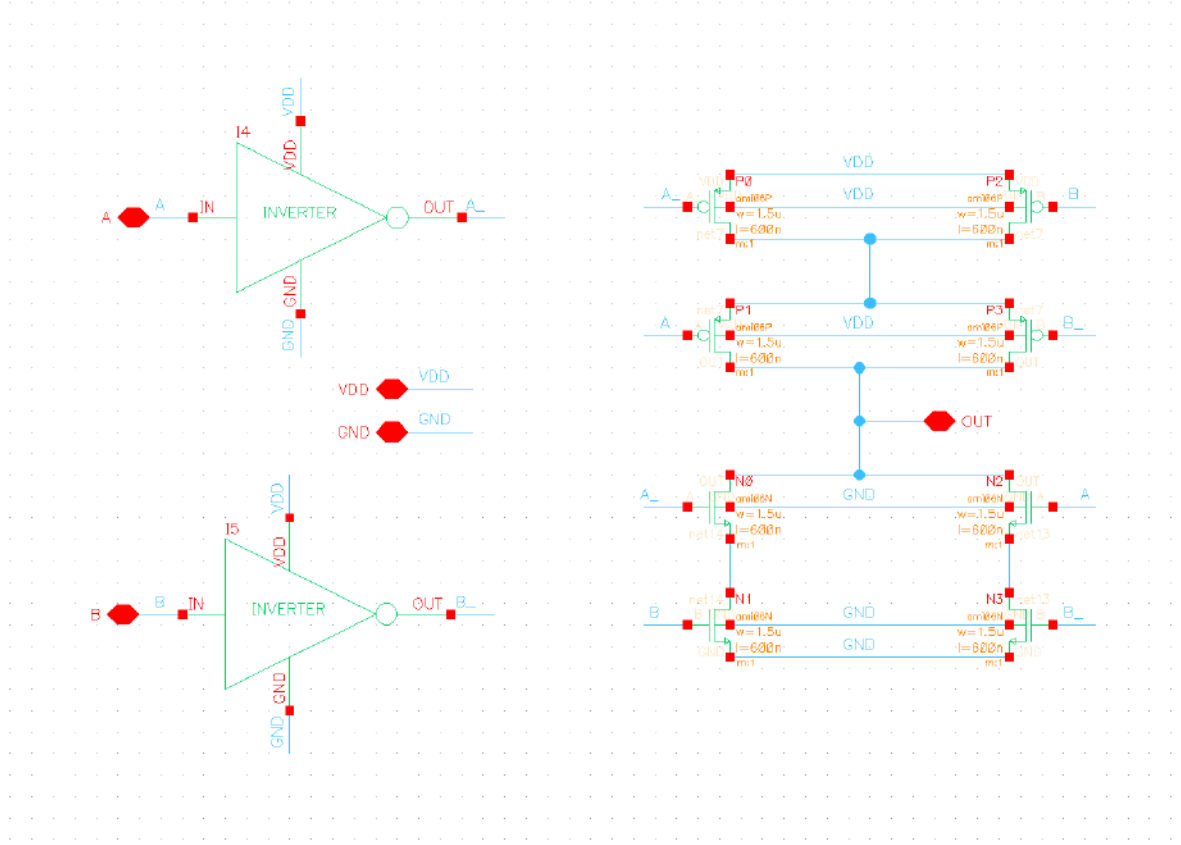


Figure 3.12: *XNOR*

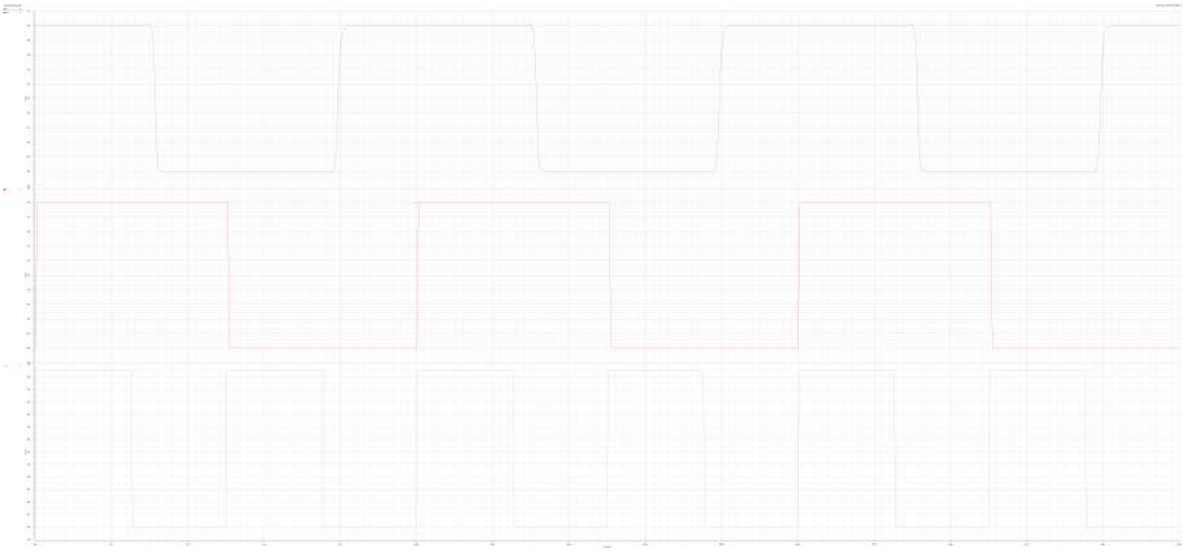


Figure 3.13: *XNOR RESULT*

The functionality of XNOR can be verified from the figure 3.13

3.3.4 SUB ADC

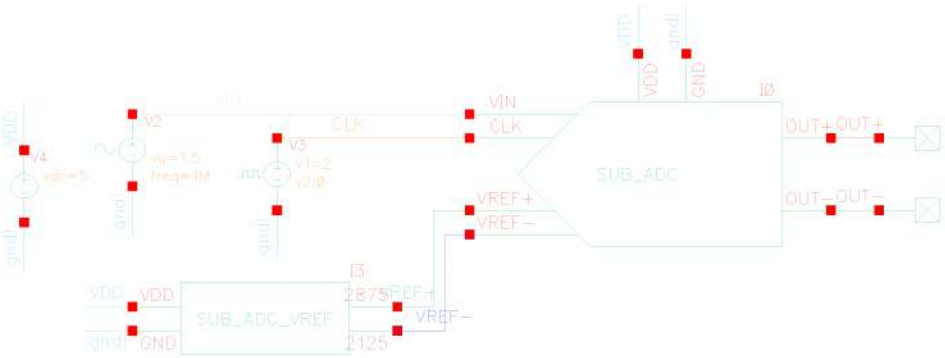


Figure 3.14: *SUB ADC Test*

Figure 3.15: *SUB ADC Result*

From the figure 3.15 the ADC is correctly sampling and giving the output at $2.875V$ and $2.125V$.

3.3.5 MUX

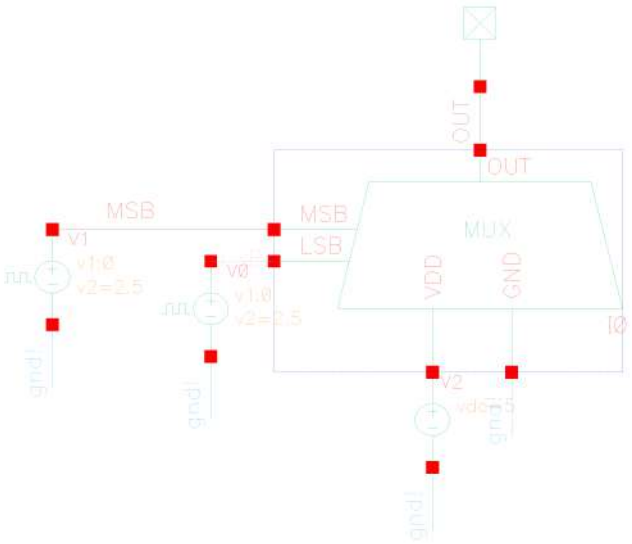


Figure 3.16: *MUX Test*

Figure 3.17: *MUX Result*

From the figure 3.17 we can see the correct functinality of our mux as it used converted binary code from thermometer code to perform the operation $10, 01, 00$ corresponds to $4V, 2.5V, 1V$ respectively.

3.3.6 Full Adder

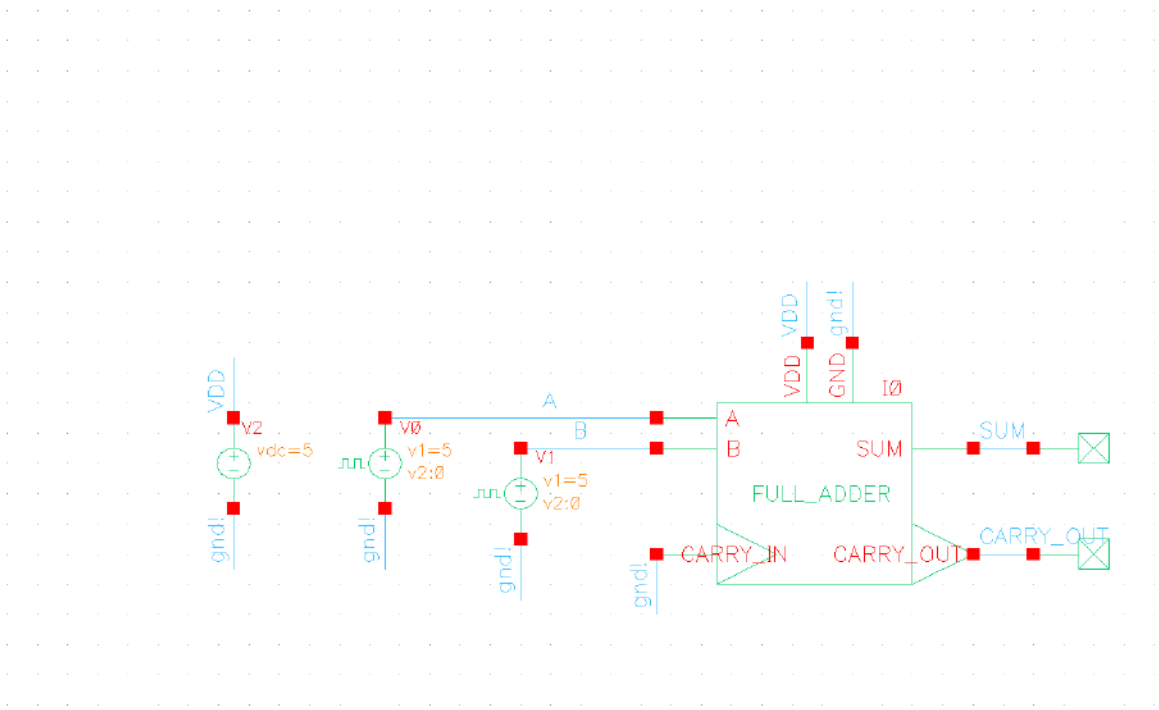


Figure 3.18: *Full Adder Test*

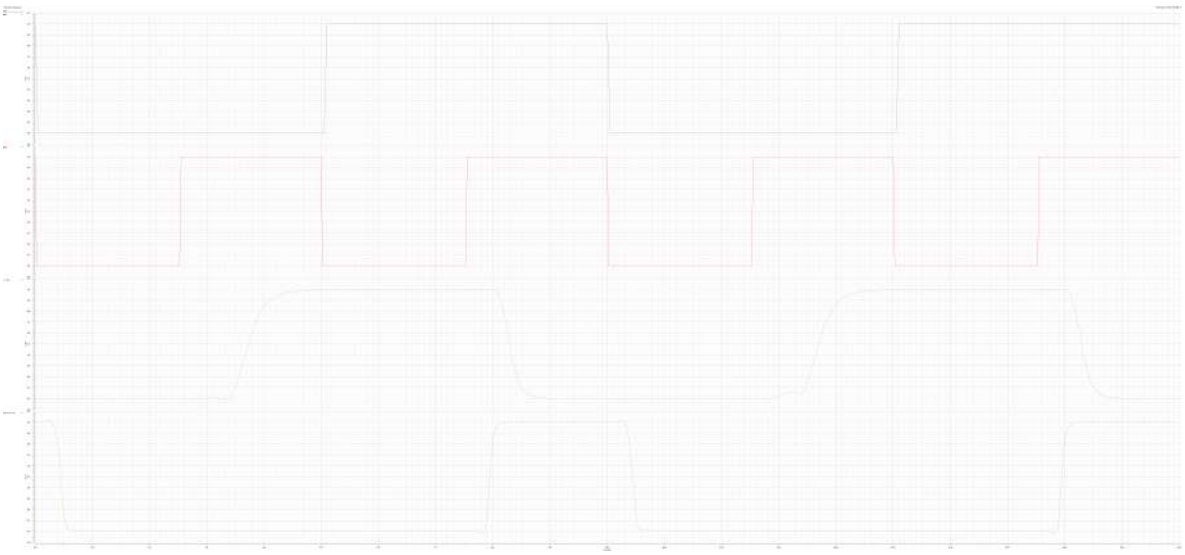


Figure 3.19: *Full Adder Result*

From the figure 3.19 we can verify the correct operation on the adder as the sum and carry for code 0011 and 0101 sum = 0011 and carry = 0 but for 1 and 1 carry = 1 and sum = 0.

3.3.7 D-Latch

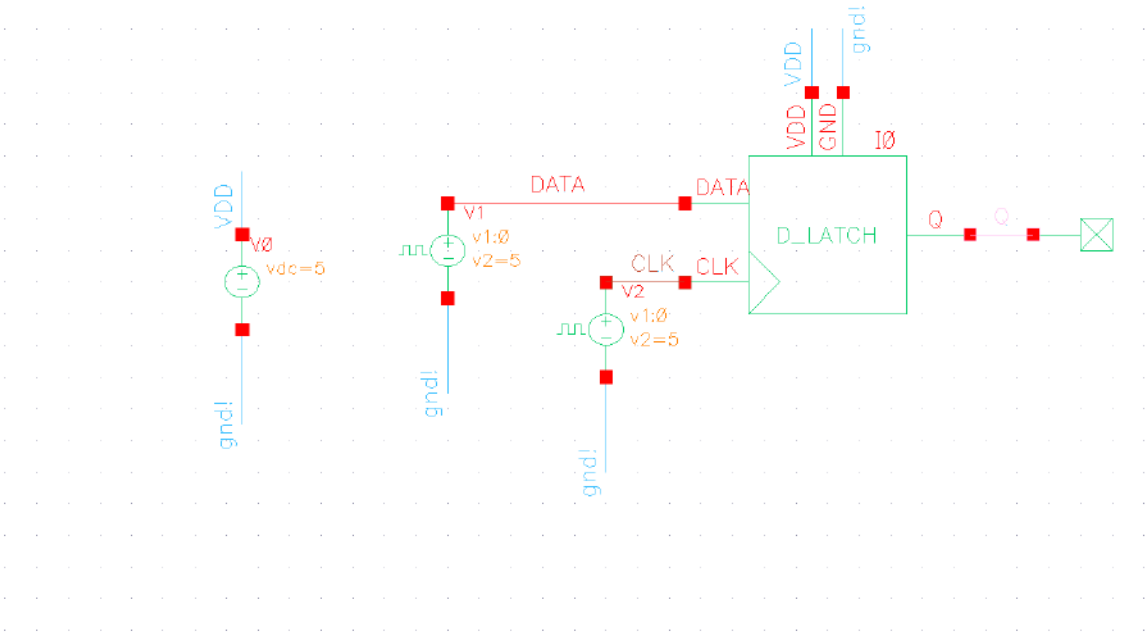


Figure 3.20: *D-Latch test*

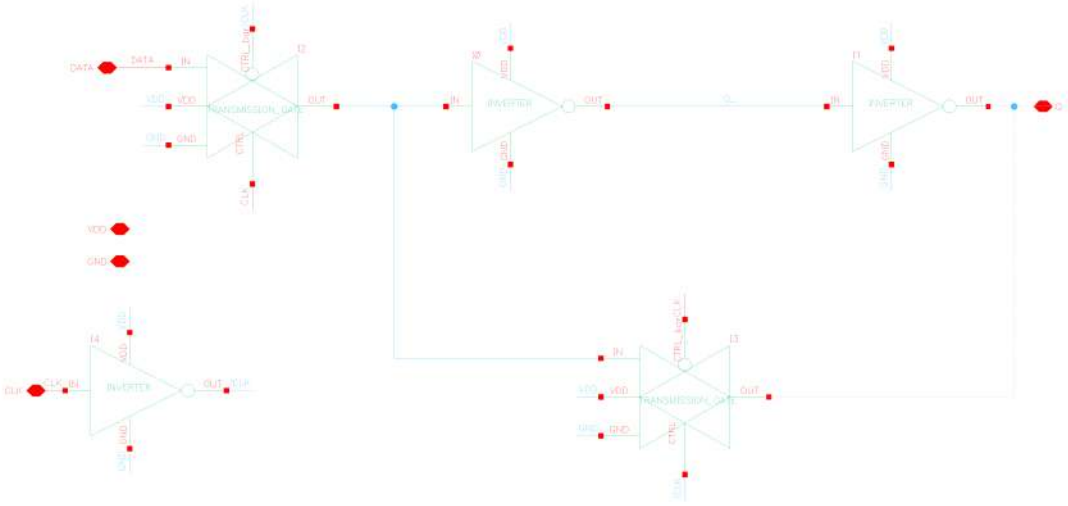


Figure 3.21: *D-Latch*

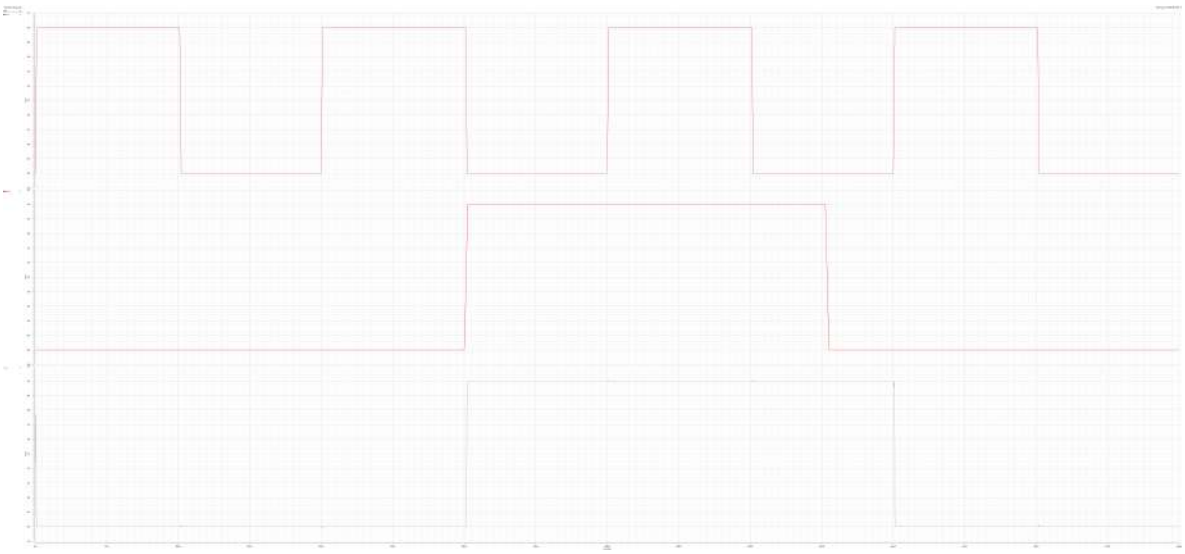


Figure 3.22: *D-Latch Result*

From the figure we can see the D-Latch holds the output bit high until the next sampling period.

3.3.8 Test 1.5 bits stage

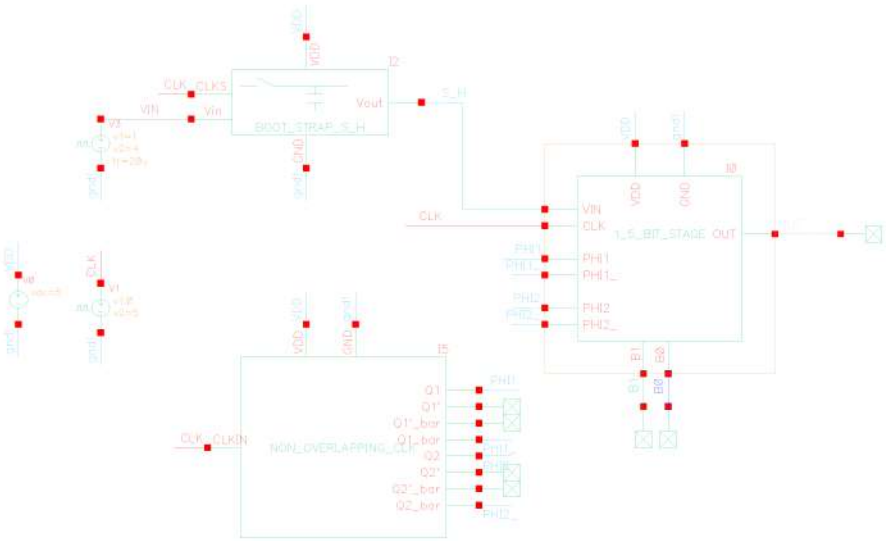


Figure 3.23: *1.5 Bits stage Test*

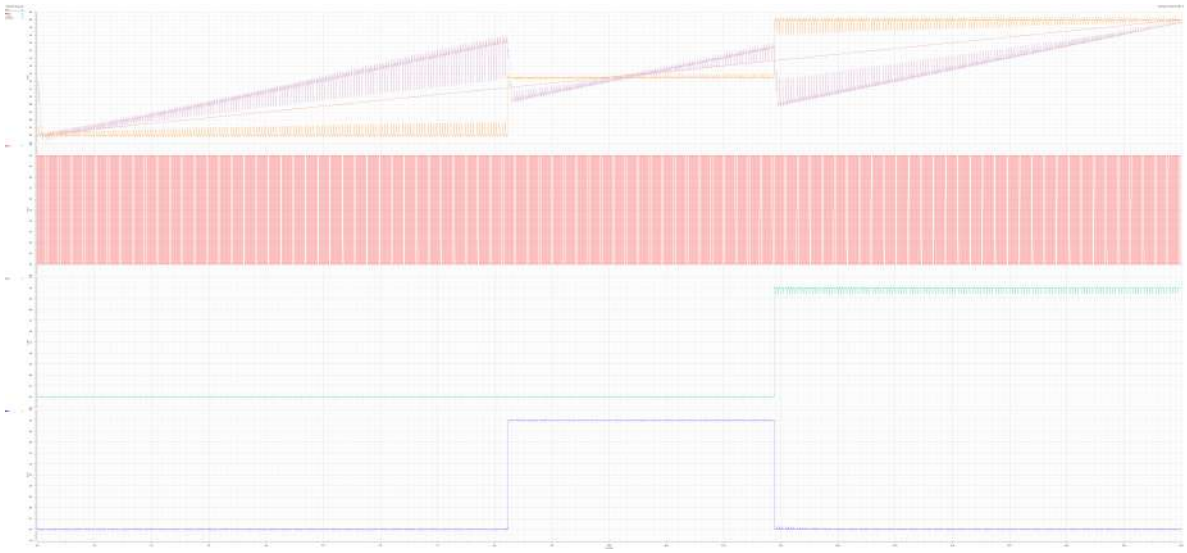


Figure 3.24: *Clock Frequency: 20Mhz and Input Frequency: 1MHz*

We can see that our 1st stage works perfectly and gives the correct code and bits for each voltage levels

3.3.9 8 Cascaded Stages 1.5 bits

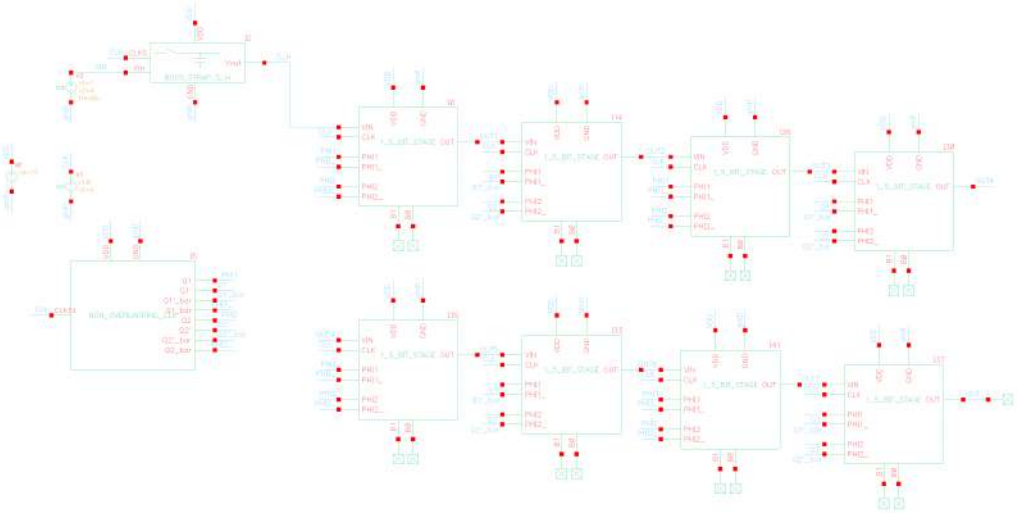


Figure 3.25: *Cascaded stages*

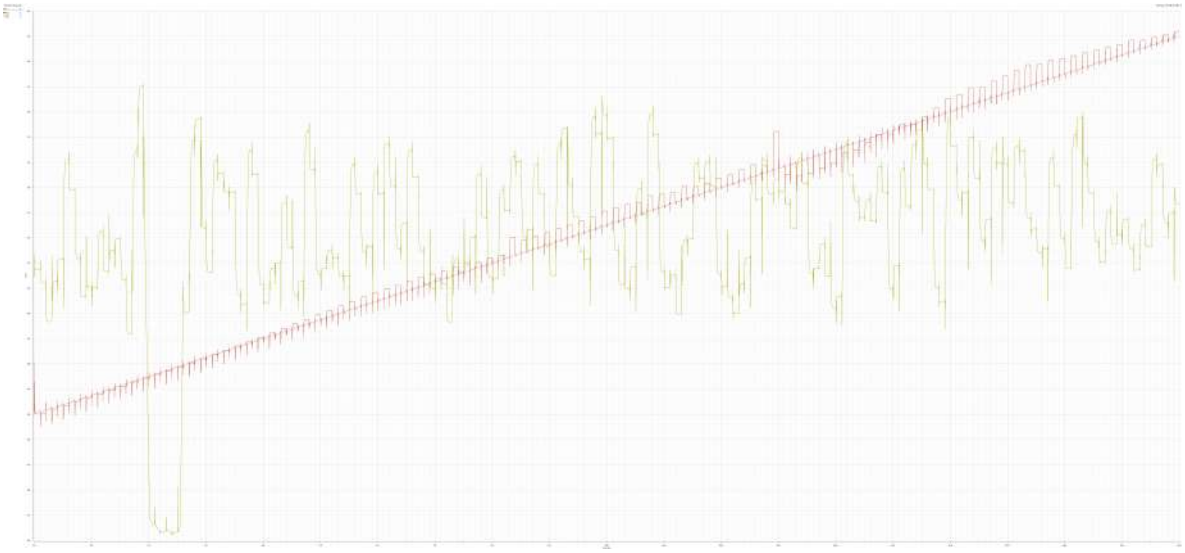


Figure 3.26: *8 Cascaded stage Result*

Our Output takes few clock cycle to settle then it stays within 1 to 4V range.

3.3.10 Test ADC array bits output

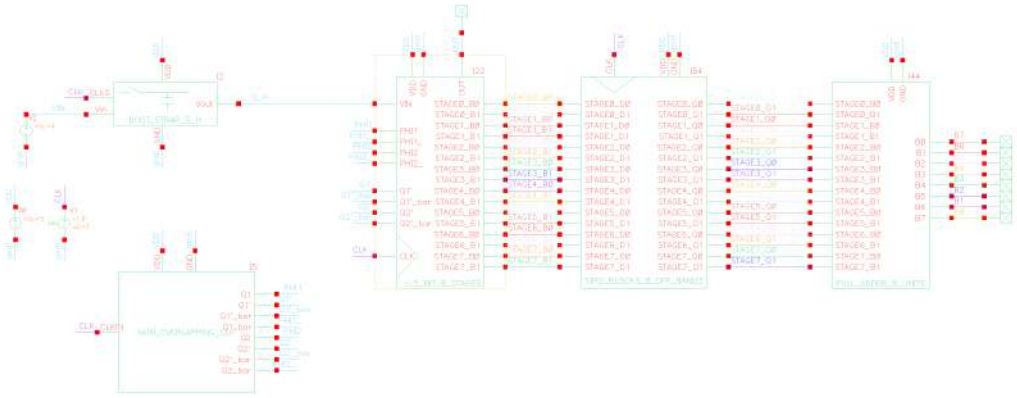


Figure 3.27: *At input 4 for Vin, Clock Frequency: 5Mhz*

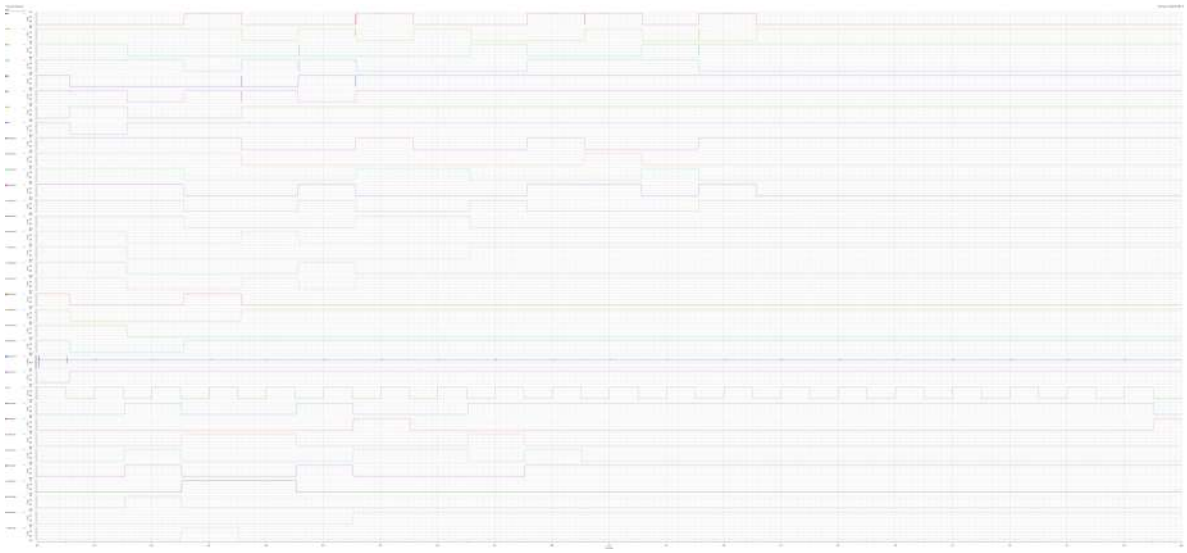


Figure 3.28: *Test Signal 4V, Clock 5MHz*

we can see we are getting output bits and the SIPO is shifts its and then the adder give the combined data.

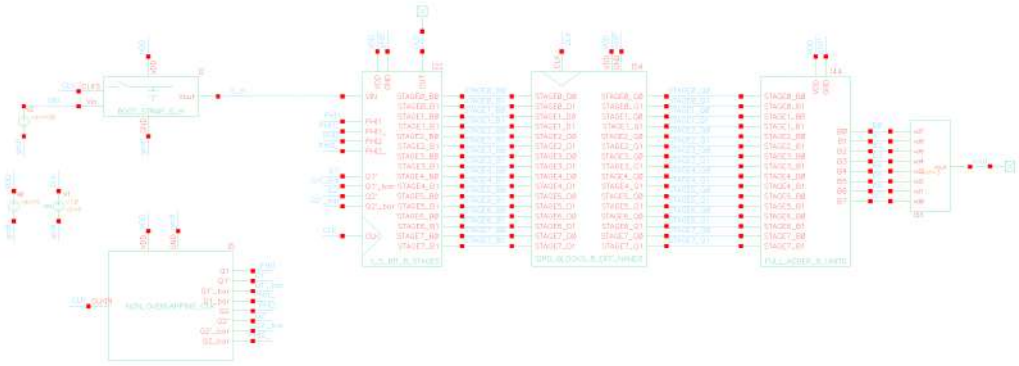


Figure 3.29: *Vin is 4 V and clock is 2MHz*

3.3.11 Test with Ideal DAC output

3.3.12 Test with RAMP Input

For out 1.5 bit ADC the bits till 4 works great but in some case as we can observe in figure 3.32 the bits flips. So we preferred the result of 1 bit ADC.

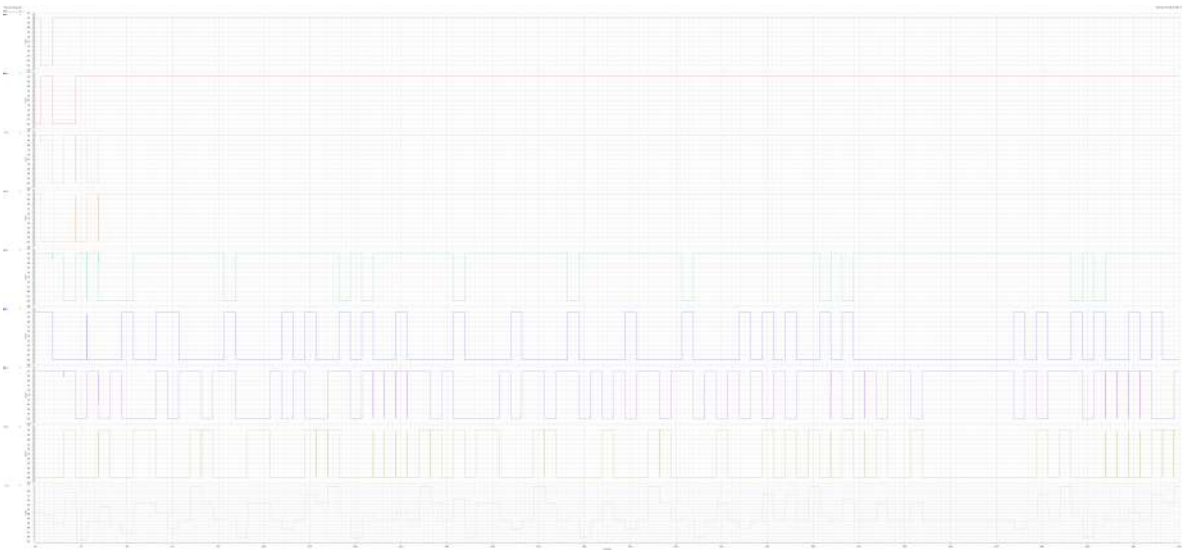


Figure 3.30: *Output bits going into DAC*

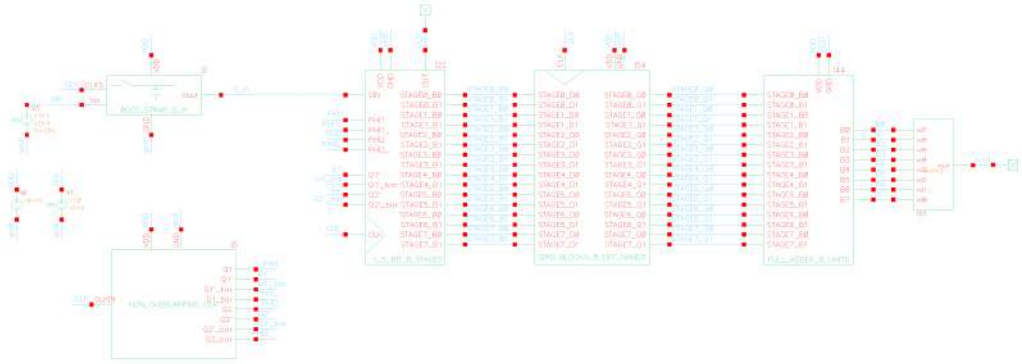


Figure 3.31: *RAMP input 1V to 4V, Rise time 20u*

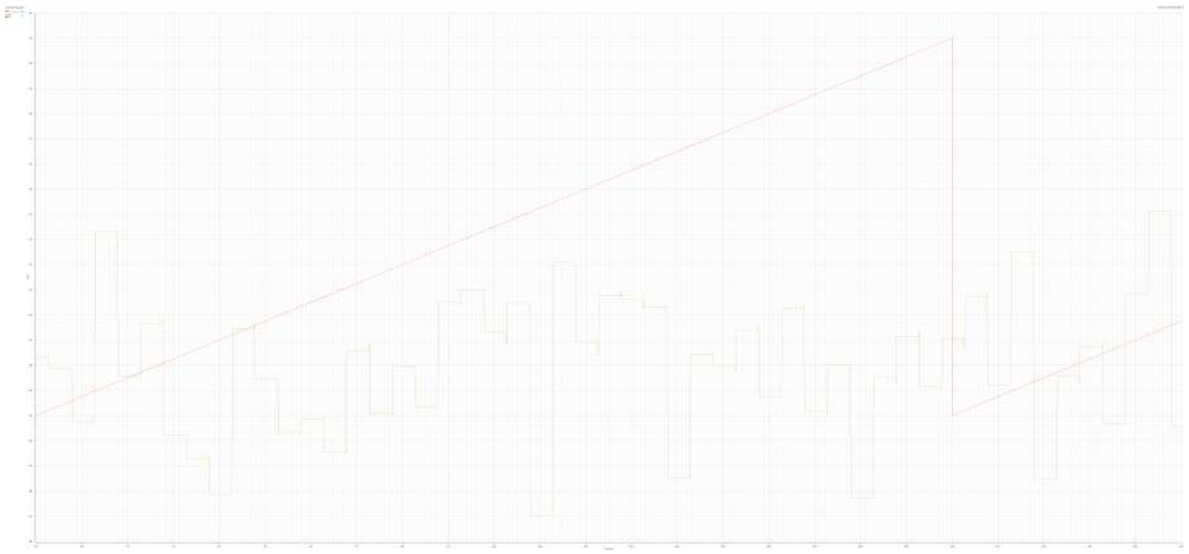


Figure 3.32: *DAC Output*

4

Design Metrics

4.1 Transfer Characteristic

4.1.1 The Transfer Characteristic and the MATLAB Code to Generate It

Transfer Characteristic

Initialization

```
clear all
```

Glolbal Variables

```
B = 8;
```

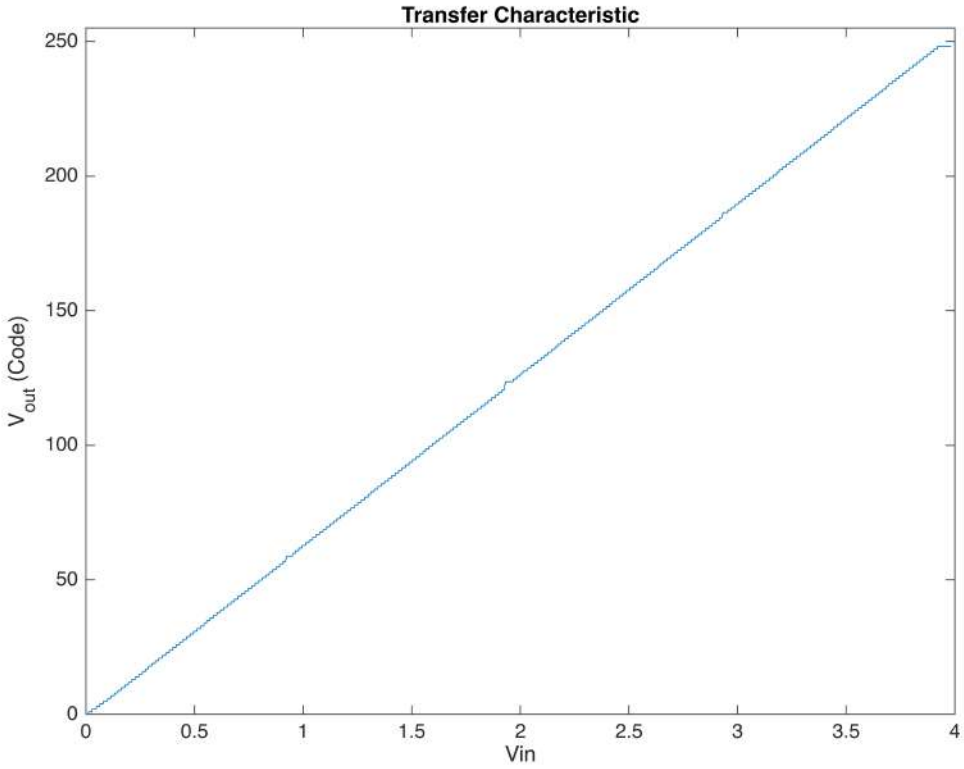
Load Data

```
Data1 = csvread("OUT8-FINE.vcsv", 6, 0);
t = Data1(:,1);
Vout = Data1(:,2);
Vin = 4.*(t(12000:end)-t(12000))/(t(500000));
```

Plot the Transfer Chacteristic

Assuming the input voltage increase from 0 V to 4 V

```
stairs(Vin, (Vout(12000:end)-Vout(12000))/4*255)
xlabel("Vin")
ylabel("V_{out} (Code)")
title("Transfer Characteristic")
ylim([0 255])
xlim([0 4])
```



Find the first transition level and the last transition level.

```
LastTran = 0;
FirstTran = 0;
TranCnt = 0;
for i=1:length(Vout)
    if (Vout(end-i) ~= Vout(end-i+1))
        LastTran = length(Vout)-i;
        break
    end
end
for i = 12000:length(Vout)
    if (Vout(i) ~= Vout(i+1))
        FirstTran = i;
        break
    end
end
for i = 12000:length(Vout)-1
    if (Vout(i) ~= Vout(i+1))
        TranCnt = TranCnt + 1;
    end
end
```

Quality of ADC Transfer Characteristic

Code below is to check if the monotonic is guareenteed in the transfer characteristic and if there is any missing code in the transfer characteristic.

```
PrevTran = FirstTran;
Monotonic = true;
MissingCode = false;
for i = FirstTran+1:LastTran
    if (Vout(i+1) ~= Vout(i))
        if (Vout(i) <= Vout(PrevTran))
            Monotonic = false;
            disp(sprintf("The non-monotonic happens at code %d",
ceil((Vout(i)-Vout(12000))/4*255)))
        end
        if ((Vout(i) - Vout(PrevTran)) >= 1.95*(4/255))
            MissingCode = true;
            disp(sprintf("The missing code happens at code %d",
ceil((Vout(i)-Vout(12000))/4*255)))
        end
        PrevTran = i;
    end
end
```

```
The missing code happens at code 123
```

```
Monotonic
```

```
Monotonic = logical
   1
```

```
MissingCode
```

```
MissingCode = logical
   1
```

4.1.2 Quality of ADC Transfer Characteristic

The monotonic is guaranteed in the transfer characteristic. There is a missing code at code 123.

4.2 INL and DNL

4.2.1 MATLAB Code to Calculate INL and DNL

DNL and INL

Initialization

```
clear all
```

Glolbal Variables

```
B = 8;
```

Load Data

```
Data1 = csvread("OUT8-FINE.vcsv", 6, 0);
t = Data1(:,1);
Vout = Data1(:,2);
Vin = 4.*(t(12000:end)-t(12000))/(t(500000));
```

Find the first transition level and the last transition level.

```
LastTran = 0;
FirstTran = 0;
TranCnt = 0;
for i=1:length(Vout)
    if (Vout(end-i) ~= Vout(end-i+1))
        LastTran = length(Vout)-i;
        break
    end
end
for i = 12000:length(Vout)
    if (Vout(i) ~= Vout(i+1))
        FirstTran = i;
        break
    end
end
for i = 12000:length(Vout)-1
    if (Vout(i) ~= Vout(i+1))
        TranCnt = TranCnt + 1;
    end
end
```

The Average Code Width

The average code width

```
Vin = 4.*t/t(end);
CWavg = (Vin(LastTran) - Vin(FirstTran)) / (TranCnt-1);
```

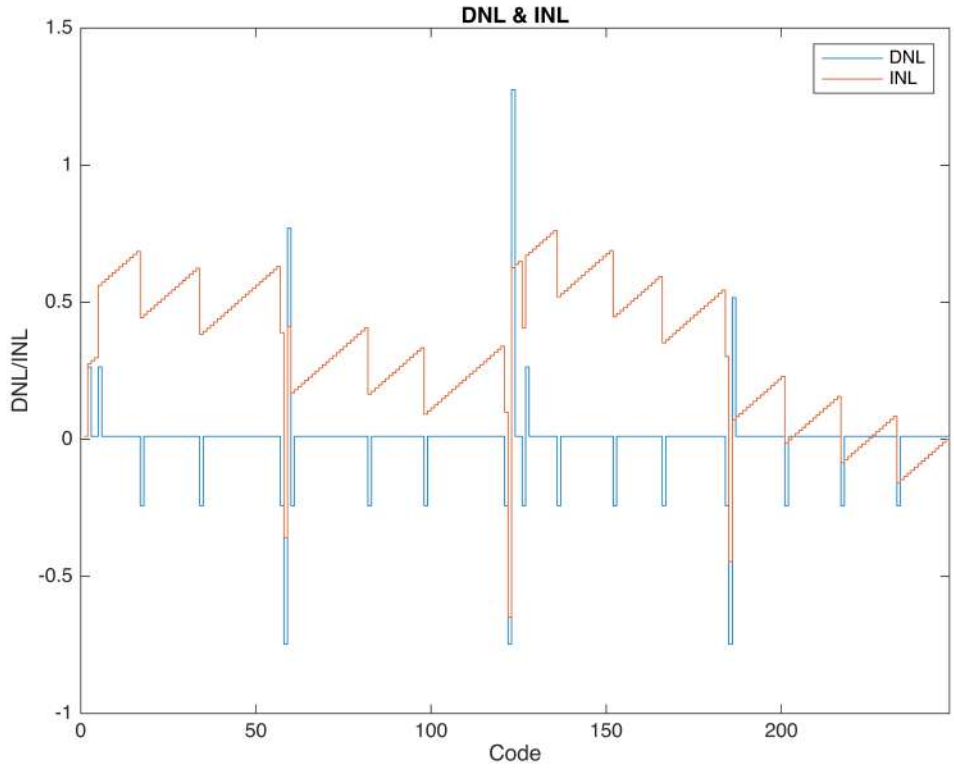
DNL & INL

Calculate the DNL and the INL

```
DNL = zeros(2^B,1);
INL = zeros(2^B,1);
PrevTran = FirstTran;
j = 1;
for i = FirstTran+1:LastTran
    if (Vout(i+1) ~= Vout(i))
        DNL(j) = (Vin(i) - Vin(PrevTran))/CWavg - 1;
        INL(j) = (Vin(i) - (j*CWavg+Vin(FirstTran)))/CWavg;
        j = j + 1;
        PrevTran = i;
    end
end
```

Plot the DNL and the INL

```
stairs(DNL)
title("DNL & INL")
ylabel("DNL/INL")
xlabel("Code")
hold on
stairs(INL)
legend("DNL", "INL")
xlim([0 TranCnt])
hold off
```



The max/min INL & DNL and the submax DNL.

```
INLmax = max(INL)
```

```
INLmax =
0.7615
```

```
INLmin = min(INL)
```

```
INLmin =
-0.6489
```

```
DNLmax = max(DNL)
```

```
DNLmax =
1.2753
```

```
DNLsubmax = max([DNL(1:find(DNL==DNLmax)-1);DNL(find(DNL==DNLmax)+1:end)])
```

```
DNLsubmax =
0.7697
```

```
DNLmin = min(DNL)
```

```
DNLmin =
-0.7472
```

4.2.2 Result

Our max INL is 0.7615 and our max DNL is 1.275;

Our max DNL is very close to 1, with discussion with Professor, she helped us understand that with this architecture (1 bit per stage) it is very difficult to get the DNL less than what we are getting.

4.3 Dynamic Power Dissipation

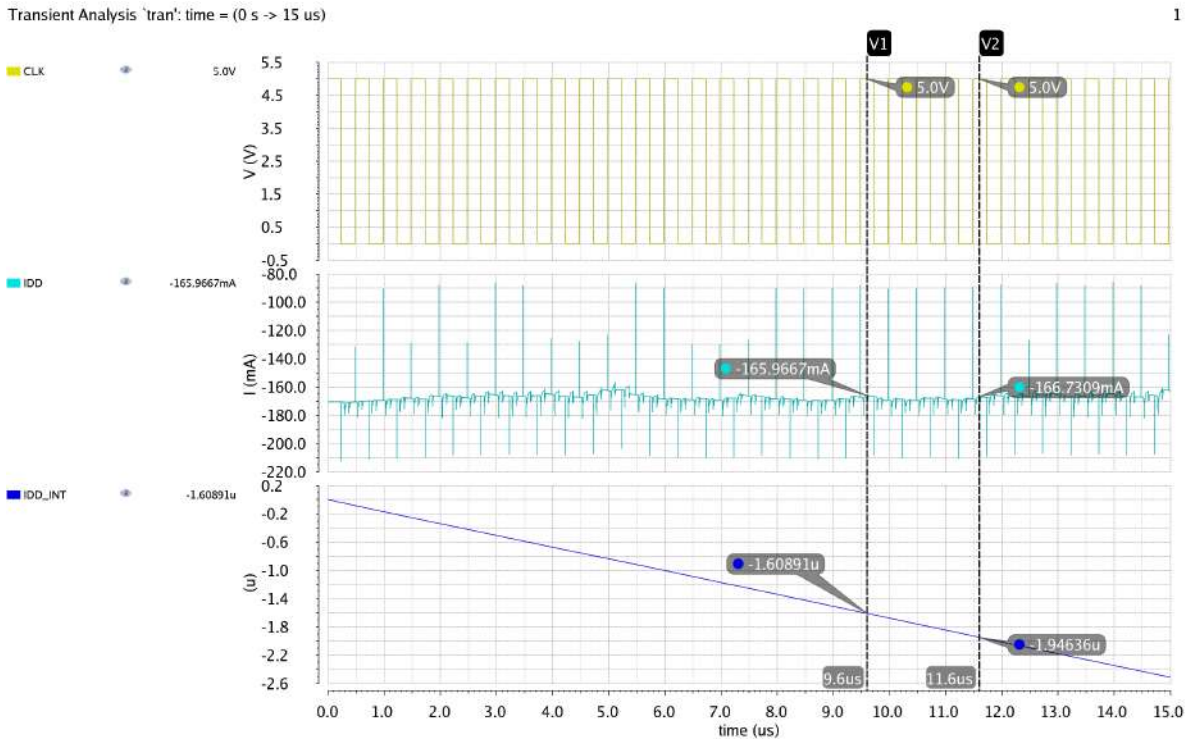


Figure 4.1: *Waveform of I_{DD} and It's Integral*

From Figure 4.1, we got that the average power

$$P_{avg} = \frac{1}{T} \int I_{DD} V_{DD} dt = \frac{1}{2\ \mu s} (1.94636\ \mu C - 1.60891\ \mu C) \times 5\ V = 0.843625\ W \quad (4.1)$$

4.4 SNDR, SFDR, and ENOB

4.4.1 MATLAB Code to Calculate SNDR, SFDR, and ENOB

AC Analyze

Initialization

```
clear all
```

Glolbal Variables

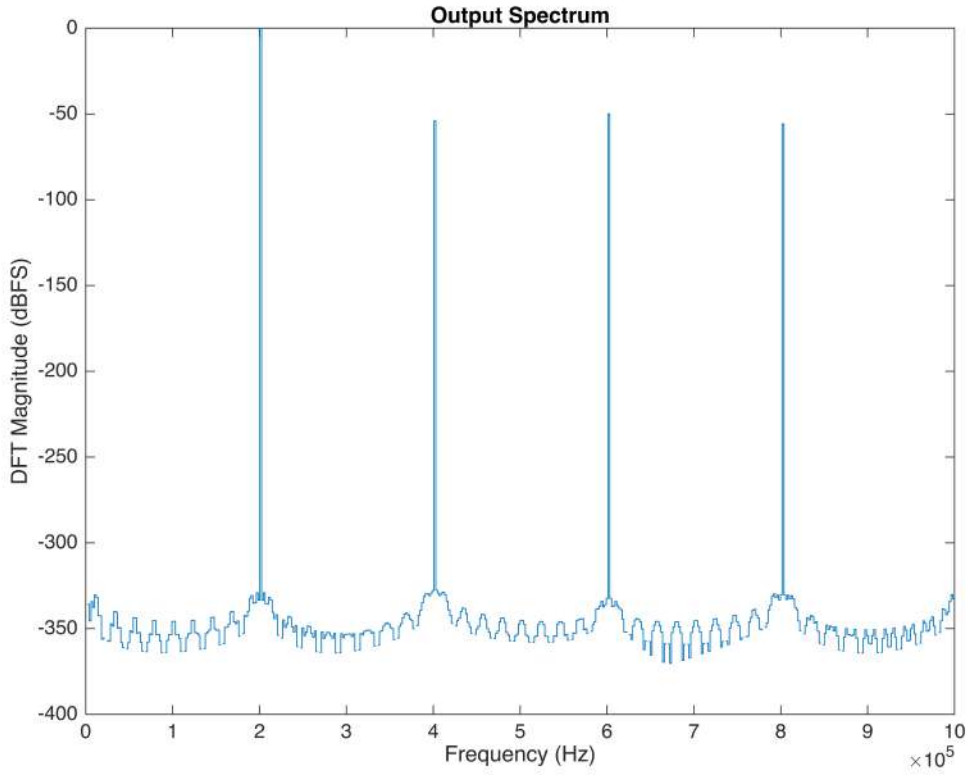
```
fsig = 200e3;
fdata = 1e9;
fsamp = 2e6;
nCycle = 99;
```

Load Data

```
Data2 = csvread("OUT8-200K.vcsv", 6, 0);
t = Data2(:,1);
Vout = Data2(:,2);
Vout = Vout(10e-6*fdata:(10e-6+nCycle/fsig)*fdata-1);
t = t(10e-6*fdata:(10e-6+nCycle/fsig)*fdata-1);
```

Plot FFT

```
Vdft = abs(fft(Vout));
Vdft = Vdft(2:floor(end/2*fsamp/fdata));
VdftdB = 20*log10(Vdft/max(Vdft));
f = linspace(fsig/nCycle, fsamp/2, length(VdftdB));
stairs(f, VdftdB)
title("Output Spectrum")
xlabel("Frequency (Hz)")
ylabel("DFT Magnitude (dBFS)")
```



SNDF & SFDR

Calculate the noise floor (dBFS).

```
nf = mean(VdftdB(~isinf(VdftdB)))
```

```
nf =
-345.9478
```

The noise power (dBFS)

```
Pn = nf + 10*log10(length(VdftdB))
```

```
Pn =
-319.0105
```

The SNDR (dB)

```
SNDR = -20*log10((sum(Vdft)-Vdft(find(VdftdB==0)))/max(Vdft))
```

```
SNDR =
43.3792
```

The largest spurious signal (dBFS)

```
Pspur = max([VdftdB(1:find(VdftdB==0)-1); VdftdB(find(VdftdB==0)+1:end)])
```

```
Pspur =
-49.9491
```

The SFDR (dB)

```
SFDR = -Pspur
```

```
SFDR =
49.9491
```

The ENOB

```
ENOB = (SNDR - 1.76)/6.02
```

```
ENOB =
6.9135
```

5

Conclusion

Table 5.1: *Final ADC Design Specifications*

Specification	Value / Description
ADC Architecture	Pipeline
Resolution	8 bits
Number of Stages	8 (1 bit per stage)
Supply Voltage (V_{DD})	5 V
Reference Voltage (V_{ref})	4 V
Input Signal Bandwidth	DC to 1 MHz
Sampling Frequency (f_s)	2 MHz
Input Voltage Range	0 to 4 V (centered around $2.5V$ with peak to peak $1.5V$)
Track-and-Hold	Project 2 (1bit) || Bootstrap Sample-and-Hold (1.5bits)
Sub-ADC	Dynamic latch comparator
DAC Levels	2 levels (1-bit DAC)
Residue Amplifier Gain	$\times 2$
DNL (Max)	≈ 1 1.2753 LSB
INL (Max)	0.7615 LSB
SNDR	43.38 dB
SFDR	49.95 dB
ENOB	6.91 bits
Power Dissipation	0.843625 W
FOM	1.648E-9
FOM (Using ENOB)	3.508E-9

In this project, we successfully implemented an 8-bit pipeline ADC architecture using both 1.5-bit and 1-bit stages. While the 1.5-bit stages provided inherent redundancy and digital error correction, we observed occasional bit-flipping at specific input

levels. To ensure more stable and consistent performance across the entire input range, we adopted a 1-bit-per-stage architecture in the final design. This allowed for clean data alignment, easier debugging, and reliable digital output with minimal complexity. Overall, the pipeline ADC achieved the desired resolution and bandwidth targets, and our design choices balanced accuracy, power, and robustness in simulation.