UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

EE6045: Data Converters
MATLAB Assignment 2024

Name: Alex O'Mahony

Student Number: 119374921

Date: 17/03/2024

Note: the random seed was kept at 31233 for all tests for repeatability

Question i)

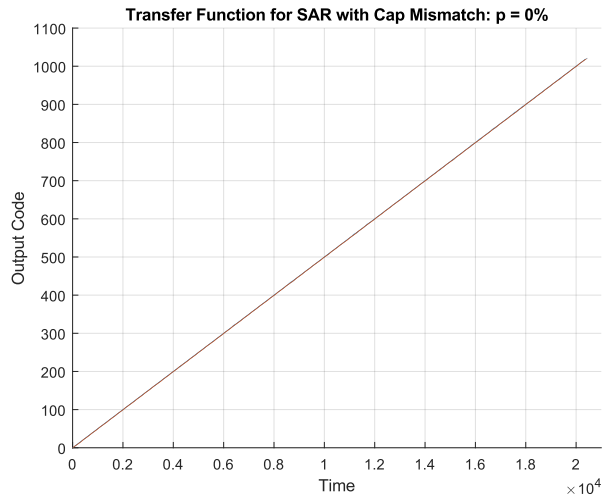
To add capacitor mismatch we can use the same method used to add mismatch to the resistor ladder. Instead, here we apply the mismatch to the **Vcdac** array. One key difference between the mismatch in the resistor ladder and the cap-dac is that the mismatch applied is proportional to the capacitor it is applied to. For instance, an 8C capacitor is going to have a higher mismatch value than a 2C capacitor as $x\%\ of\ 8C > x\%\ of\ 2C$.
One issue with this method of applying mismatch is that the mismatch for each resistor/capacitor is independent of the rest. In reality, the mismatch for each resistor/capacitor should compound and affect the rest of the values. For instance, if the bottom resistor in the ladder is off by -5% and the next one up is also off by -5% then the voltage at the top of the second resistor should be off by -10% and not -5% as is implemented here.

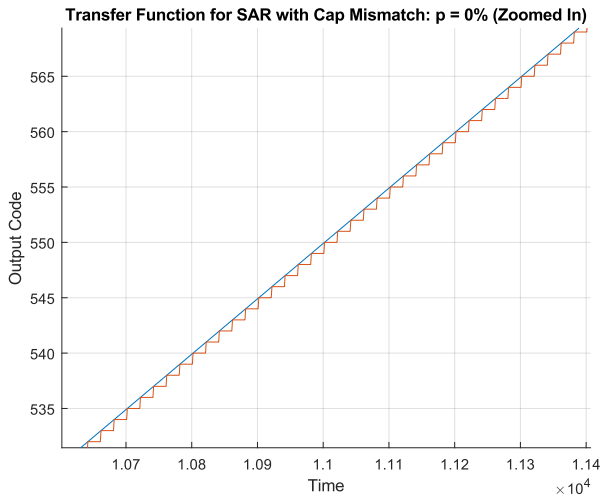
```
1. cap_p = 1/100;                       % Set percentage of mismatch to apply
2. randn('seed', 31233);                % Set random number seed
3. mismatch = randn(1,Nbit)*cap_p;      % Create array of random offsets to apply to caps
4. Vcdac = Vcdac.*(1+mismatch);         % Apply mismatch to capacitor vals (proportional to size)
```

Question ii)

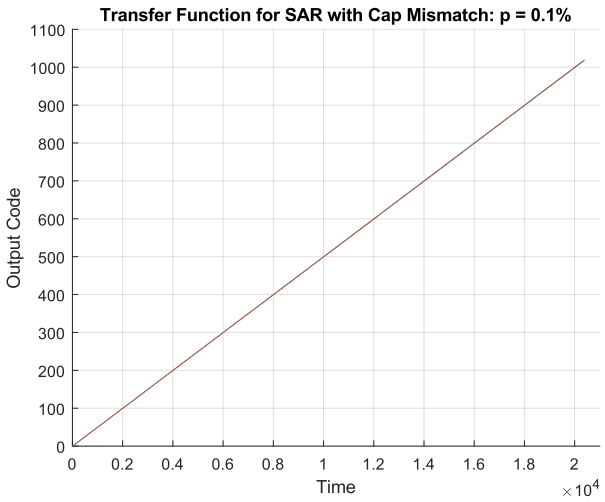
Transfer Function

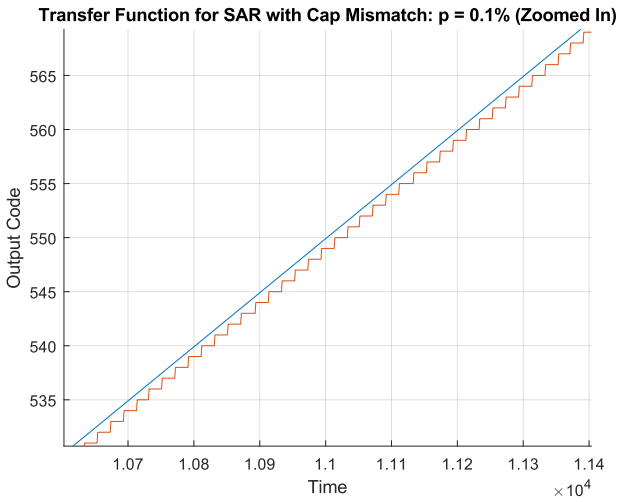
Capacitor mismatch **p = 0 %**



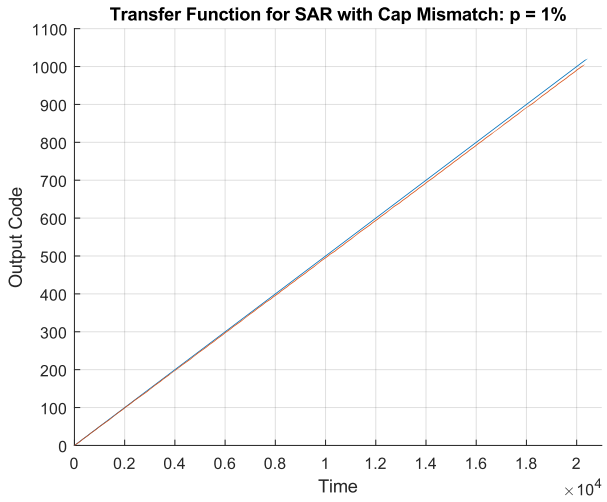


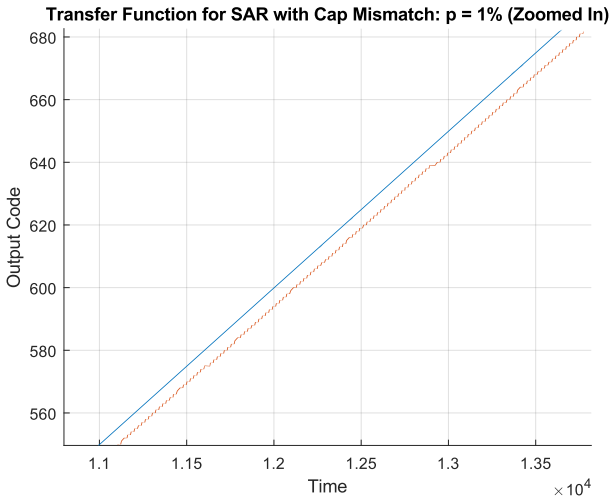
Capacitor mismatch **p = 0.1 %**



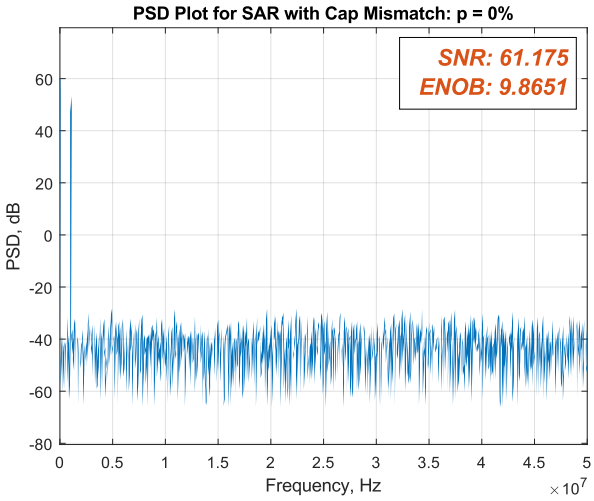


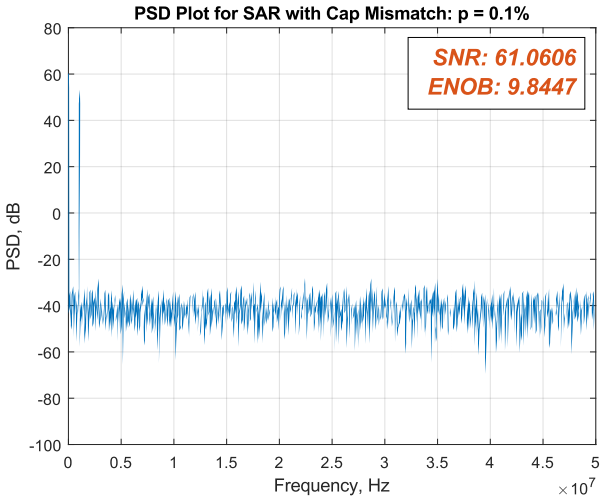
Capacitor mismatch **p = 1 %**

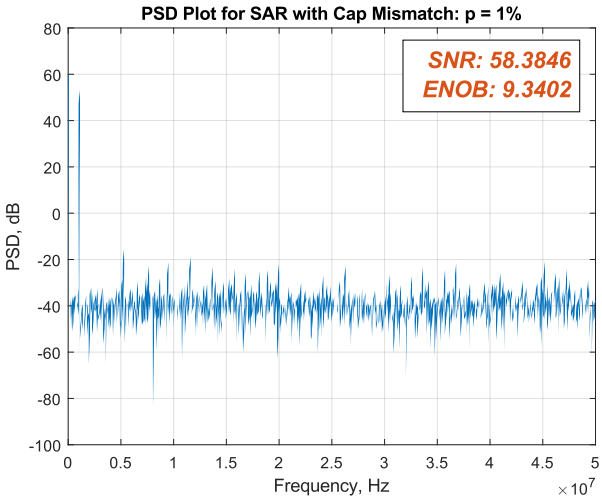




Spectrum, ENOB, SNR

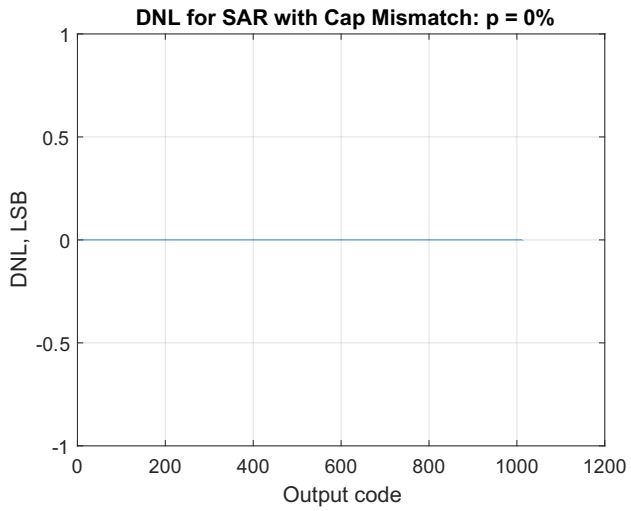


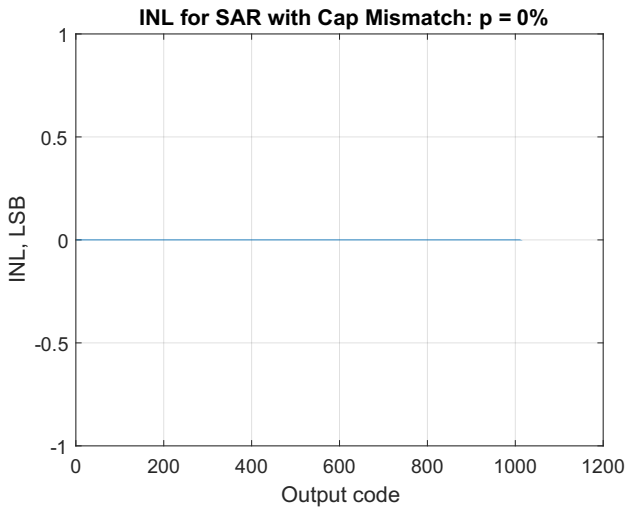




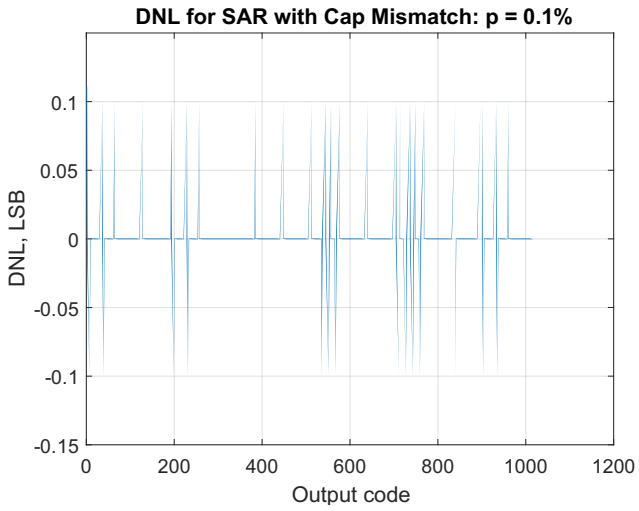
INL/DNL

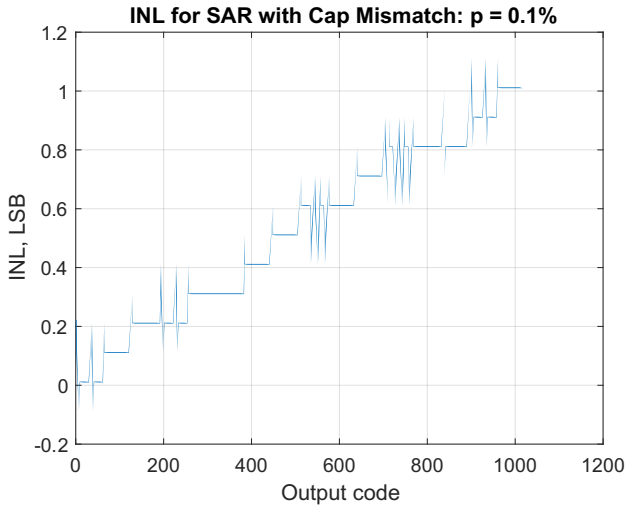
Capacitor mismatch **p = 0%**



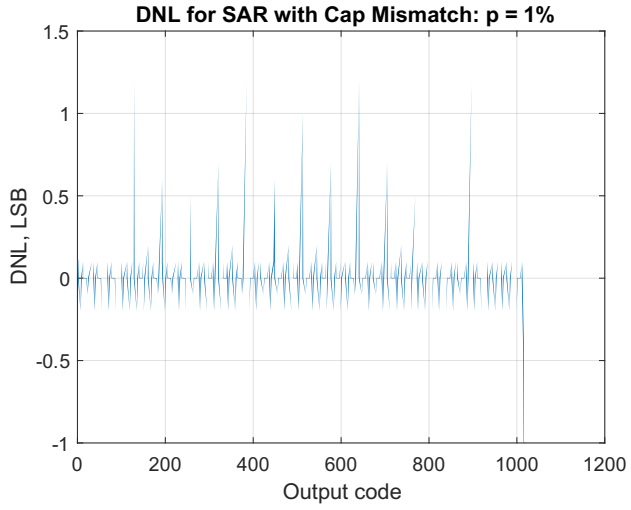


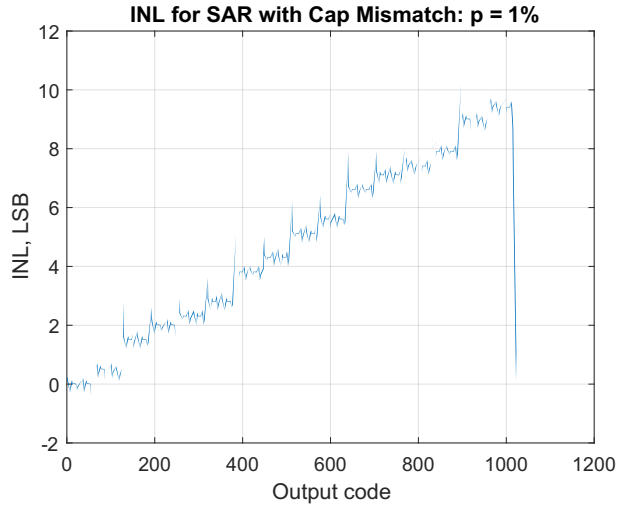
Capacitor mismatch **p = 0.1%**





Capacitor mismatch **p = 1%**





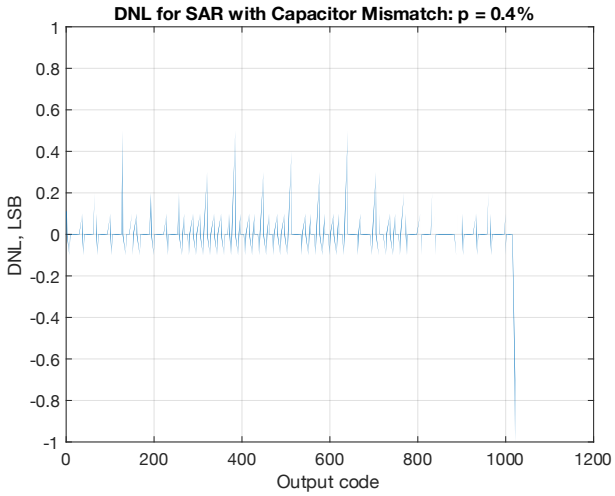
Which one is more robust to mismatch?

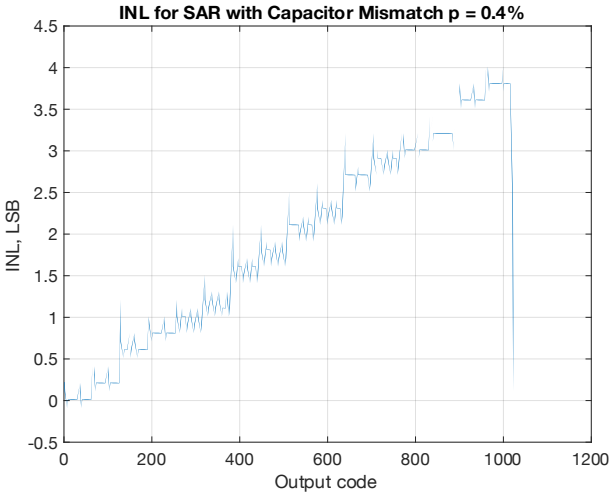
The SAR ADC is much more robust to mismatch. We can see that for the SAR with 0.1% mismatch the ENOB decreases from the original 9.8651 down to 9.8447, whereas in the Flash adc from the notes provided we saw that a mismatch of 0.1% reduced the ENOB from 9.86 down ti 8.97. This small mismatch has a much stronger effect on the flash ADC.

Similarly, when looking at the DNL/INL of both flash and SAR when mismatch is applied

Maximum mismatch to get 0.5LSB DNL/INL

We can see from the plots above that a mismatch of p=1% causes the DNL to rise above 0.5LSB multiple times, and the INL rises gradually far above 1 LSB. Slowly increasing the mismatch from 0.1% where we can see the DNL is safely within ±0.5LSB we can find the point at which the max DNL is around 0.5LSB. This comes to be a p value of about p=0.4%. This value for p however, allows the INL to increase gradually to nearly 4 LSB. We can see from the more extreme mismatch p = 1% transfer function that the output codes are mostly linear but have a different slope, which is what is causing the DNL to tend to one side (positive or negative) and thus the INL grows gradually. I believe this can also be accommodated for on the digital end with just the extreme values being most negatively affected (V_FS and GND).



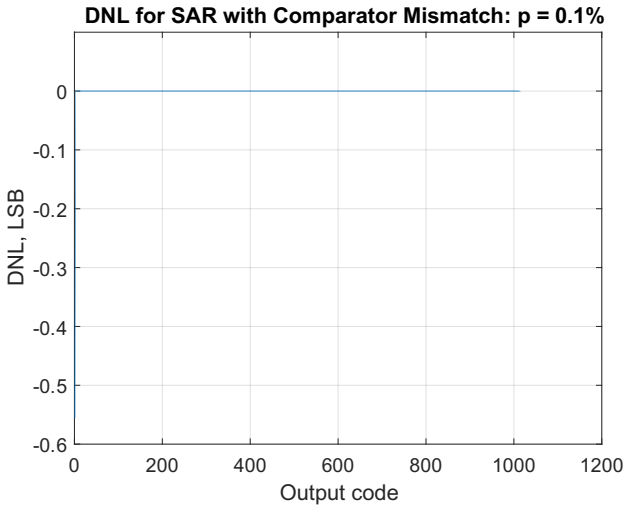


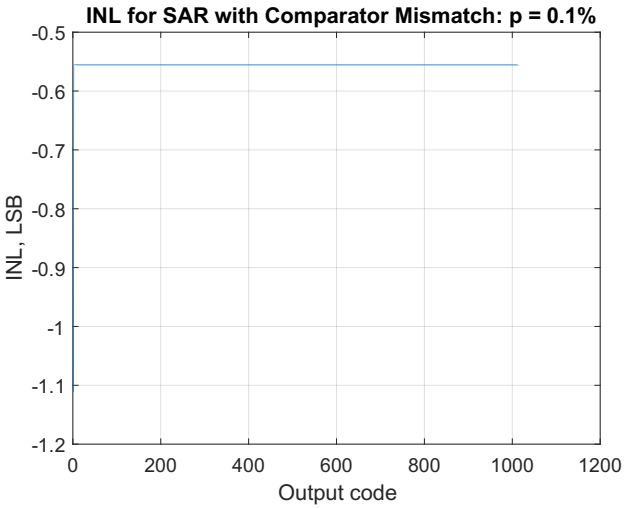
Question iii)

DNL/INL

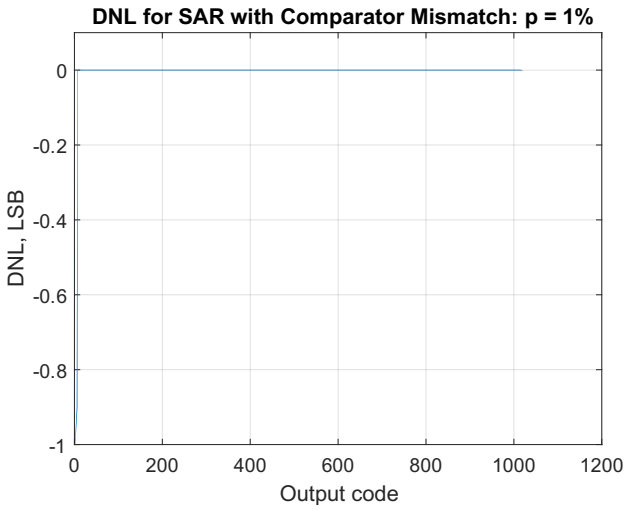
To test comparator mismatch and isolate the effect, capacitor mismatch for this section was set to 0%. This allows us to analyse the effect of just the comparator mismatch without the results being skewed by the capacitor mismatch.

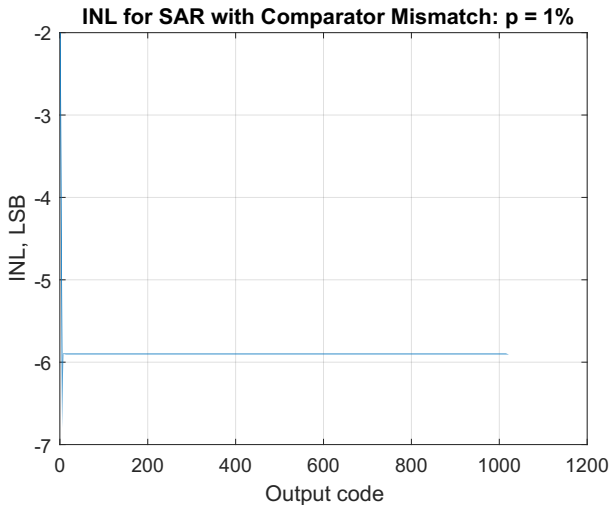
Comparator mismatch **p = 0.1%**



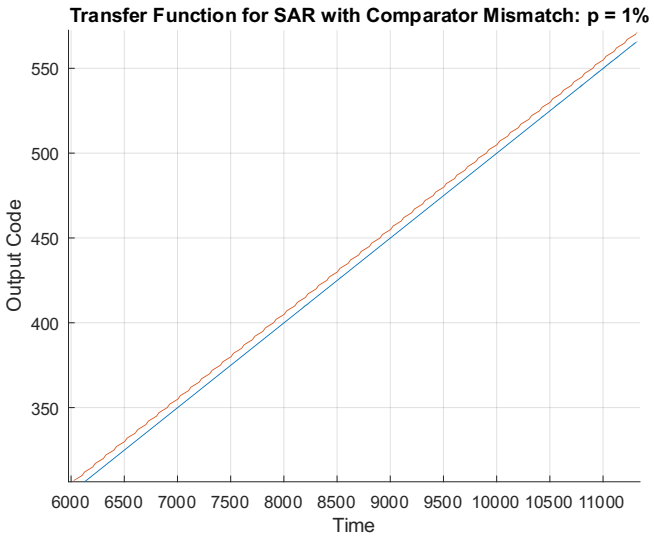


Comparator mismatch **p = 1%**



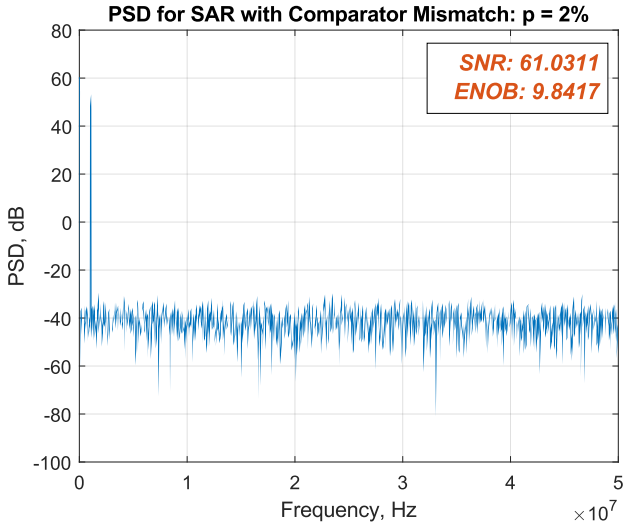


We can see that with increased comparator mismatch the DNL is mostly unnaffected except for the lower end of the range. The INL initially shows poor performance, however, this large offset is all due to the initial few DNL output codes. Taking a closer look at the transfer function, shown on the right, we can see that the output codes are smooth and linear, but have an offset due to the comparator mismatch. Since the error in the output code is mostly constant across the entire range, this is an error that can be accounted for on the digital side. The only major issue is the reduced accuracy around the maximum values, GND and V_FS.



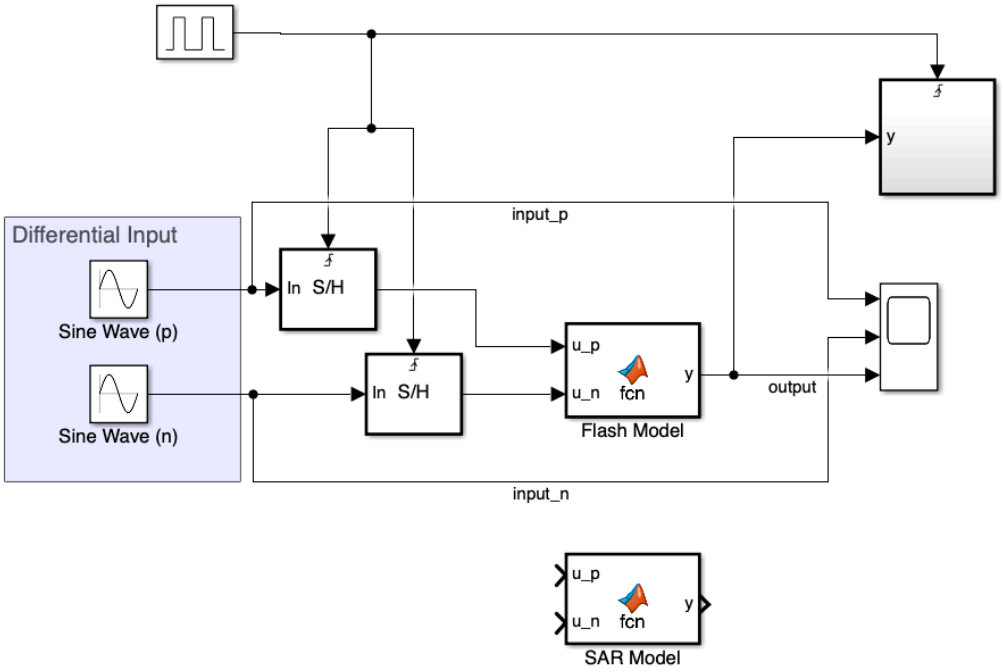
Spectrum

Introducing comparator mismatch has very little effect on the Spectrum, SNR and ENOB values. As we can see in the plot below, even with a comparator mismatch of p=2%, the SNR is almost the same as without any (SNR for p=0% was 61.175). The same goes for the ENOB value, only decreasing from 9.8651 to 9.8417. The noise floor also stays around -40dB and does not rise much with comparator mismatch. This shows that the SAR ADC is much less affected by comparator mismatch compared to the Flash ADC. This is due to there only being one comparator in the SAR compared to $2^N - 1$ comparators in a Flash where each mismatch is compounded with each other.



Question iv)

The Simulink model must first be modified to allow for a fully differential input. The input sine wave is duplicated with a 180° phase shift from the original to make the two signals complementary.



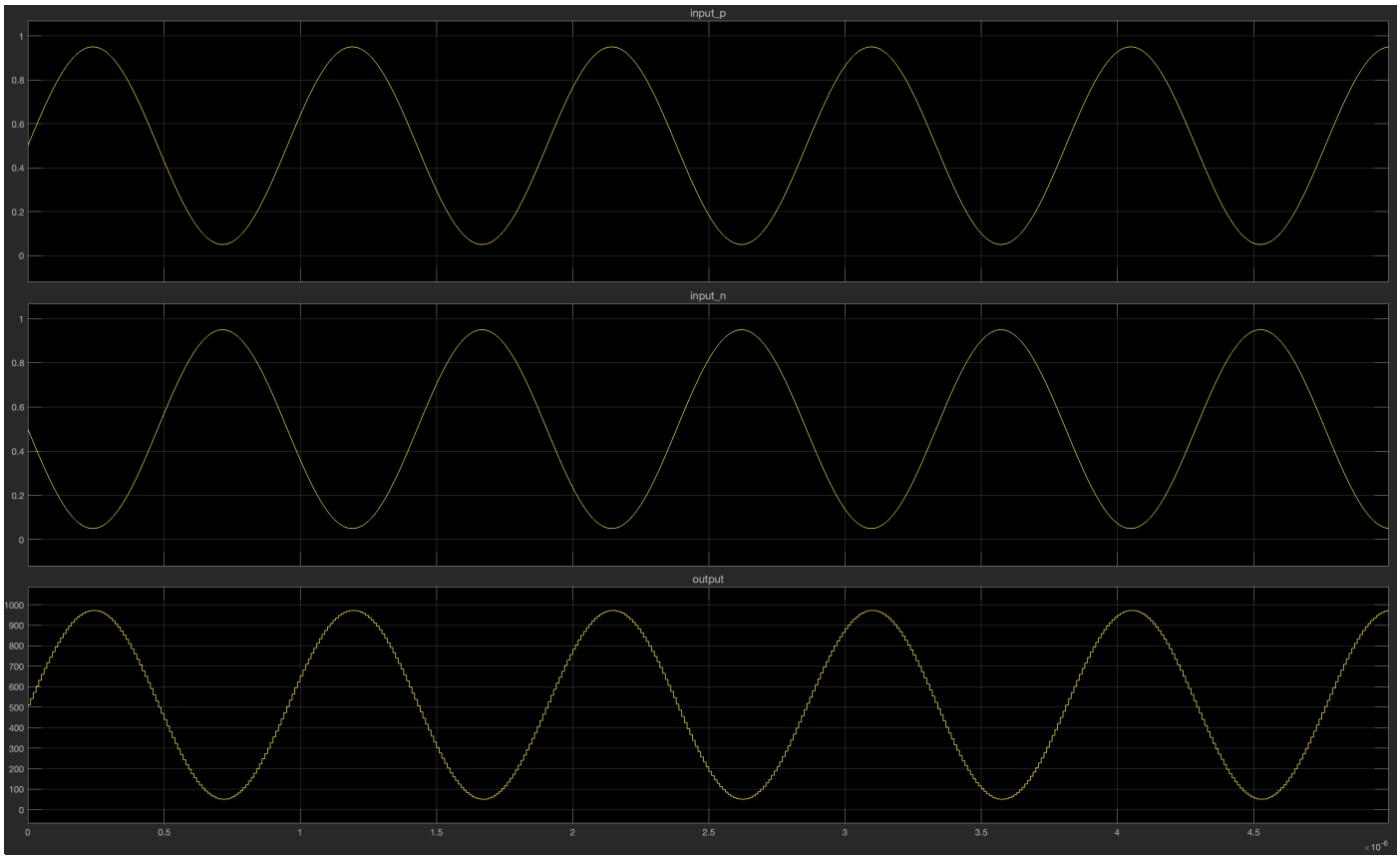
Flash ADC

For a fully differential input, the flash ADC will generate two thermometer codes, one for the positive input and one for the negative input. The two thermometer codes can then be averaged to give the output code. This helps to ensure that any common-mode noise or interference present in both the positive and negative inputs is rejected. The differential flash uses two resistor ladders, with the positive ranging from GND to V_FS and the negative the same but with inverted polarity. The code for this setup is shown below. The rladder_p and rladder_n are equal but inverted.

```
function y = fcn(u_p, u_n)
vin_p=u_p;
vin_n=u_n;

Nbit=10;                        %%Number of bits.

LSB=1/2^Nbit;                   %%LSB size (considers Vp=1 and Vn=0).
rladder_p = [1:1:2^Nbit-1]*LSB;     %%%Reference voltage generation.
rladder_n = fliplr(rladder_p);


p=0.1/100;                      %%percentage of mismatch
randn('seed', 31233);           %% Uses a fixed seed for the PRNG

mismatch_p=randn(1,2^Nbit-1)*p;   %%Random variable to be added to rladder - positive
rladder_p=rladder_p.*(1+mismatch_p);  %%Mismatch added to rladder.

mismatch_n=randn(1,2^Nbit-1)*p;   %%Random variable to be added to rladder - negative
rladder_n=rladder_n.*(1+mismatch_n);  %%Mismatch added to rladder.

vout_p=ones(1,2^Nbit-1);          %%%Formal initialisation of vout variable.
vout_n=ones(1,2^Nbit-1);

for i=1:1:2^Nbit-1              %%2^Nbit -1 comparators
    % Compare positive input against positive reference ladder
```

```
    if vin_p < rladder_p(i)
        vout_p(i) = 0;
    else
        vout_p(i) = 1;
    end

    % Compare negative input against negative reference ladder
    if vin_n < rladder_n(i)
        vout_n(i) = 1;
    else
        vout_n(i) = 0;
    end
end

y= 0.5 * (sum(vout_p)+sum(vout_n));
```

Flash ADC Output



SAR ADC

The SAR model was made fully differential by following the algorithm described in “A Study of SAR ADC and Implementation of 10-bit Asynchronous Design” [1]. For the MSB test we can simply compare the two inputs, vin_p and vin_n. We know that if vin_p > vin_n then the input must be greater than the Vcm. We can then continue to determine the exact delta between the positive and negative inputs. Since we know both inputs are around a common Vcm, all we need to determine is the voltage difference between the signals at each point.
Below is the code to achieve this algorithm, along with the output plots produced. The Simulink setup is exactly the same as the Flash model setup with the Flash and SAR blocks switched.
The value ***test_voltage*** is altered for each bit test depending on the result of the previous bit test, procedurally narrowing down the testing voltage until it finds the input voltage value down to 1 LSB.

```
function y = fcn(u_p, u_n)
vin_p=u_p;
vin_n=u_n;

Nbit=10;                    %%%Number of bits

Vcdac = 1./2.^[1:1:Nbit];   %%Voltage vector from CDAC.

sar=ones(1,Nbit);           %%%Formal initialization of variable

p=0.0/100;                      %%percentage of mismatch
randn('seed', 31233);           %% Uses a fixed seed for the PRNG
mismatch = randn(1,NBit)*p

Vcdac = Vcdac.*(1+mismatch);

test_voltage = 0;               % voltage to compare vin_p - vin_n to

for i=1:1:Nbit

    if vin_p - vin_n > test_voltage
        sar(i) = 1;
        test_voltage = test_voltage + Vcdac(i);
    else
        sar(i) = 0;
        test_voltage = test_voltage - Vcdac(i);
    end

end

weights=2.^[0:1:Nbit-1];
weights=fliplr(weights);
y=sum(sar.*weights);

```

SAR ADC Output