

# DESIGN OF 12-BIT ADC

## End-term Project, 6.775

Wesley O Jin, Varun Aggarwal

### 1. Introduction

In this project, we were asked to create a 12-bit ADC that would be accurate to  $\pm 1$ LSB of INL and  $\pm 1/2$ LSB of DNL. Our primary concern was to find ways of dealing with errors resulting from capacitor mismatch and comparator offset. Using an op-amp with a gain of more than 19,500 and ideal switches, problems resulting from finite gain and charge injection were of secondary importance.

Our design features fourteen, 1-bit pipelined stages, eleven with a gain of two and two with a gain of one for digital error correction. The use of unity gain stages is a widely used technique to compensate for comparator offset.<sup>1</sup> To correct for deviations between capacitances, our ADC features two capacitor arrays of fourteen components each. A fabricated chip would use a digital core to order pairs of capacitors from each array according to how well they match with one another. Each pair would then be assigned to a stage with the best-matching pair going to the first and the worst going to the last. In doing so, the crucial, initial stages are made to be the most accurate at the expense of accuracy of later stages where it is less important.

### 2. ADC Specifications

Input Voltage Range: 0.6, 1.2

12-bit output

Ideal Voltage Sources Available: 0.6V, 1.2V, 0.9V(CM)

Available: A clock of frequency 5MHz and its complement.

### 3. Design of basic ADC

Our ADC used the basic structure of a pipeline ADC. It has eleven stages (comparators plus additional stages for error correction). Each stage computes a single bit and creates a residue voltage, which is then used by the next stage to compute the next bit. It contains the following elements:

1. Multiplier and Subtractor: Each stage performs a multiply by 2 operation and also a subtraction operation to map the input voltage to a residue, which can be fed into the next stage for calculation of the next lower MSB. The multiplier-subtractor is shown in Fig.1.

---

<sup>1</sup> Y. Chiu, P. R. Gray and B. Nicolic, "A 14-b 12-MS/s CMOS Pipeline ADC With Over 100-dB SFDR" IEEE Journal of Solid-State Circuits, vol. 39, pp. 2139 - 2151, December 2004

Implementation Details: We used the opamp as designed in the mid-term project. It meets all specifications except that its noise is 40% the requirement. We used ideal switches instead of MOS switches to eliminate charge injection. The input range was fixed from 0.6V to 1.2V for two reasons: A. The gain of the opamp decreases considerably at points below 0.6V, which gives error from the gain of 2 curve, B. A MOS pass-gate can only transmit a voltage which is less than the full-scale voltage by the threshold voltage,  $V_t$ .  $V_t$  for the devices is around 0.5, so it is safe to keep the voltage 0.6V and 1.2V.

The opamp settles well in 100ns as desired and it is shown in Fig. 2 with capacitor size of 0.5pF(For input of 0.6). The settled output was checked at four points, 0.6V, 0.9V 0.9V(comparator forced to 1) and 1.2V. It gave a gain of 2 with 12-bit accuracy.

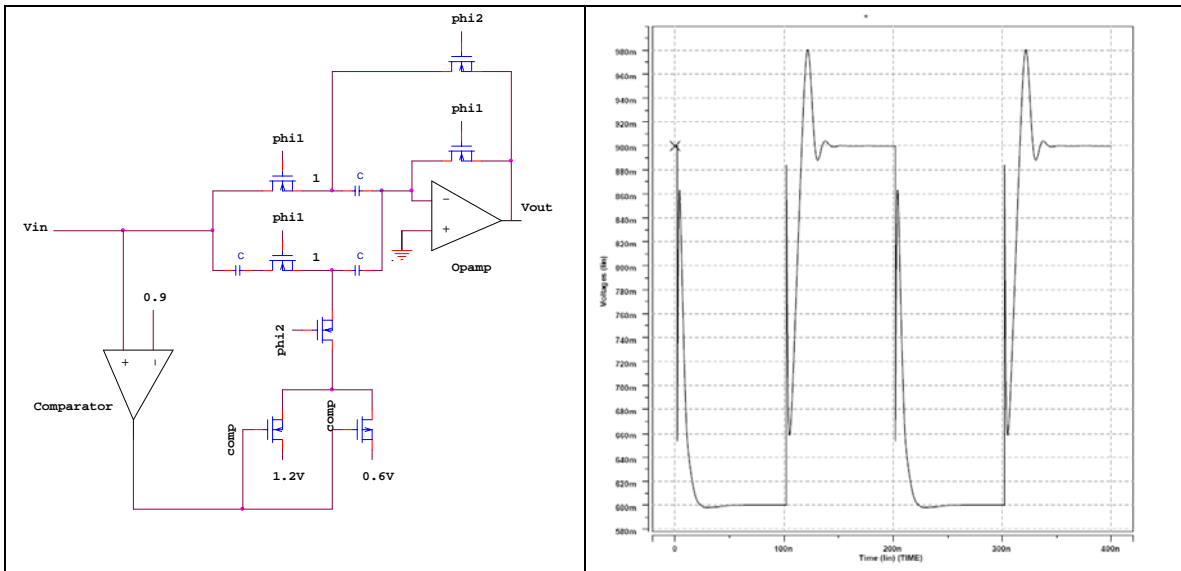


Figure 1: Circuit for single stage of gain 2      Figure 2: Residue plot for 0.6V

2. Comparator: A simple comparator was used in this project. It contains the first gain stage of our mid-term opamp followed by a fast CMOS digital latch. A latch is used for two purposes: A, To increase the speed of the comparator, B. The comparator makes the decision in the sampling phase and keeps that value in the multiply by two stage. The comparator is shown in Fig. 3. We planned to use digital error correction to correct for the comparator offset and the scheme performed well. The comparator can be further optimized for power, speed and accuracy. We didn't spend time on it, since that was the not the main goal of the project. We wanted to see whether the ADC could be made to work by using simple off-the-shelf blocks.

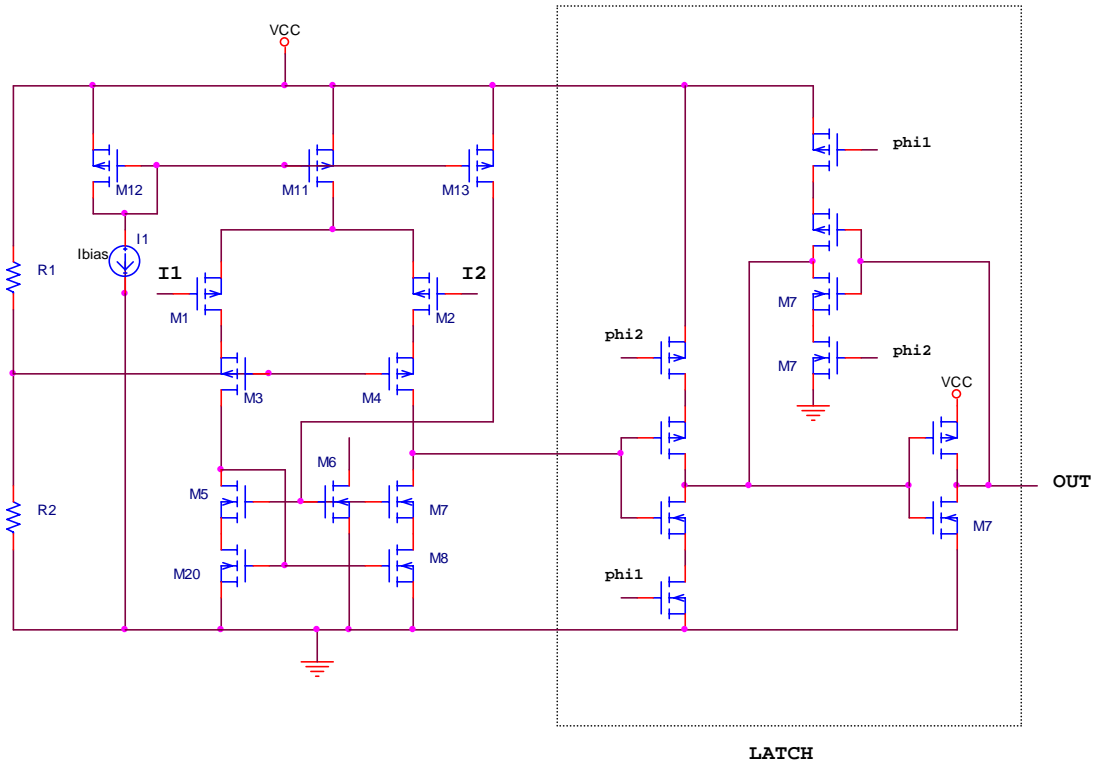


Figure 3: Comparator – Differential Stage followed by a latch

3. Multiply-by-1 and Subtraction Stage: The multiply-by-1 stage and subtraction was required for Digital Error Correction (DEC). It mapped the input voltage between 0.6 to 1.2 to a residue of 0.6V to 0.9V. The multiply-by-1 stage is very similar to the multiply by 2 stage and its schematic is shown in Fig. 4.

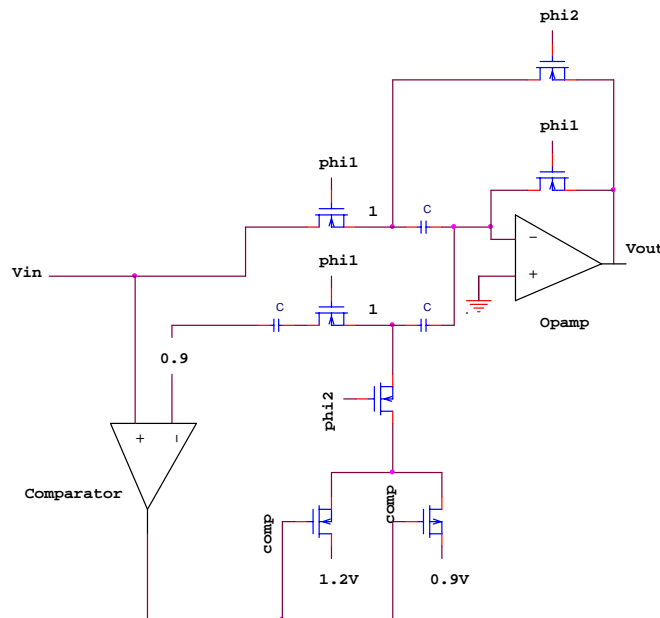


Figure 4: Single stage of gain 1 for Digital Error Correction

#### 4. Digital Error Correction

Digital Error Correction (DEC) is done after every three bits. In the first attempt, we tried DEC after the first 6 stages. However, our ADC suffered from a high DNL due to saturation. We then tried DEC for every 3-bits except for the last three as, shown in Figure 5, where there is a gain of stage 1 after the first two gain of 2 stages, then another gain of 1 stage after 3 gain of 2 stages and a final gain of 1 stage after another 3 gain of 2 stages.

As shown in the Figure, our system overlaps the bits and then sums them. The result propagates from LSB to MSB. In an actual chip, these operations would be performed by a digital core, which takes in 15-bits and gives out the corrected 12-bits.

For simulation, we manually shifted and added the codes to form a table of readings.

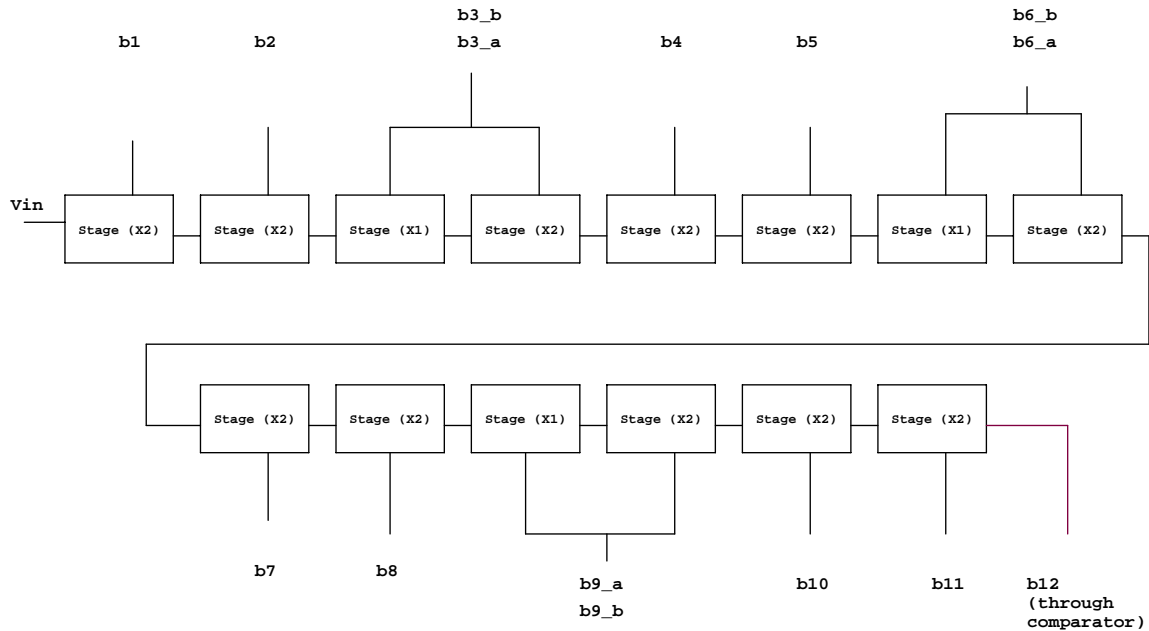


Figure 5: The final ADC pipeline

#### 5. Correction for Capacitor Mismatch

We assumed that capacitances would have a Gaussian distribution centered at the specified value with a standard deviation given by  $0.0025/(\text{cap\_value})^{0.5}$ . In a pipelined ADC with  $b$ -bits of resolution, a maximum capacitor mismatch of  $1/2^{(b-i)}$  is allowable in each of the  $i$ th stages.

Our ADC contains two groups of fourteen capacitors each, which can be connected to any of the fourteen stages via MOS switches. After fabrication, all capacitances are

measured by an analog circuit and converted into digital values. A digital circuit then pairs capacitors from each group, according to how well they match with one another, and produces the necessary control signals to connect the respective capacitors to each stage. Stages are assigned pairs in the order that they appear in the pipeline so that the first stage receives the best matching pair and the last receives the worst.

The feasibility of this scheme was tested in MATLAB. Fourteen pairs of mismatched capacitor (valued around 0.5pF) were generated, sorted and then inserted into each stage using the scheme described above. Each stage was then tested for an accuracy of  $1/2^{(b+1-i)}$ . In 1000 simulations, 230 capacitor-pairs failed to meet this requirement, giving an acceptable yield of more than 75%. With a faster opamp facilitating the use of higher value capacitors such as 1pF, the yield increases to more than 95%.

A single capacitor with MOS switches would appear as shown in Fig. 6. (with three switches shown) We call this a Capacitor-Switch Array.

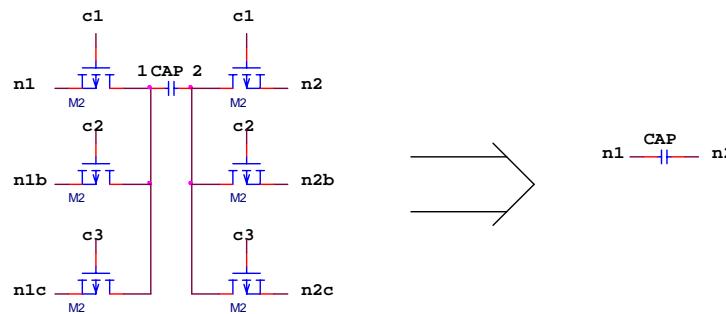


Figure 6: Capacitor-Switch Array<sup>2</sup>

For any two nodes in a given stage, n1 and n2, fourteen capacitors are connected through two MOS switches as shown in the figure 6 (one of the fourteen capacitor-switch arrays is shown). At a given time, only one MOS pair around the CAP is ON while all others are OFF. For instance, in the above figure, to connect the CAP to n1 and n2, c1 would be pulled high, while all other control signals c2...c14 would be low.

At any given moment, there will be fourteen MOS switches connected to n1 and n2 for each capacitor-switch array. However, only one will be ON at a given time. Similarly, there will be 14 MOS connected to node 1 and 2, where 1 will be ON and 13 will be OFF.

## 6. Issues concerning our technique

1. As mentioned above, fourteen MOS switches sit on the critical nodes of the circuit. Though only one will be on at a time, all will add heavily to the capacitance of the node and increase the settling time and accuracy of the resultant capacitor. To alleviate this problem, one may use a more intelligent technique like multiplexer to reduce the number of MOS directly connected to the critical switch. Also, the size of

<sup>2</sup> In Fig. 6, 7 and 8, pmos have been drawn. These are actually nmos. The error is regretted.

the MOSFETs should be kept small, just more than the required for charging the capacitors and settling the op-amp in the clock frequency.

2. The capacitances of MOS would also bring about some coupling between the two points connected through an OFF MOS. This capacitance may change the behavior of the circuit and also lead to charge re-distribution through the 14 MOS capacitances (even if one is not significant, in total they may become significant).
3. In shrinking technologies, where leakage power becomes significant, this big array of switched of MOS on each node may sink considerable amount of current discharging capacitors and requiring refresh or high clock frequency.
4. In the present approach, we cannot scale down the capacitor sizes as we move to the stages representing LSB. This shall result in area and power penalty.
5. The array of MOSFETs required to control a capacitor will be immense (28\*14). Though this seems to be a huge area penalty, it isn't so. There are many techniques which use two FD-op-amps per stage to average out capacitor mismatch. A properly designed FD-opamp requires around 20 MOS devices where sizes could be much bigger than those here. Thus it can be concluded that this scheme shall approximately take the same area as by a capacitor-averaging scheme using two op-amps per stage.
6. The number of switches and area maybe decreased by using clever techniques informed by statistical data. For instance, one can use fixed scaled capacitors in the later stages and only use the above scheme with the first n stages, where n is decided through MATLAB simulations.

## **7. SPICE Implementation of Error Correction Scheme**

For SPICE simulations, it was assumed that the ordering of the capacitor will be already known. So the ordering of the capacitor was done using MATLAB, which were then plugged into SPICE subckts using a PERL script. [Refer to the MATLAB and Perl scripts in the Appendix]

For SPICE simulations, one should put the switches exactly how they would appear in the circuit. This would mean that each capacitor will be connected to all the 14 ADC stages through mos switches. This generates an extremely densely coupled network, which is computationally very expensive to simulate with SPICE.

To reduce this complexity, we did the following. For each capacitor, we connected node n1, n2, 1 and 2 each with 13 capacitors, apart from the one inside the circuit of the current stage. The other end of the switched-off transistors were connected to an arbitrary waveform oscillating between 0 to 1.8V generated using the pulse command. This waveform was expected to model the voltage variations that would occur on the other terminal of the MOS. This is shown in Fig. 7. Only two MOSFETs are shown instead of 13 for clarity. Here c1 is high, while c2 is low.

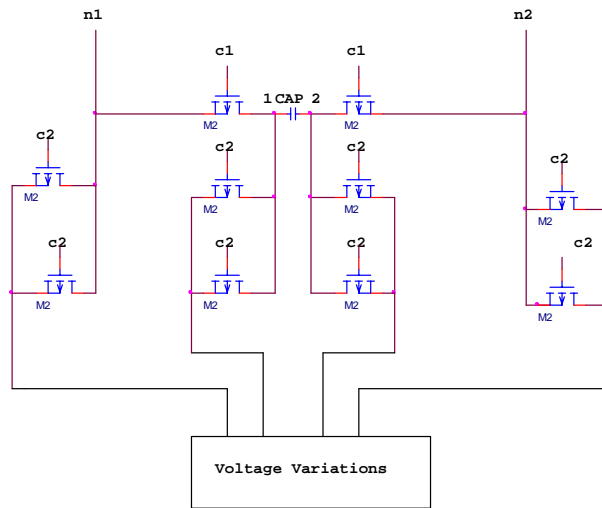


Figure 7: Modeling the capacitor switch array (Iteration 1)

The present approach was also computationally very expensive, since each capacitor had 27 MOSFETs around it and each stage had two such capacitors. To reduce the complexity further, we took a highly simplifying step. Since all these switches were in parallel, we replaced them with one switch which had the width as the sum of all the switches.<sup>3</sup> This is shown in Figure 8.

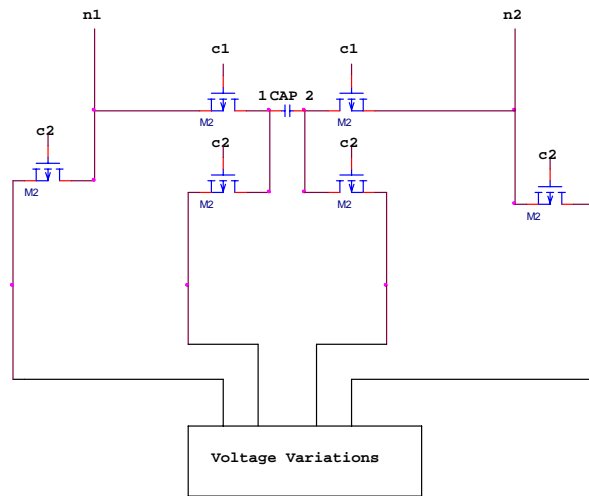


Figure 8: Modeling the capacitor switch array (Iteration 2)

We used a capacitor value of 0.5pF and a mosfet  $W=0.5\mu^4$  and  $L=0.18\mu$ . The total  $W$  of parasitic MOSFETs was set to  $13.5\mu$ .

<sup>3</sup> This is not equivalent in terms of layout parasitics which change when these many MOSFETs are replaced by a single mos. (routing parasitics, etc.)

<sup>4</sup> Though we have actually used ideal switches for all MOS switches in the multiplier (except the ones in the capacitor model, which are actual mos), we replaced all switches by MOS of  $0.5\mu$  to see if it settled. It settled in the required range of 100ns.

## 8. Simulation Results

Simulation was done around the three MSB transition points for the three ADCs: ADC with no capacitor mismatch, ADC with capacitor mismatch and ADC with capacitor mismatch with error connection.

Simulation was done across 7 points around the 3 MSB transitions with a least count of 1/2LSB. At some readings, the precision was further increased to get a better idea of the characteristics.

### 8.1 ADC with no capacitor mismatch

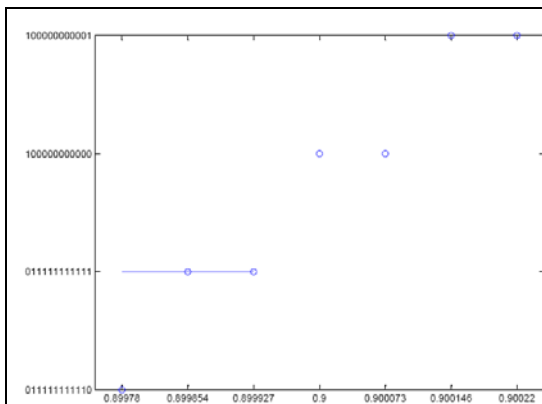
Results of simulation are tabulated in Table 1. It shows the input voltage and the corresponding digital output is shown. As it may be observed from the table, the ADC's behavior is decent and well-structured. Figure 9 shows 3 graphs for the MSB transitions. The X-axis represent the input voltage while the y-axis shows the digital output. A horizontal line has been drawn on each graph to indicate the size of 1 LSB.

From the table, it can be inferred that the INL is in the range of  $\pm 1\text{LSB}$ . One can say with surety that the DNL is in the range of  $\pm 1/2\text{LSB}$  for the given points. Due to the well-behaved nature of the curve, we can mathematically show that the step size is between 0.5 and 1.5. This can be concluded as follows. Consider 0111-1111-1110 to 0111-1111-1111 and 0111-1111-1111 to 1000-0000-0000 transition. The first transition here happens in a range of 0 to 0.5 LSB, then there is a constant code for 0.5 LSB and then the next transition also ranges between 0 to 0.5LSB. Thus the difference between the two transitions has to be between 0.5LSB to 1.5LSB. (This argument is on the assumption that the curve is monotonic between the points observed)

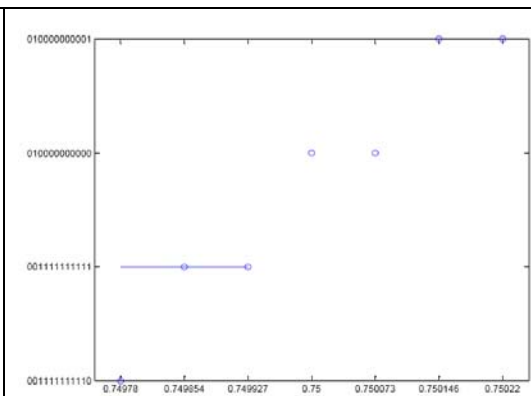
Table 1: Results for ADC with no capacitor mismatch

<i>Voltage Representation</i>	<i>Voltage</i>	<i>Code</i>
0.6	0.6	0000-0000-0000
0.9 – 1.5LSB	0.8997802734375000	0111-1111-1110
0.9 – LSB	0.8998535156250000	0111-1111-1111
0.9 – 0.5LSB	0.8999267578125	0111-1111-1111
0.9	0.9	1000-0000-0000
0.9 + 0.5LSB	0.90007324218750	1000-0000-0000
0.9 + LSB	0.9001464843750	1000-0000-0001
0.9 + 1.5*LSB	0.9002197265625	1000-0000-0001

0.75 – 1.5LSB	0.7497802734375	0011-1111-1110
0.75 – LSB	0.749853515625	0011-1111-1111
0.75 – 0.5LSB	0.7499267578125	0011-1111-1111
0.75	0.75	01000-0000-000
0.75 + 0.5LSB	0.75007324218750	01000-0000-000
0.75 + LSB	0.7501464843750	0100-0000-0001
0.75 + 1.5*LSB	0.7502197265625	0100-0000-0001
0.675 – 1.5LSB	0.6747802734375	0001-1111-1110
0.675 – LSB	0.674853515625	0001-1111-1111
0.675 – 0.5LSB	0.6749267578125	0001-1111-1111
0.675	0.68	0010-0000-000
0.675 + 0.5LSB	0.675073242187	0010-0000-000
0.675 + LSB	0.6751464843750	0010-0000-0001
0.675 + 1.5*LSB	0.6752197265625	0010-0000-0001
1.2	1.2	1111-1111-1111



a. Points around 0.9V



b. Points around 0.75V

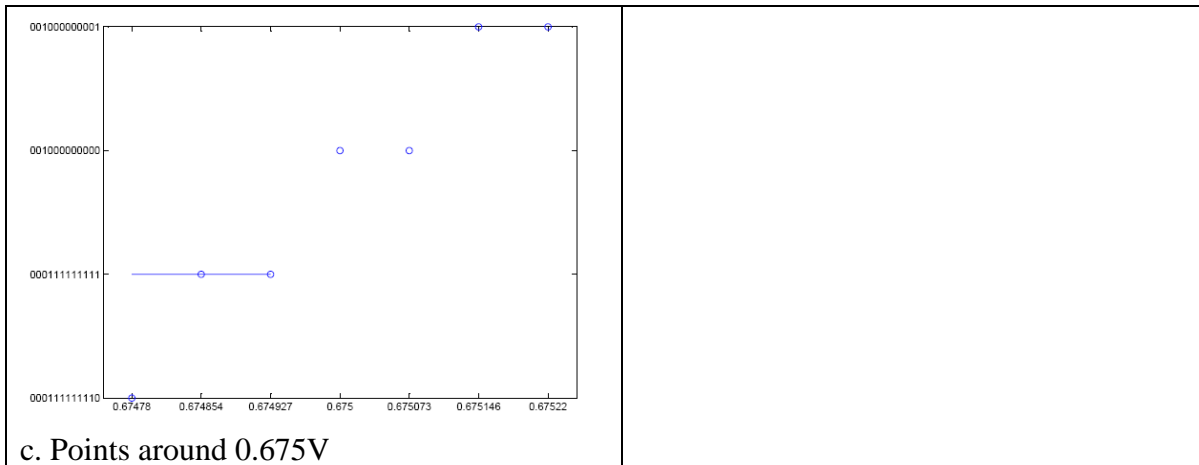


Figure 9: Graphs for MSB transition in ADC without capacitor mismatch

## 8.2 ADC with capacitor mismatch

ADC with capacitor mismatch was also simulated at the same points and an attempt was made to find its transition points, since they didn't lie on the ideal curve. We didn't spend much time analyzing its characteristics. Table 2 shows a small number of points to indicate the misbehaved nature of the curve.

Table 2: results for ADC with capacitor mismatch

<i>Voltage Representation</i>	<i>Voltage</i>	<i>Code</i>
0.6	0.6	0000-0000-0000
$0.9 + 1.5 * \text{LSB}$	0.9002197265625	0111-1111-1110
$0.9 + 2.0 * \text{LSB}$	0.90029296875000	0111-1111-1111
$0.9 + 3.0 * \text{LSB}$	0.9003662109375	0111-1111-1111
$0.9 + 3.5 * \text{LSB}$	0.900439453125	1000-0000-0000
$0.9 + 4.0 * \text{LSB}$	0.9005126953125	1000-0000-0000
$0.75 - 5.5 * \text{LSB}$	.749194335937	0100-0000-0111
1.2	1.2	1-0000-0000-0011
$1.2 - 1.5 \text{LSB}$	1.1997802734375	1-0000-0000-0000

## 8.3 ADC with capacitor mismatch with correction

The results have been tabulated in table 3 and graphs have been plotted in Figure 10.

One may observe that the DNL is in the range of  $\pm 1\text{LSB}$ . Though one can say with confidence that the DNL is in the range of  $\pm 0.5\text{LSB}$  for the points around 0.675V, we can only make a claim that the DNL around 0.9V and 0.75V is in the range of  $\pm 1\text{LSB}$ . To find out whether it is in 0.5LSB range, we need to look at values between the current considered values. A binary search in-between the points can then be done to find out the range of DNL, however it is not clear if such a search is bounded for the number of iterations required. We did one iteration of binary search to give a better estimate of the situation. The addition values considered are marked in grey in the table.

One may deduce after finding the new values, that the step size lies between 0.5 and 1.5, and thus the DNL is in the specified range. This may be easily concluded from the table using the methodology in Section 8.1.

Table 3: Results for ADC with capacitor mismatch with correction

<i>Voltage Representation</i>	<i>Voltage</i>	<i>Code</i>
0.6	0.6	0000-0000-0000
0.9 – 1.5LSB	0.8997802734375000	0111-1111-1110
0.9 – 1.25LSB	.89981689453125000000	0111-1111-1111 (not reqd)
0.9 – LSB	0.8998535156250000	0111-1111-1111
0.9 – 0.5LSB	0.8999267578125	0111-1111-1111
0.9 – 0.25LSB	.89996337890625	1000-0000-0000
0.9	0.9	1000-0000-0000
0.9 + 0.25LSB	.900036621093750	1000-0000-0000
0.9 + 0.5LSB	0.90007324218750	1000-0000-0001
0.9 + LSB	0.9001464843750	1000-0000-0001
0.9 + 1.25*LSB	0.90018310546875	1000-0000-0001 (not reqd)
0.9 + 1.5*LSB	0.9002197265625	1000-0000-0010
0.75 – 1.5LSB	0.7497802734375	0011-1111-1110
0.75 – LSB	0.749853515625	0011-1111-1111

0.75 – 0.5LSB	0.7499267578125	0011-1111-1111
0.75 – 0.25LSB	.74996337890625	0100-0000-0000
0.75	0.75	0100-0000-0000
0.75 – 1.25LSB	.75003662109375	0100-0000-0000
0.75 + 0.5LSB	0.75007324218750	0100-0000-0001
0.75 + LSB	0.7501464843750	0100-0000-0001
0.75 + 1.5*LSB	0.7502197265625	0100-0000-0001
0.75 + 1.75*LSB	0.750256340	0100-0000-0010
0.75 + 2.00*LSB	.75029296	0100-0000-0010
0.675 – 1.5LSB	0.6747802734375	0001-1111-1110
0.675 – LSB	0.674853515625	0001-1111-1111
0.675 – 0.5LSB	0.6749267578125	0001-1111-1111
0.675	0.675	0010-0000-0000
0.675 + 0.5LSB	0.675073242187	0010-0000-0000
0.675 + LSB	0.6751464843750	0010-0000-0001
0.675 + 1.5*LSB	0.6752197265625	0010-0000-0001
1.2	1.2	1-0000-0000-0000
1.2 – 0.5LSB	1.1999267578125	1111-1111-1111
1.2 - LSB	1.1998535156250	1111-1111-1111

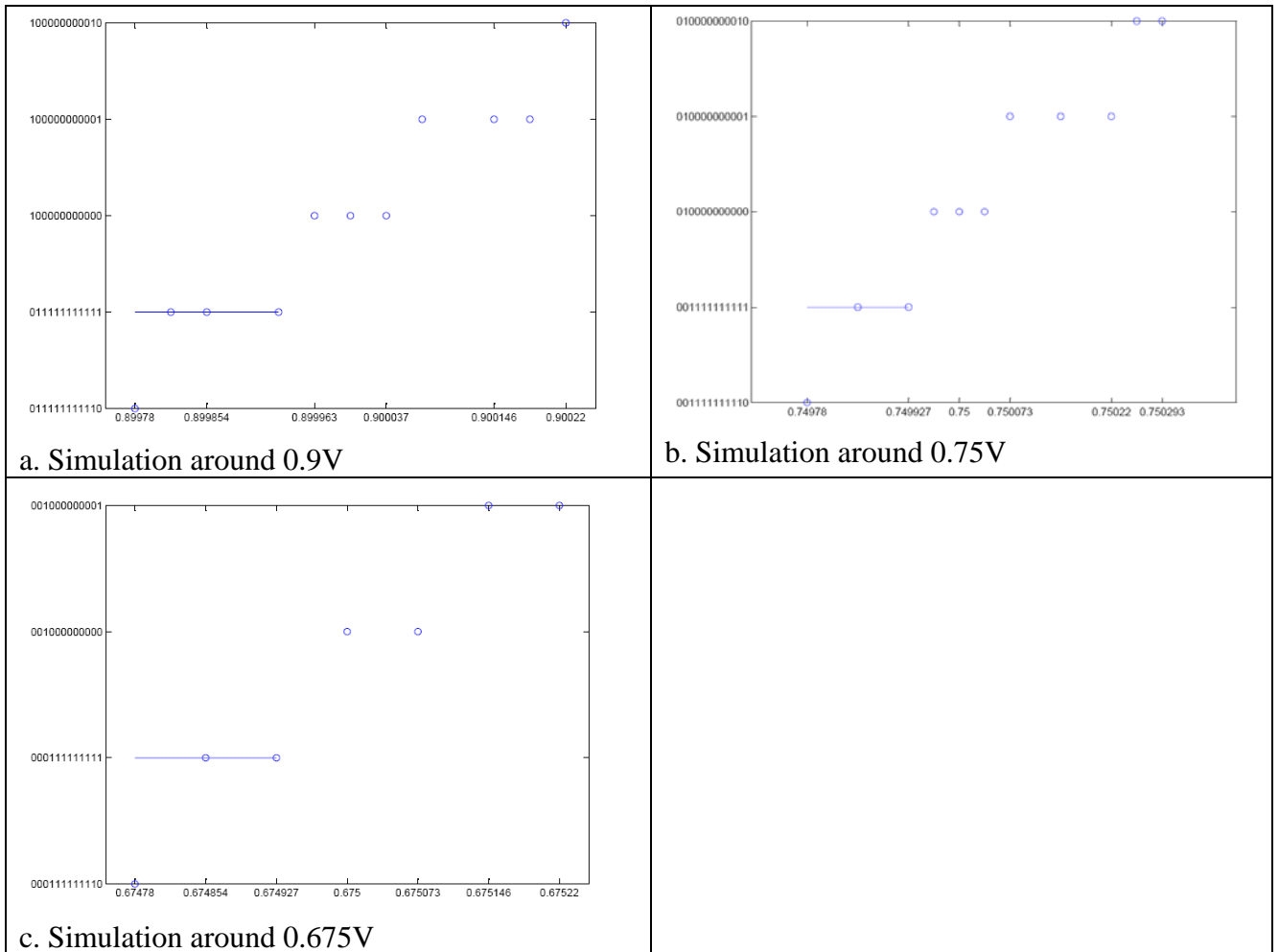


Figure 9: Graphs for MSB transition in ADC with capacitor mismatch and correction

## 9. Figure of Merit (FOM)

For calculating the FOM, we only consider the quantization noise for this project. For the total power, we sum up the power of all opamps. Though our comparator uses the first stage of our opamp and shall draw considerable power, we assume that in a real scenario, a low power comparator (dynamic comparator) can be used.

Total power: 14X 98.1uW  
 ENOB = Number of Bits = 12  
 $f_s = 5 \text{ MHz}$

FOM = 0.006706pJ

## 10. Conclusion

A 12-bit ADC with capacitor mismatch was designed and simulated to give an INL and DNL of  $\pm 1 \text{ LSB}$  and  $\pm 1/2 \text{ LSB}$ . A new technique to handle capacitor mismatch was

proposed and tested. It meets the performance criteria in present experiments. The issues and suggestions of improvement for the current approach have been summarized in Section 7. Future work maybe motivated in the following directions.

- The exact capacitor measurement technique and the composition of the digital block have not been dealt with in this project. They should be investigated.
- More accurate simulations are required to be confident about this approach. A rough model of the capacitor with mos-switches has been used to decrease simulation time. An exact model should be used to do simulations. Also, extra connections to the capacitor for measurement circuitry has been ignored.
- To improve this technique one can consider better switch arrays to plug-in and plug-out capacitors which decrease the parasitic capacitance on the node (for e.g., multiplexer). Also, one can be informed by statistical data to get the scheme working with lesser number of switches.

## APPENDIX

\*The master file which contains the ADC structure

\* Different subcircuits are shown for different stages, all these stages have different capacitor values due to mismatch. Same subckt is used for ADC without mismatch

\* Two representative measure statements are shown. For each simulation 7 voltage conversions can be measured.

```
.include stage1_2  
.include stage1_5  
.include stage1_8
```

```
.include stage_1  
.include stage_10  
.include stage_11  
.include stage_12  
.include stage_2  
.include stage_3  
.include stage_4  
.include stage_5  
.include stage_6  
.include stage_7  
.include stage_8  
.include stage_9  
.include cap_switch_subc
```

\* Vin Vout Vbit V+ V- Vc Vr1 Vr2 Clk1 Vclk2

```
x1 40 41 60 13 0 10 50 51 3 31 stage_1  
x2 41 42 61 13 0 10 50 51 31 3 stage_2
```

```
x2a 42 421 62 13 0 10 50 10 3 31 stage1_2
```

```
x3 421 43 621 13 0 10 50 51 31 3 stage_3  
x4 43 44 63 13 0 10 50 51 3 31 stage_4  
x5 44 45 64 13 0 10 50 51 31 3 stage_5
```

```
x51 45 451 65 13 0 10 50 10 3 31 stage1_5
```

```
x6 451 46 651 13 0 10 50 51 31 3 stage_6  
x7 46 47 66 13 0 10 50 51 3 31 stage_7  
x8 47 48 67 13 0 10 50 51 31 3 stage_8
```

```
x8a 48 481 68 13 0 10 50 10 3 31 stage1_8
```

```
x9 481 49 681 13 0 10 50 51 31 3 stage_9  
x10 49 410 69 13 0 10 50 51 3 31 stage_10  
x11 410 411 610 13 0 10 50 51 31 3 stage_11  
x12 411 412 611 13 0 10 50 51 3 31 stage_12
```

```
.param v1= .8997802734375000  
.param v2= 0.8998535156250000000  
.param v3= 0.8999267578125000  
.param v4= 0.9  
.param v5= 0.90007324218750  
.param v6= 0.9001464843750000000
```

```
.param v7= 0.90021972656250000000
```

```
Vin 40 0 PWL(
```

```
+0 v1  
+ 300n v1  
+ 300.1n v2  
+ 500.0n v2  
+ 500.1n v3  
+ 700.0n v3  
+ 700.1n v4  
+ 900.0n v4  
+ 900.1n v5  
+1100.0n v5  
+1100.1n v6  
+1300.0n v6  
+1300.1n v7)
```

```
Vdd 13 0 1.8v
```

```
Vcm 10 0 dc 0.9
```

```
Vref1 50 0 1.2V
```

```
Vref2 51 0 0.6V
```

```
*Vin 40 0 PULSE(vin vin 0n 0n 0n 400n 800n)
```

```
Vclk 3 0 PULSE(1.8 0 0n 0n 0n 102n 200n)
```

```
Vclk1 31 0 PULSE(0 1.8 2n 0n 0n 98n 200n)
```

```
.measure tran bit1 find V(60) At=250ns  
.measure tran bit2 find V(61) At=350ns  
.measure tran bit3 find V(62) At=450ns  
.measure tran bit31 find V(621) At=550ns  
.measure tran bit4 find V(63) At=650ns  
.measure tran bit5 find V(64) At=750ns  
.measure tran bit6 find V(65) At=850ns  
.measure tran bit61 find V(651) At=950ns  
.measure tran bit7 find V(66) At=1050ns  
.measure tran bit8 find V(67) At=1150ns  
.measure tran bit9 find V(68) At=1250ns  
.measure tran bit91 find V(681) At=1350ns  
.measure tran bit10 find V(69) At=1450ns  
.measure tran bit11 find V(610) At=1550ns  
.measure tran bit12 find V(611) At=1650ns
```

```
.measure tran bbit1 find V(60) At=450ns  
.measure tran bbit2 find V(61) At=550ns  
.measure tran bbit3 find V(62) At=650ns  
.measure tran bbit31 find V(621) At=750ns  
.measure tran bbit4 find V(63) At=850ns  
.measure tran bbit5 find V(64) At=950ns  
.measure tran bbit6 find V(65) At=1050ns  
.measure tran bbit61 find V(651) At=1150ns  
.measure tran bbit7 find V(66) At=1250ns  
.measure tran bbit8 find V(67) At=1350ns  
.measure tran bbit9 find V(68) At=1450ns  
.measure tran bbit91 find V(681) At=1550ns  
.measure tran bbit10 find V(69) At=1650ns
```

```
.measure tran bbit11 find V(610) At=1750ns
.measure tran bbit12 find V(611) At=1850ns
```

```
.tran 0.05n 2400n
.options reltol=1e-20 chgtol=1e-20
```

```
.end
```

```
*****
```

```
*SPICE file for Stage of gain 2 (model for MOS removed),
* File for stage of gain 1 is similar
* Invokes model of capacitor, Stage with ideal caps just use simple capacitor declaration statement
```

```
.subckt stage_1 5 2 150 13 0 10 50 51 3 31
```

```
xi1 1 10 2 13 0 opamp1
```

```
.include cascode_subc
.include comp_var_subc
.include cap_switch_subc
```

```
* Invokes the model of capacitor
```

```
xc3 6 1 capa cv=0.503871p
xc4 1 4 capa cv=0.503826p
```

```
Gm1 4 5 VCR NPWL(1) 3 0 0.5,1000g 0.6,500 1.8,0
Gm2 6 5 VCR NPWL(1) 3 0 0.5,1000g 0.6,500 1.8,0
Gm4 1 2 VCR NPWL(1) 3 0 0.5,1000g 0.6,500 1.8,0
Gm3 4 11 VCR NPWL(1) 31 0 0.5,1000g 0.6,500 1.8,0
Gm5 6 2 VCR NPWL(1) 31 0 0.5,1000g 0.6,500 1.8,0
Gmc1 51 11 VCR NPWL(1) 150 0 0.0,0 1.2,500 1.3,1000g
Gmc2 11 50 VCR NPWL(1) 150 0 0.5,1000g 0.6,500 1.8,0
```

```
xcomp 5 10 150 13 0 3 31 comp
```

```
.ends
```

```
*****
```

```
*Subcircuit for capacitor model (model for mos removed)
```

```
.subckt capa 1 2 cv=0.5p
```

```
Vc1 c1 0 1.8V
Vc2 c2 0 0
v1 11 0 PULSE(0 1.8 2n 0n 0n 10n 20n)
v2 22 0 PULSE(0 1.8 0n 0n 0n 10n 20n)
.param cv=0.5p
c1 n1 n2 cv
```

```
Ma1 n1 c1 1 1 CMOSN w=0.5u l=0.18u
Ma2 n2 c1 2 2 CMOSN w=0.5u l=0.18u
```

```
Ma10 n1 c2 11 11 CMOSN w=6.5u l=0.18u
Ma20 n2 c2 22 22 CMOSN w=6.5u l=0.18u
```

```
Ma11 1 c2 11 11 CMOSN w=6.5u l=0.18u
Ma21 2 c2 22 22 CMOSN w=6.5u l=0.18u
```

.ends

\*SPICE files for comparator and opamp have not been added due to space constraints. They maybe requested.

MATLAB file for generating best ordered capacitors

```
%
X=[]
C=[]
A1 = randn(14,1)*(.0025/((0.5)^0.5)) + 0.5
B1 = randn(14,1)*(.0025/((0.5)^0.5)) + 0.5
A = A1;
B = B1;
C = [A1 B1];
for k=1:14
    bestdiff=1
    for i=1:(14 - k+1)
        for j=1:(14 - k +1)
            if( abs(A(i) - B(j))/min(A(i),B(j))<bestdiff)
                bestdiff = abs(A(i) - B(j))/min(A(i),B(j));
                best_i = i;
                best_j = j;
                X(k,1) = A(i);
                X(k,2) = B(j);
            end
        end
    end
    A(best_i) = [];
    B(best_j) = [];
end

error = abs(X(:,1) - X(:,2))./(min(X'))'
L = 1./2.^(14:-1:1);
error = [error L]
E = error(:,1) > error(:,2)

for i=1:14
    for j=1:2
        fprintf( 1, "%f",',', C(i,j))
    end
end
fprintf( 1, '\n\n')

for i=1:14
    for j=1:2
        fprintf( 1, "%f",',', X(i,j))
    end
end
end
```

Perl Script for plugging capacitor values in subckt

```
## Values copied from MATLAB output
```

```
@caps
```

```
=( "0.503871", "0.503826", "0.498665", "0.498759", "0.500419", "0.500311", "0.501514", "0.501378", "0.503167", "0.503327", "0.502043", "0.502203", "0.502584", "0.502825", "0.499173", "0.499535", "0.500143", "0.500750", "0.502011", "0.501113", "0.502394", "0.500841", "0.498954", "0.497376", "0.494785", "0.496437", "0.499096", "0.496492", "0.493374", "0.505104" );
```

```
$gain2template = "stage_subc.bak";
```

```
$gain1template = "stage1_subc.bak";
```

```
$c = 0;
```

```
for ($i=1; $i<=12;$i++)
```

```
{
```

```
  open(IFILE,$gain2template);
```

```
  @lines = <IFILE>;
```

```
  close(IFILE);
```

```
  unlink("stage_$$i");
```

```
  print $caps[$c];
```

```
  print $caps[$c+1];
```

```
  open(OFILE,">stage_$$i");
```

```
  for ($r = 0; $r <= $#lines; $r++)
```

```
  {
```

```
    $lines[$r] =~ s/stage/stage_$$i/;
```

```
    $lines[$r] =~ s/cap1/$caps[$c]/;
```

```
    $lines[$r] =~ s/cap2/$caps[$c+1]/;
```

```
    print OFILE $lines[$r];
```

```
  }
```

```
  close(OFILE);
```

```
  $c = $c+2;
```

```
if ($i == 2 || $i == 5 || $i == 8)
```

```
{
```

```
  open(IFILE,$gain1template);
```

```
  @lines = <IFILE>;
```

```
  close(IFILE);
```

```
  unlink("stage1_$$i");
```

```
  open(OFILE,">stage1_$$i");
```

```
  for ($r = 0; $r <= $#lines; $r++)
```

```
  {
```

```
    $lines[$r] =~ s/stage1/stage1_$$i/;
```

```
    $lines[$r] =~ s/cap1/$caps[$c]/;
```

```
    $lines[$r] =~ s/cap2/$caps[$c+1]/;
```

```
    print OFILE $lines[$r];
```

```
  }
```

```
  close(OFILE);
```

```
  $c = $c+2;
```

```
}
```

```
}
```

