UiO : University of Oslo

Sequential 3D Cyclic ADC

Master Thesis

Sebastian E. Pedersen Wood

15 January 2021

Contents

List of Figures

List of Tables

1 Introduction

The project will investigate CMOS 3D sequential integration to design integrated circuits in 3D rather than the traditional 2D. 3D integration is very well suited for mixed-signal circuits as the CMOS transistors of different technologies can be stacked in different tiers and have a very fine grained interconnection between them. The project will utilize the 3D integration to design a Cyclic/Algorithmic analog-to-digital converter (ADC) to find advantages and disadvantages, as well as gaining experience in 3D design techniques.

The 3D-MUSE project offered a single tape-out for its participants and has an exceedingly long production time. The tape-out deadline was originally set in May 2019, but changed multiple times throughout the project. The final deadline was in November 2019. Due to this early deadline, this thesis was somewhat split in two. The first half consisted of designing for a time restricted tape-out, and was by necessity restricted to simple design blocks. The tape-out designs are therefore incomplete. The other half of the project was then a redesigned and completed implementation of the cyclic ADC, carried out with simulations and partial layout without a tape-out. It also worth mentioning that the 3D-MUSE project uses experimental production techniques, the lead time is quite longer that usual. The die is not expected to be delivered before this thesis' deadline and thus physical testing will have to be completed outside the scope of this thesis.

The application, or use case, for a 3D integrated cyclic ADC in the 3D-MUSE project is in sounding rockets/space probes and Langmuir probes. A Langmuir probe is a device with a small conductor that can be inserted into plasma to collect ion or electron currents that flow to it in response to different voltages [1]. It is used to determine the temperature and density of electrons in the plasma. A SAR ADC has previously been implemented in a different process for this application, but it is believed that a cyclic ADC might be a better and definitely a more space conservative solution.

In this thesis, the 3D technology will be described and how a cyclic ADC can be implemented in this technology.

The design and implementation of a switch-capacitor cyclic ADC is then described, where the shortcomings of the design are also mentioned. This implementation of the ADC was taped-out early in the project..

A new design, based on a different switch-capacitor implementation is then introduced. This improvement utilizes the different tiers more methodically and has a higher accuracy. The new design is simulation only, with partial layout.

In the discussion chapter the benefits of moving to 3D is discussed, as well as thoughts on future improvements. Improvements include a different division of the different tiers and implementing a differential design.

1.1 3D sequential integration

3D sequential integration is the topic of a 3D-MUSE project, that UiO are involved in together with partner CEA/LETI in France [2]. It promises to allow 3D CMOS designs where transistors are no longer restricted to just a 2D plane, but can be stacked on top of eachother. In contrast to 'parallel' 3D integration that stacks several 2D wafers on top of each other using relatively huge and few vertical connections between the tiers, 'sequential 3D integration' builds the CMOS tiers on a single substrate using photolitography [3]. This way, the density of vertical connections is of the same order of magnitude as the horizontal connections in 2D. So transistors of a circuit module can be placed freely in any tier with no significant disadvantages. This offers new opportunities to make tiers with different types of transistors and thus to make circuits that include different types of transistors, e.g transistors that are optimized for the digital signal domain in one and transistors that are optimized for the analog signal domain in another. Mixed signal circuits can therefore be designed with highly optimized components, as illustrated by this conceptual recursive SAR ADC in figure 1.1.

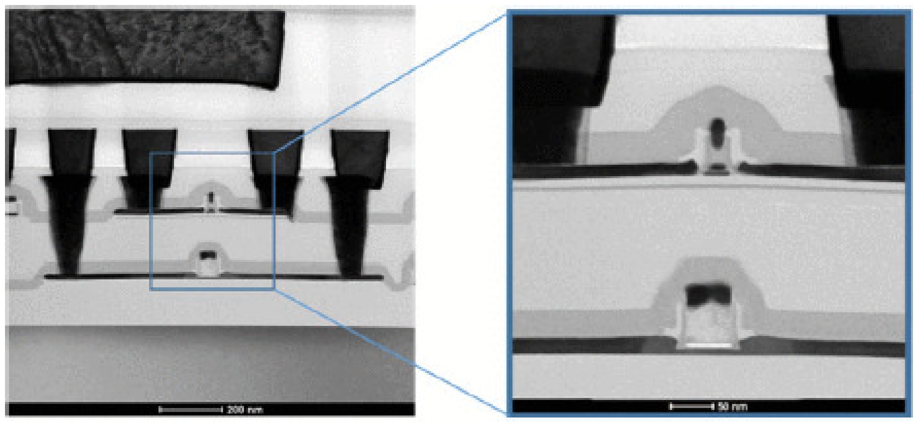


Figure 1.1: Cross-section of the 3D sequential structure in a SAR ADC.

The state of the art 3D parallel integration allows a interconnection pitch in the order above 1µm, which gives a density less than 10^6 connections per mm^2. In our current process we have about 200nm pitch, or $2.5 \cdot 10^7$ connections per mm^2.

Optimized devices are not the only advantage in this design process. In general, due to 3D placement the total interconnect wiring will be reduced enhancing speed and power efficiency. Furthermore, the LETI 3D sequential integration process allows to have a ground plane between the tiers, insulating the analog from the digital domain for reduced cross-talk noise.

Figure 1.2 shows the process flow of the 3D integration and how it stacks active device layers on top of each other in a sequential matter [4]. This process of 3D implementation differs from 3D packaging where the tiers are fabricated in parallel and later stacked and bonded to each other. The 3D sequential integration offers more and unique connectivity opportunities between the layers because the top active patterning is defined by Process of Reference lithography. Compared to traditional 3D packaging where the alignment accuracy and feature size of inter-tier interconnections and stacked tiers are limited by bonding alignment accuracy, 3D sequential integration are only limited by stepper resolution[5].

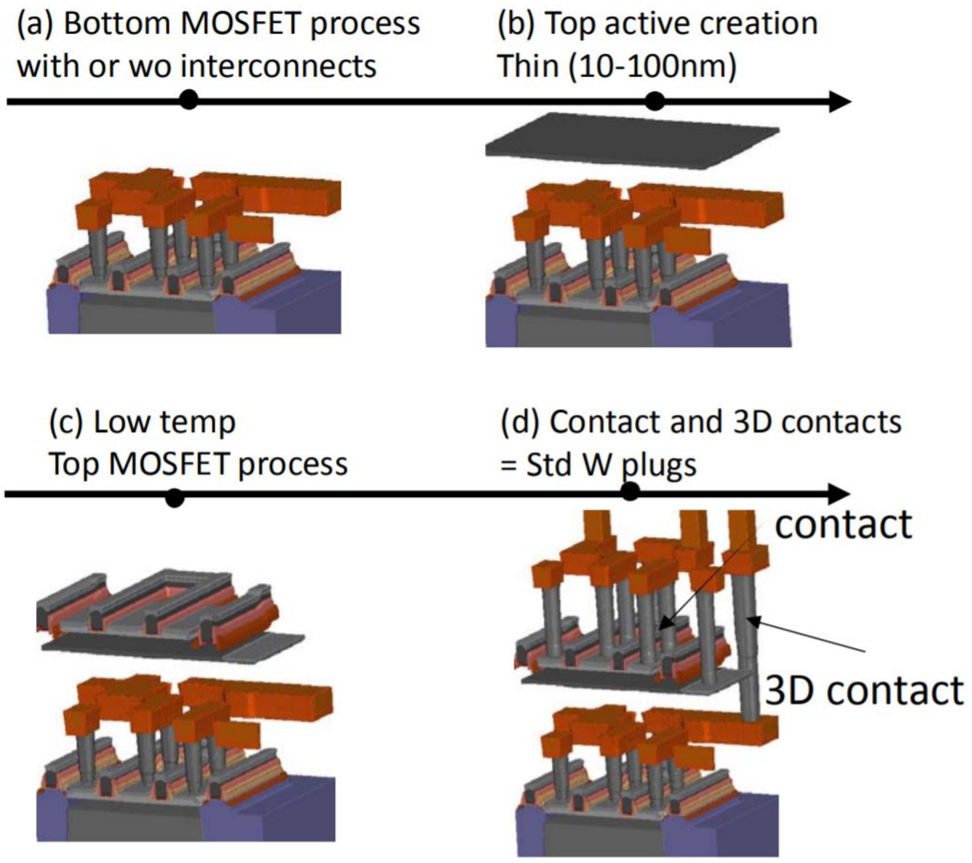


Figure 1.2: Process flow of the 3D sequential structure.

As the 3D sequential integration technology is not fully developed, the bottom tier has a higher chance of success as this tier is produced traditionally. The top tier, however, is produced at a much lower temperature. Given that the top layer is at an experimental stage in production, the chances of success for each transistors is lower than the bottom layer.

In the process design kit for this project, the bottom layer technology is 28nm FDSOI and the top layer is 65nm shrink technology.

1.2 Analog to digital converter

Analog to digital converters tie together the analog world and digital systems.
The input of an ADC can be assumed to be an infinite number of values, but the output can be selected from only a finite set of codes given by the converter's output resolution [6]. Due to the ADCs resolution, and its conversion speed, AD converters often appear as a bottleneck in applications for data processing as it limits systems overall precision or speed.

An ADC executes three operations, sampling the analog input signal, quantizing the sampled signal and assigning a digital code to the quantized output. For an ideal ADC, the transfer characteristic is a uniform staircase function where an analog input signal with infinitely large resolution can be 'mapped' to an N-bit staircase. The input-output characteristics of a 3bit ideal ADC is show below.

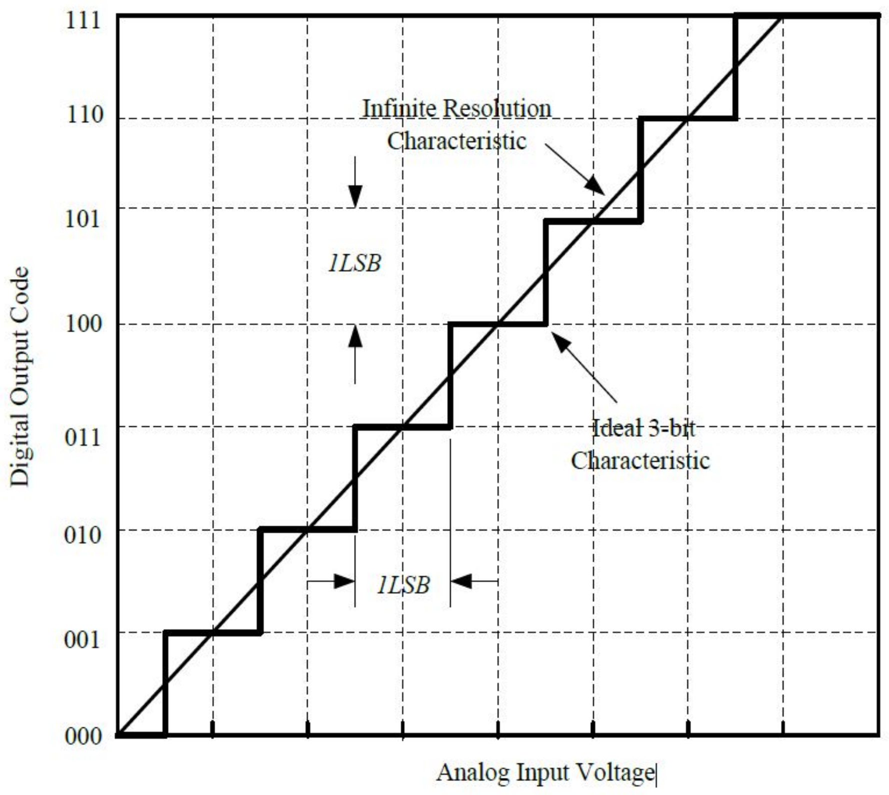


Figure 1.3: Ideal input-output characteristics of a 3bit ADC.

ADCs can be designed with several different architectures, that have their individual advantages and disadvantages. The principle trade-offs in an ADC is conversion rate/time, resolution and energy consumption. However, some architectures increase their die area with a higher resolution. A comparison of some architectures and their respective resolution and conversion rate is shown below.

Table 1: Classification of ADC architectures.

Conversion rate	**ADC architecture**	**Resolution**
Slow	Integrating, incremental	Very high resolution >16bits
Medium	Successive approximation 1bit pipeline Cyclic	Moderate resolution >10bits
Fast	Flash Folding Interpolating	Low resolution >6bits

Among these ADCs, the cyclic ADC is known for its capability to achieve medium resolution while requiring a small silicon area and whilst having a reasonable conversion [7]. In some ways, cyclic ADC is similar to SAR ADC. The most common feature of cyclic ADC and SAR ADC is that both of them need n clock cycles to finish one conversion. The most uncommon difference is that cyclic ADC uses a simpler DAC, but with several constant reference voltages, unlike a large DAC with capacitor array in SAR ADC. Generally, a cyclic ADC benefits from high-speed, low-power operation while maintaining relatively small chip area, compared with SAR ADC.

Compared with other ADCs, the cyclic ADC is superior in overcoming the difficulty of compatibility between ADC speed and bit resolution, while maintaining low-noise and high dynamic range [7]. It generally has lower hardware complexity than the other ADCs [8].

These characteristics makes the cyclic ADC a good candidate for 3D sequential integration and for the intended use of implementing the ADC in sensor front ends in rockets and/or space probes.

1.3 Cyclic ADC

A cyclic, or algorithmic, ADC is a converter that uses one cycle for one bit conversion. This means that the output of one cycle is fed back to the input, meaning that the conversion requires N cycles for an N bit conversion (N being the resolution of the ADC). In each cycle, the input signal is sampled to determine the bit and the residue voltage is generated by either adding or subtracting a reference voltage and doubling the resulting voltage [9]. The operation of a cyclic ADC is shown in figure 1.4.

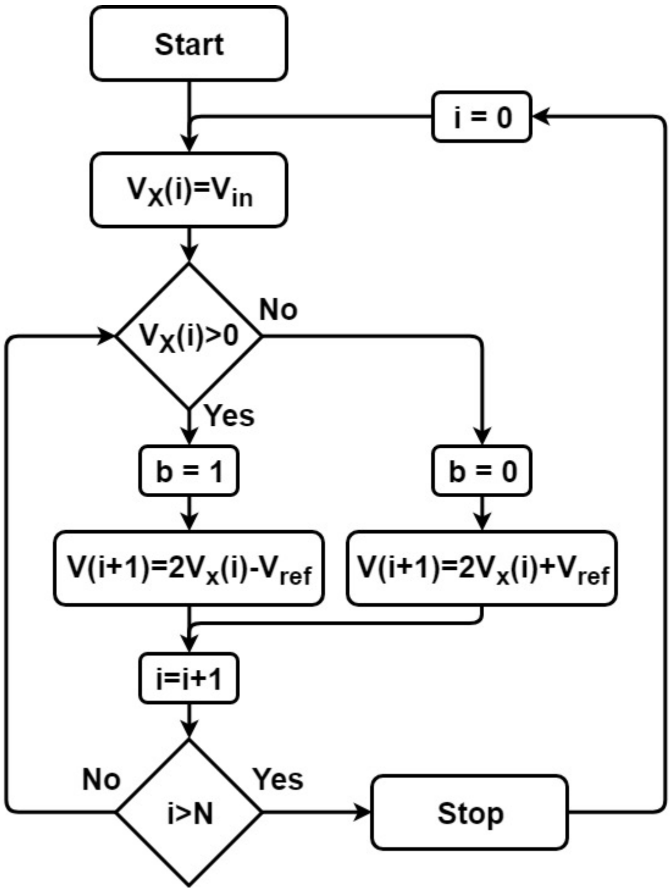


Figure 1.4: Flowchart of a cyclic ADC.

The ADC consists of a multiply-by-2 amplifier, sample and hold circuit (S/H), a comparator and a adder/subtractor circuit for the reference voltage. After a cycle the same S/H circuit samples the residue voltage given by the adder/subtractor and a new cycle begins. The input signal is disabled until the conversion is complete. A block diagram of the ADC is shown in figure 1.5.

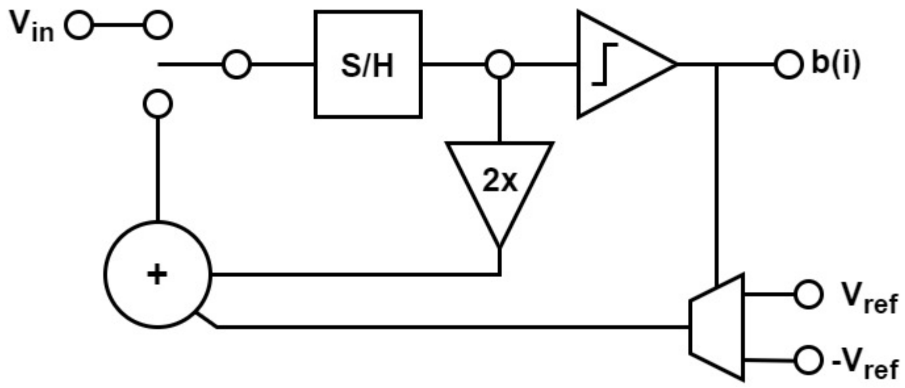


Figure 1.5: Simplified block diagram of the cyclic ADC.

The conversion shown in the figure above can be expressed mathematically by the following discrete-time equations:

$$V(1) = V_{IN}$$

$$V(i+1) = 2 \cdot V(i) + (-1)^{b(i)} \cdot V_{REF} \tag{1}$$

where

$$\begin{aligned} b(i) &= 1, \quad if\ V(i) \geqslant 0 \\ b(i) &= 0, \quad if\ V(i) < 0 \end{aligned} \tag{2}$$

The key operations in the conversion are therefore multiplication by two, addition and subtraction.

Conventional Cyclic/Algorithmic ADCs are often subject to conversion errors because of inaccuracies in the residue voltages. This residue error might occur if the comparator offset is too high and/or because of loop-gain non-idealities. The gain of the amplifier and capacitor mismatch also needs to be accurate to obtain low errors. Because the residue error gets amplified, the error will increase for each cycle. This will increase the conversion error. Overall accuracy is then usually limited by the last conversion cycle.

2 Tapeout

Because CMOS 3D integration allows for mixed-signal circuits, the analog and digital components can be placed in the different tiers. The components can also be mixed between the tiers for maximum space efficiency.

As the transistors in the bottom tier technology were expected to have a higher yield, the analog components are placed in the bottom tier and the digital in the top. The bottom tier is a standard industrial process with industrial level yield, while the top is experimental and is expected to not conform closely to the simulation models and even transistor failures can occur. Some components were not ready before the deadline, and needs to be implemented off-chip. The components that was not ready were the amplify-by-two and the adder/subtractor circuit.

The comparator and S/H circuit was implemented in the bottom layer. The counter was implemented in both the top and bottom layer. The multiplexers were in the top layer.

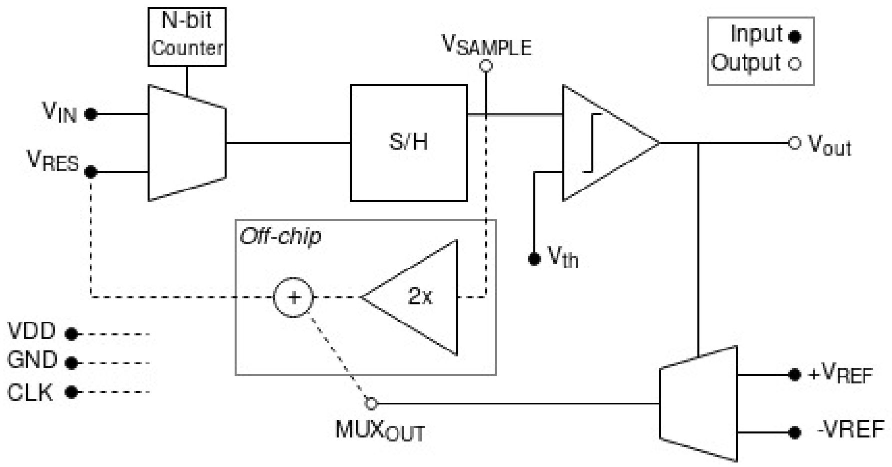


Figure 2.6: Illustration of the ADC core and which cells that have been included, and excluded, in the tape-out.

The schematics and layout are explained and shown in the following subchapters.

2.1 Comparator

The comparator is based on a modified strongARM latch design by Behzad Razavi [10]. This design consumes zero static power and directly produces rail-to-rail outputs. It is however clocked/synchronous and needs to be timed correctly for it to work properly. The schematic of the comparator is shown in 2.9 below.

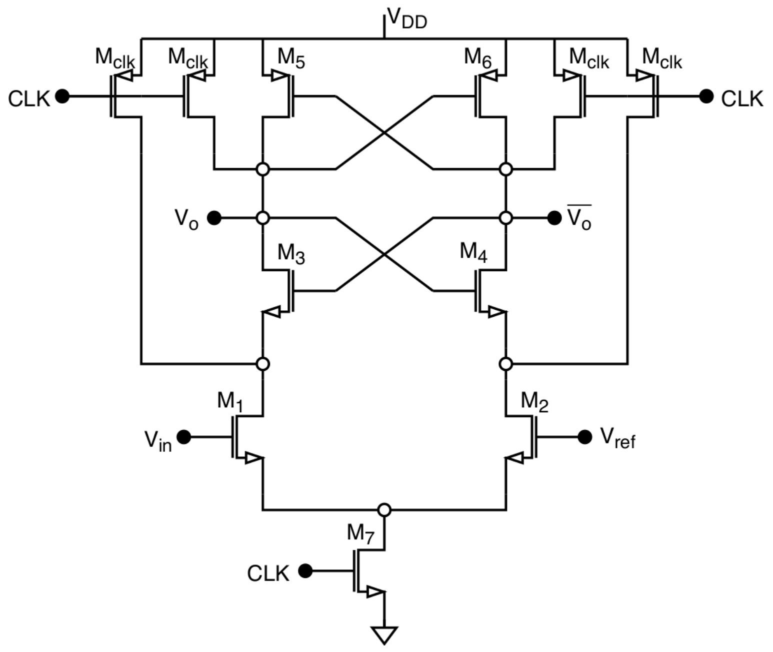


Figure 2.7: Schematics of the comparator.

The comparator was simulated by applying a rising voltage around the reference voltage to the input to find the resolution. By increasing the clock frequency until the comparator no longer can accurately compare the input to the reference voltage, the maximal sample rate and decision speed is found. The resolution of the comparator is the ΔV_{in} in figure 2.8 which is larger than 14bit, as stated by the equation below.

$$2^n = \frac{V_{range}}{\Delta V_{in}} = \frac{1V}{500.036mV - 499.996mV} \tag{3}$$

The specifications of the comparator is shown in the table below.

Table 2: Comparator specifications.

Parameter	**Value**
V_{DD}	1V
Input range	0-1V
Vref	0.5V
Sample rate	10 MS/s
Decision speed	<1ns
Resolution	14bit
Transistor	**Size (W/L)**
$M_{1,2}$	70
$M_{3,4}$	40
$M_{5,6}$	40
M_7	150
M_{CLK}	20

The simulation results of the comparator with a rising input voltage and a clock frequency of 1GHz is shown in figure 2.8. The two cursors show how the comparator correctly toggles the output when the input voltage exceeds the reference voltage. When running the same simulation with a higher clock frequency and/or with a slower rise time on the input voltage, the output of the comparator gave false negatives.

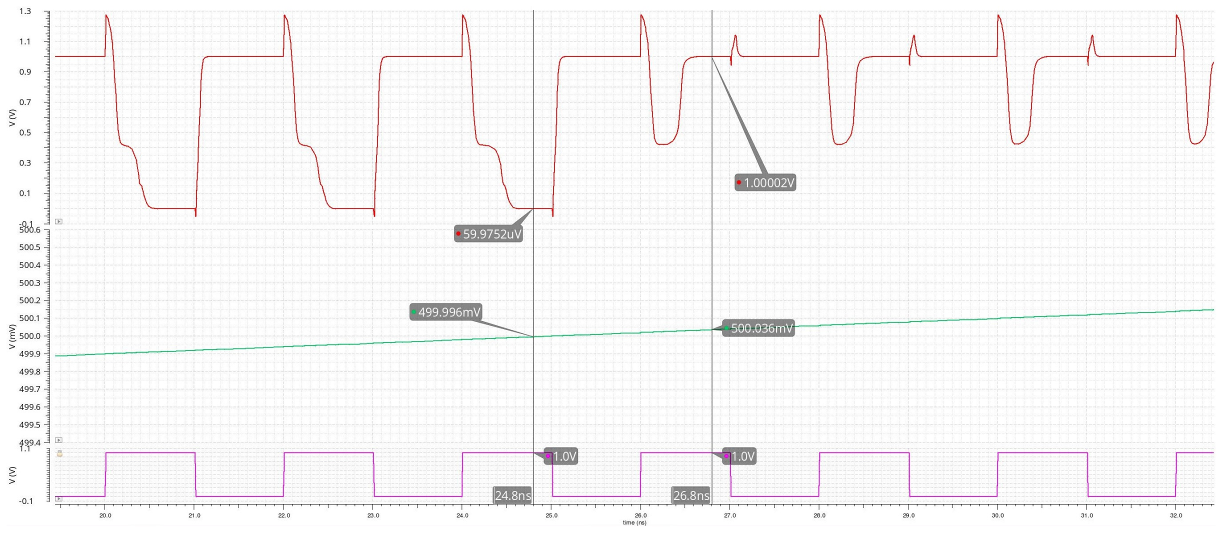


Figure 2.8: Simulation results of the comparator with a rise in V_{in} around V_{ref}=500mV.

Figure 2.9 below shows the layout of the comparator. The comparator is 6.11μm wide and 5.77μm tall. To increase accuracy, the layout was made to be as symmetric as possible.

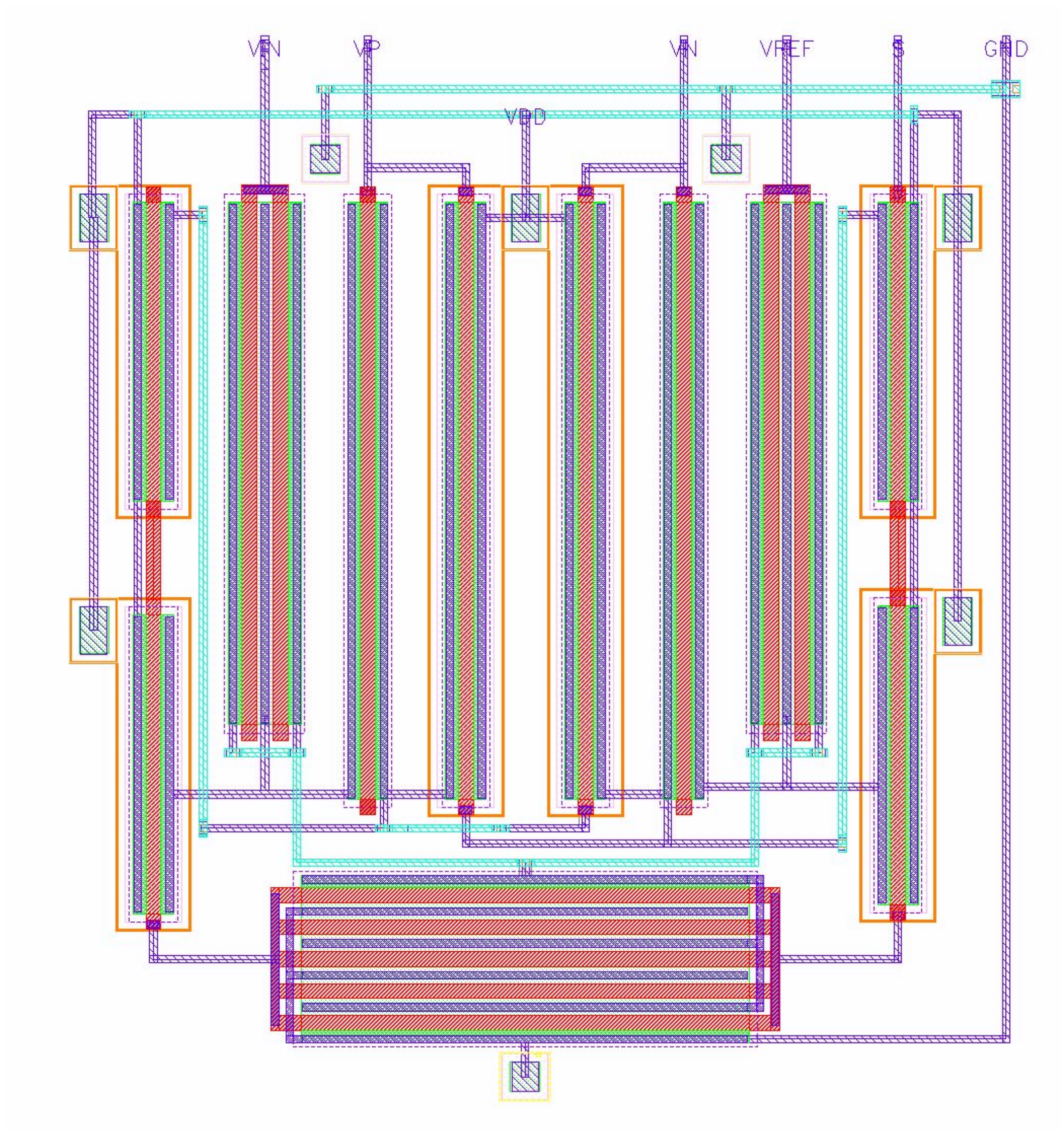


Figure 2.9: Layout of the comparator. 6.11um width, 5.77um height.

2.2 Input multiplexer

The input multiplexer decides whether the ADC should be fed a new sample or the residue voltage. Its output is controlled by a 3bit counter that is reset after counting to seven. When it resets, the multiplexer output is assigned to sample a new voltage of the signal, otherwise it is set to the residue voltage.

The counter being used is a 3 bit ripple counter. When the counter has reached seven counts, the three outputs ($Q_{1,2,3}$) goes high for one clock cycle. See figure 2.12 below. In this cycle the multiplexer changes input to V_2, or the input voltage. Otherwise the input is V1, or the residual voltage.

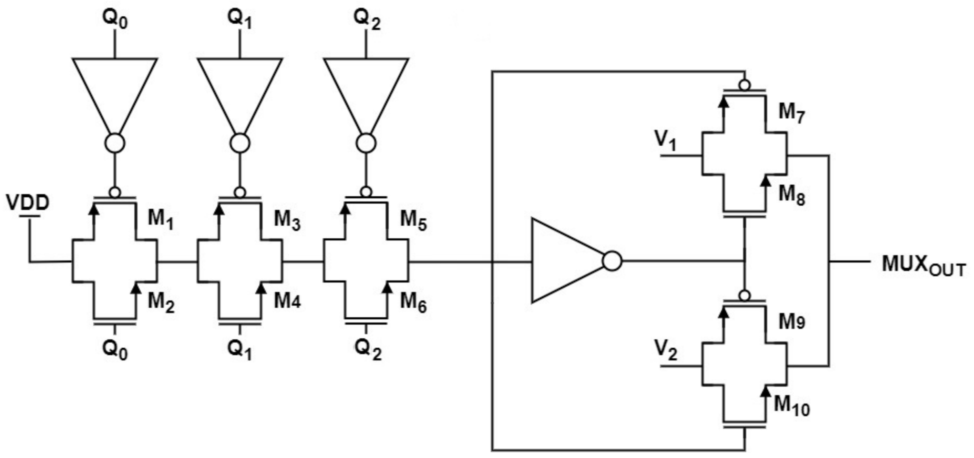


Figure 2.10: Input multiplexer schematic.

One mistake was discovered after the tape-out was delivered however. The input of the inverter is pulled up to V_{DD} when the counter is 111, but when the counter is not 111 the input is floating. A pull down on the input of the inverter was forgotten, which means that it is floating and is likely high all the time.

The layout of the input multiplexer is shown in figure 2.13 below. This cell is placed in the top layer.

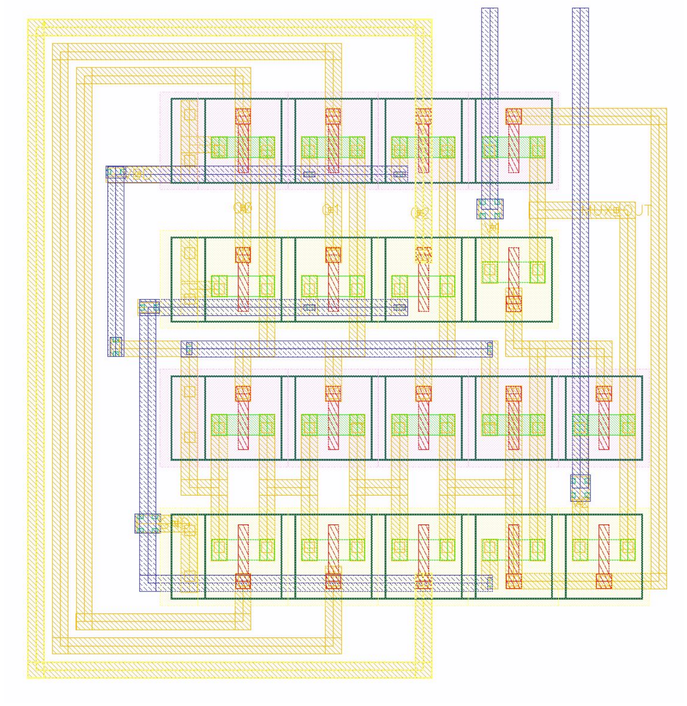


Figure 2.11: Input multiplexer layout. 6.247um width, 6.44um height.

2.3 Reference voltage multiplexer

The reference voltage multiplexer selects whether or not the sampled voltage should subtract or add V_{ref} based on the decision from the comparator. If the comparator outputs a '1', the multiplexer selects $-V_{ref}$ so the sampled voltage gets V_{ref} subtracted (see figure 1.4).

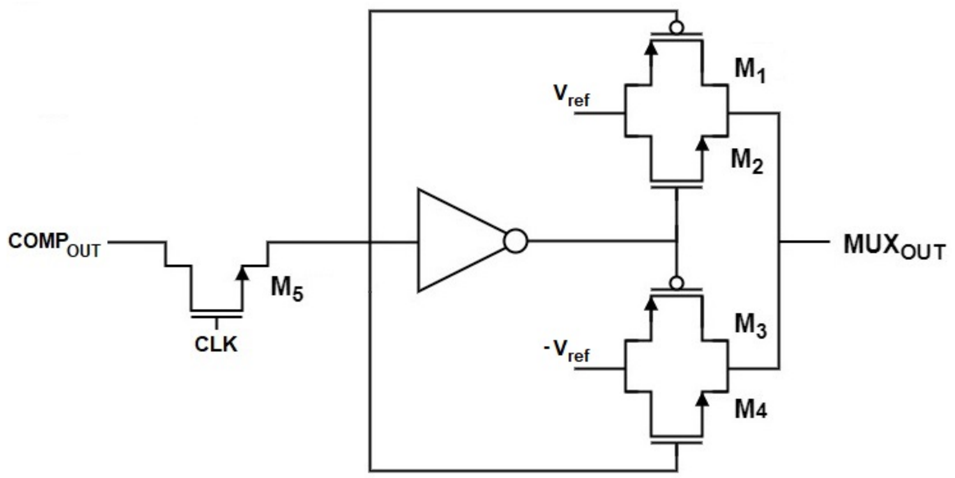


Figure 2.12: Reference voltage multiplexer schematic.

As with the input multiplexer, the reference voltage multiplexer is also placed in the top tier.

This multiplexer implementation is also lacking a pull resistor, which means that the input that is assigned to the output is uncertain. However, the $\mathrm{MUX_{OUT}}$ signal is not connected internally in the ADC core on the chip but needs to be processed externally (see figure 2.6, and can therefore be fixed off-chip.

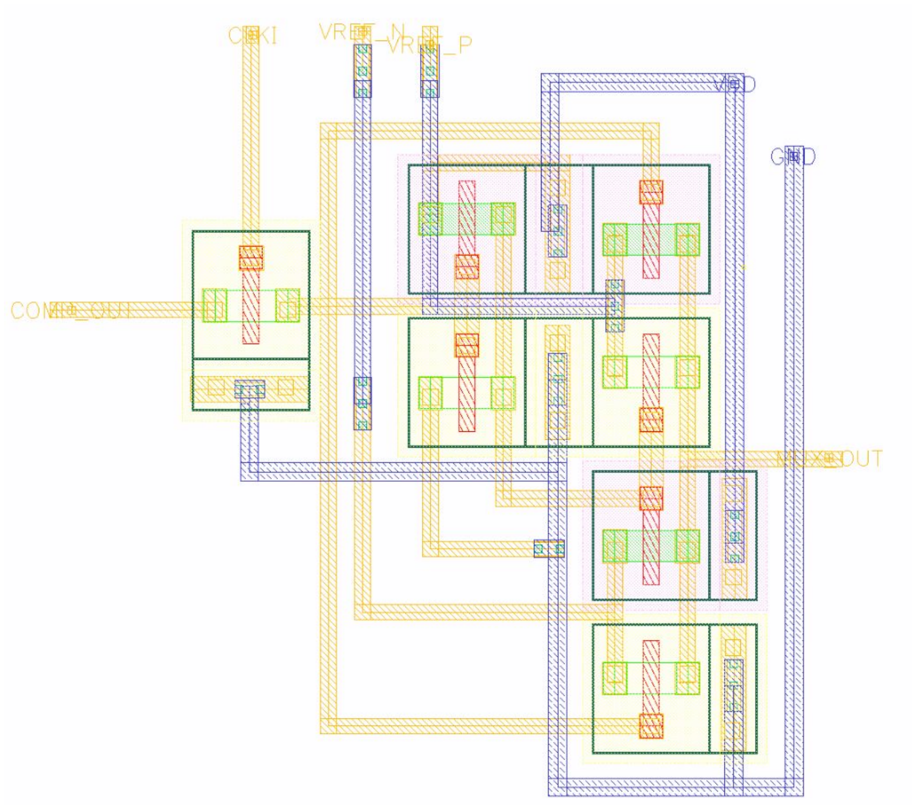


Figure 2.13: Reference voltage multiplexer layout. 4.824um width, 4.931um height.

2.4 3 bit counter

The counter is a 3 bit asynchronous ripple counter design that counts each change (rising edge) of an external, main clock [11]. It works by implementing a 'chain' of flip-flops where the least significant flip-flop (bit 0) is triggered by the external clock. All other flip-flops are clock by the output of the previous, less significant flip-flop.

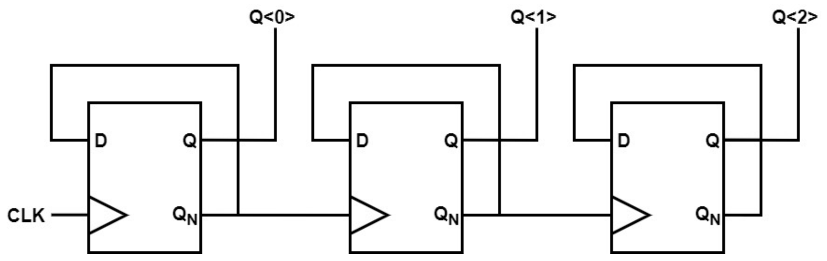


Figure 2.14: 3bit ripple counter schematic.

The timings and output of the 3bit counter is shown in figure 2.15. In the figure, Q0 is the least significant bit. output changes on the falling flank of the input clock.

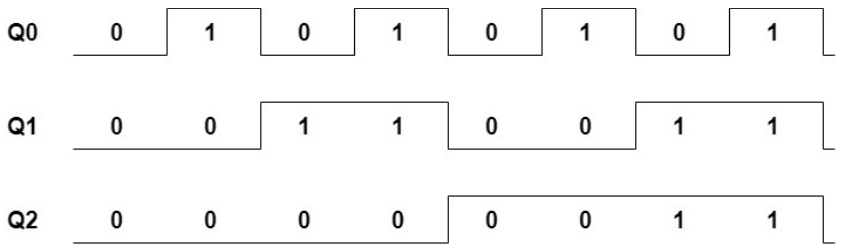


Figure 2.15: 3bit ripple counter timing diagram. After the 8th 'count' it resets to zero and continues indefinitely.

The layout of the counter is shown below in figure 2.16. It utilizes both the top and bottom tier to maximize area efficiency. The counter is surround by traces of metal layer 1 and 2, which are primarily used for V_{DD} and ground connections.

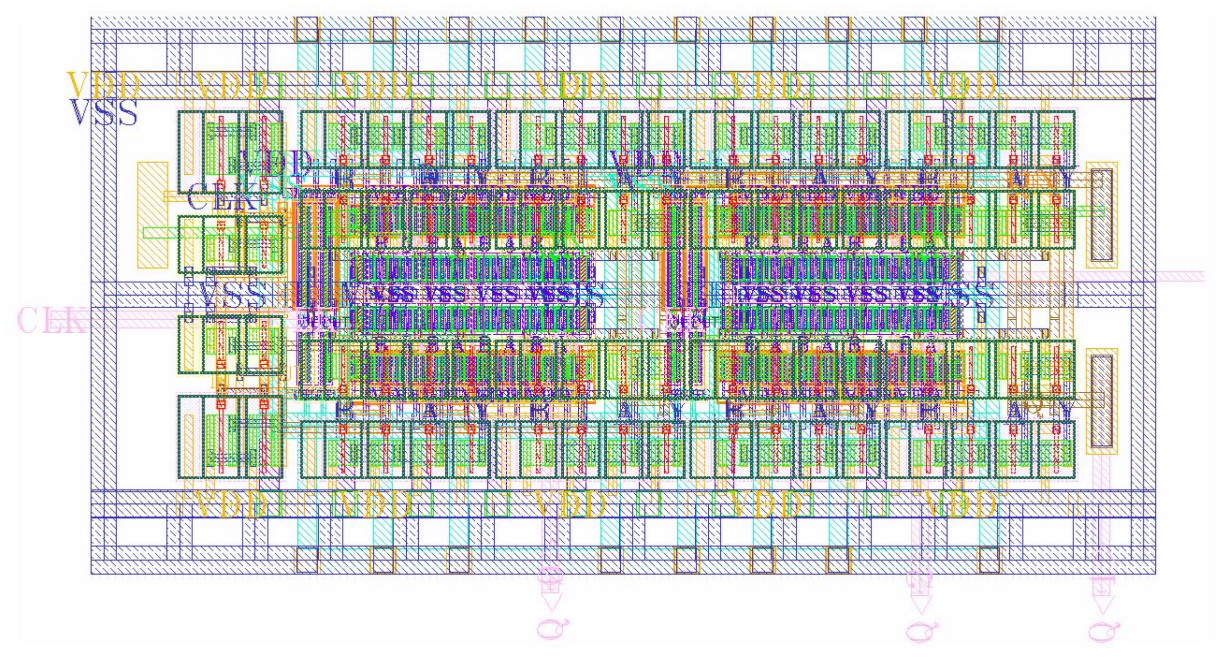


Figure 2.16: 3bit counter layout. 23.683um width, 12.515um height.

2.5 Layout

To finalize the layout, a padframe had to be assembled according to a predefined block size. The padframe includes a grid of metal 1 and 2, ensuring good connection to ground (metal 1) and V_{DD} (metal 2). The ADC core is connected to the ground grid. The total block size for this run is 1040µm x 1040µm, with a pad width of 60μm and 30μm pad spacing. Each side has 10 pads, and with $10 \cdot 4 = 40$ total pads and two pads on each side used to supply the pad frame. Thus it is 32 available pads.

As shown in figure 2.6 the core uses 10 pins of the padframe, excluding ground which is connected to the padframe itself.

The sub-circuits described previously are placed in the allocated space in the top left of the padframe. Necessary connections were made to the pads. The different pins and functions are shown in Table 3. Ground is connected directly to the ground layer in the padframe.

Table 3: Layout pinout

Pin	Function
1	VDD
2	CLK
3	V_{RES}
4	V_{IN}
5	V_{SAMPLE}
6	V_{REFN}
7	V_{REFP}
8	MUXOUT
9	V_{TH}
10	V_{OUT}

The corresponding pins to table 3 above and the layout of the tape-out is shown in figure 2.17. The figure shows a closer look at the ADC implementation and used pins. The ADC implementation is located in the top left corner of the chip, as previously mentioned.

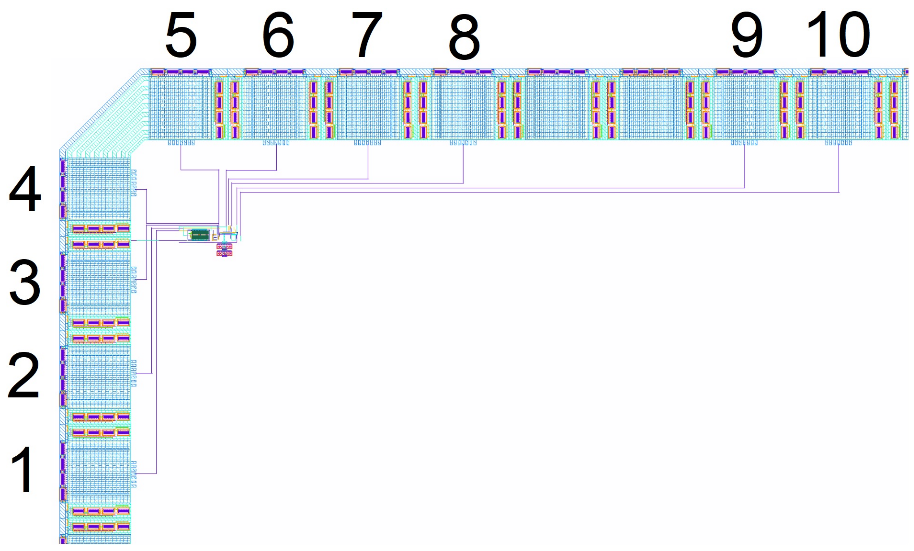


Figure 2.17: Layout of the cyclic ADC core made for tape-out.

Figure 2.18 shows an even closer look of the ADC, where it is possible to distinguish the different cells that have been implemented.

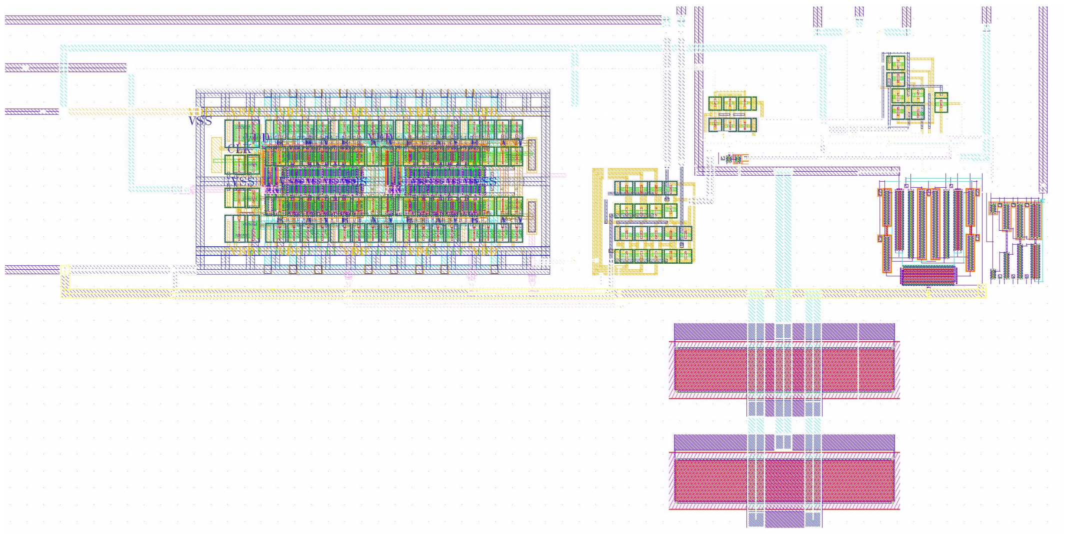


Figure 2.18: Zoomed view of ADC core layout. The core is approximately 59um wide and 32.1um tall.

2.6 Discussion

As previously mentioned, the tape-out boiled down the implemented cyclic ADC to simple design blocks due to the time restrictions early in the project. The process of designing in 3D rather than traditional 2D also introduced a new design mindset. With little previous knowledge and experience of integrated circuit design in 2D, the tape-out section had a steep learning curve in terms of layout techniques and is in retrospect lacking some finesse because of it. The cells are therefore not as consistent in the layout as they could have been.

The important findings for the tape-out section remains to be seen until the chip is received and tested. More work is required for it the ADC to be fully functional. The cells missing in figure 2.6 (amplifier and adder/subtractor) needs to be implemented off-chip to obtain the ADCs specifications.

3 Switched-Capacitor Cyclic ADC

The multiply by two amplifier is one of the most important components in the cyclic ADC, as the accuracy requirements. The amplifier can be realized by implementing a switched capacitor circuit, which acts as both the S/H and multiply by two circuit. Building an accurate gain of two is a difficult task, as capacitor mismatch leads to gain errors and non-linearites in the converter [9].

3.1 Ratio dependent SC gain

To achieve a certain gain, a capacitance ratio between two capacitors is often used. In a ratio dependent switch capacitor circuit, the input voltage is sampled on one capacitor before transferring the sampled charge to the other capacitor. If the first capacitor has double the capacitance of the second, the output voltage will be doubled and hence a gain of two is achieved.

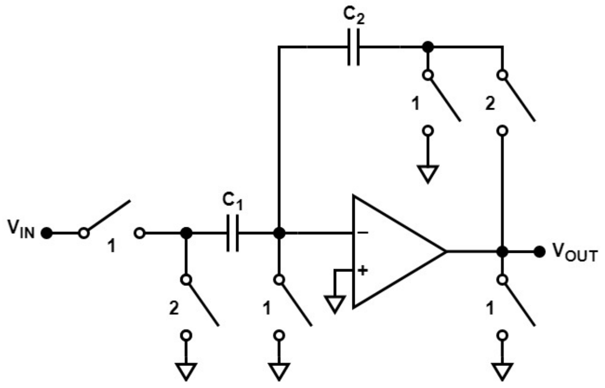


Figure 3.19: Simple switch-capacitor gain stage.

In figure 3.19 above, a simple switch-capacitor gain stage shown. This gain stage uses two phases to sample and amplify. In the first phase the input voltage is sampled onto C_1 and C_2 discharges. During phase two, the charge on C_1 is transferred to C_2. At the end of phase two, the output voltage is

$$V_{out} = -\frac{Q_1}{C_2} = -V_{in} \cdot \frac{C_1}{C_2} \tag{4}$$

From equation 4 it can be seen that the gain is equal to the ratio between capacitor C_1/C_2. As the gain is set by the ratio of capacitance between the two capacitors, the accuracy is limited to the capacitance mismatch.

3.2 Ratio independent SC gain

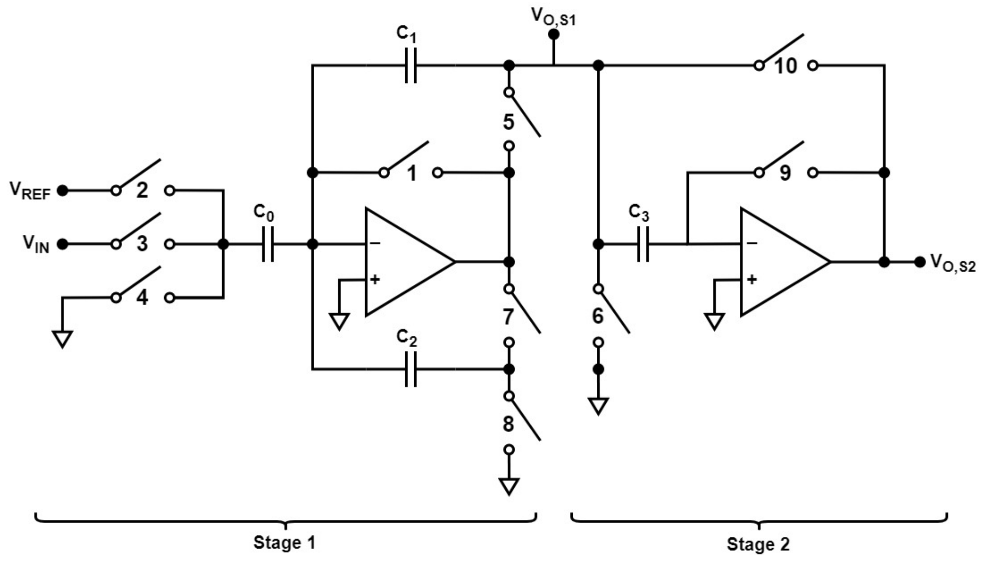


Figure 3.20: Circuit schematic of the ratio-independent switch-capacitor implementation.

A ratio independent gain cycle is proposed in [9]. This topology operates with two operational amplifiers, four capacitors and ten switches no matter the number of bits per sample converted. For a 1-bit conversion the circuit needs three clock cycles, therefore, an n-bit conversion requires 3n clock cycles.

This implementation is divided in two stages. Stage 1 is used for multiplication by two and addition/subtraction and stage 2 is used as a S/H circuit and comparator.

Each conversion uses a switching sequence with three phases each. In the first sequence the input voltage is sampled and the MSB is determined. Based on what the MSB is, the next sequence multiplies the input voltage, which was sampled in the MSB sequence, by two and either add or subtract the reference voltage from that multiplied voltage. Each conversion of a sample therefore starts with a MSB sequence before repeating a new sequence based on what the preceding bit was. [12]

The first phase of each sequence is named 'reset', the second phase is named 'computation' and the third is named 'decision'.

Phase 1 is the reset phase, where the outputs of both amplifiers are zeroed. In the MSB reset phase, V_{in} is ready to be transferred to the output of stage 1 amp. In the reset phase of the residual sequences (b(i)=0 and b(i)=1) V_{ref} or 0 are connected to C_0 in order to either add or subtract V_{ref} in the transfer phase 2. Additionally, the result from the previous phase is transferred from C_3 back onto to C_1 to be used to compute the next residual in phase 2.

In the next phase, computation, the input at C_0 is transferred as charge to the output of stage 1 amplifier, so that's V_{in} in the MSB computation phase 2, and $\pm V_{ref}$ in the residual computation phase 2. In the residual phase 2, the previous value is also added two times onto the output of stage 1 amp, once stored on C_1 and once transferred via C_2.

In the final phase, decision, the result of phase 2 is transferred via C_3 to the input of stage 2 amp. Stage 2 amp is in open loop configuration and evaluates the result and decides the bit value of this iteration. At the same time the result of phase 2 is sampled onto C_2 to be used later in the next computation phase.

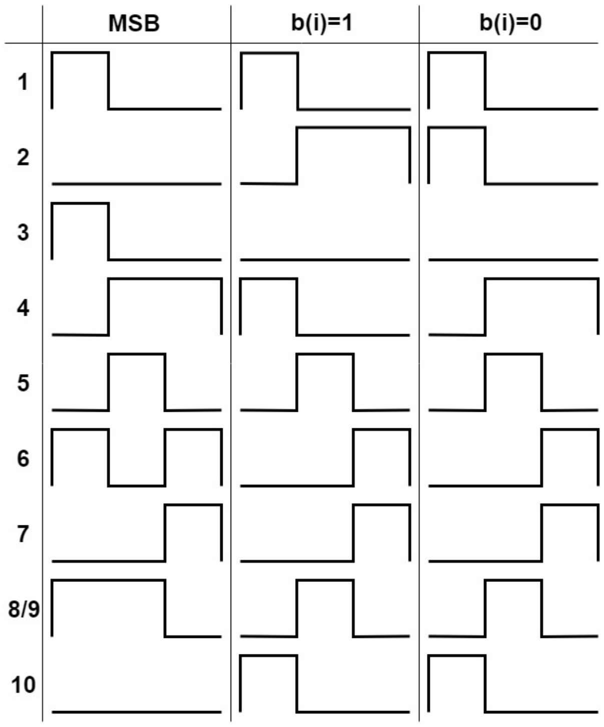


Figure 3.21: Timings for the different switches in the different sequences.

How the multiplication by two, addition/subtraction, S/H and comparison is explained by focusing on the charge transferred between the capacitors in each phase of each sequence. The timing diagram for the switches within the different sequences and phases are shown in figure 3.21 [12].

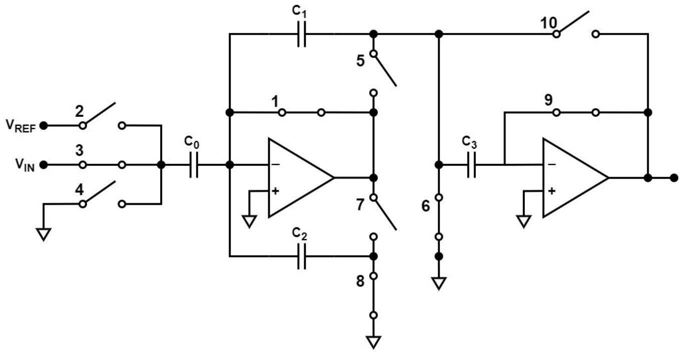


Figure 3.22: MSB phase 1, 'reset'.

In the first phase of the MSB sequence, called 'reset', the input (V$_{\text{in})}$ is sampled on C$_0$ as switch 3 closes (conducts) and opens again.

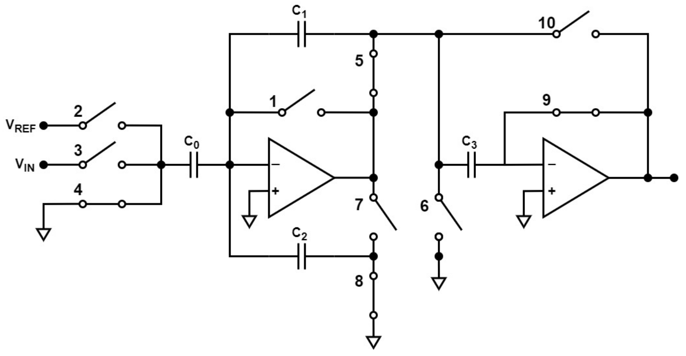


Figure 3.23: MSB phase 2.

In the second phase, computation, the charge on C$_0$ is transferred to C$_1$ and sampled onto C$_3$. The direct feedback, switch 1, opens and switch 5 closes so the feedback goes through C$_1$ and also connects C$_3$.

$$Q_{C_1} = V_{O,S1} \cdot C_1 \tag{5}$$

$$Q_{C_2} = 0 \tag{6}$$

$$Q_{C_3} = V_{O,S1} \cdot C_3 \tag{7}$$

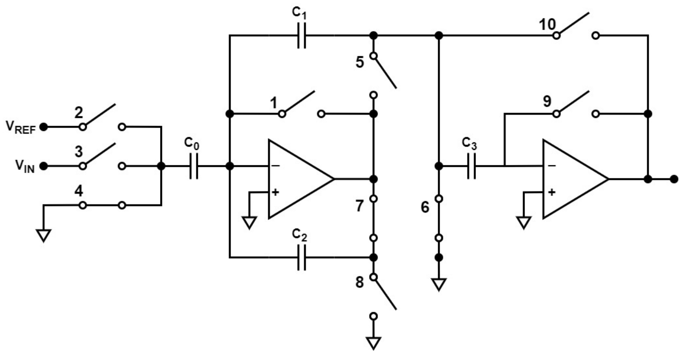


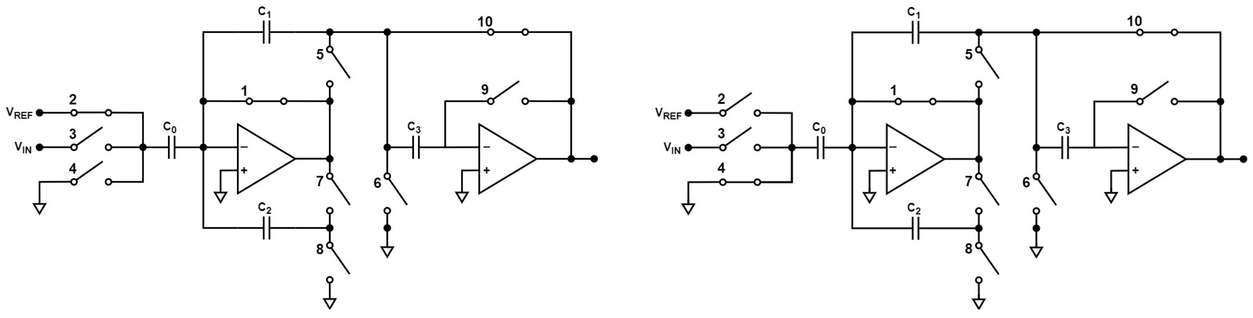
Figure 3.24: MSB phase 3.

In the third and final phase of the MSB sequence, the comparison operation is done in the second stage and the MSB is obtained. C_1 is disconnected (switch 5) and then C_2 connects (switch 7). The feedback loop on the second amplifier disconnects (switch 9) and C_3 connects to ground (switch 6) and the output of the amplifier gives the MSB.

$$Q_{C_1} = 0 \tag{8}$$

$$Q_{C_2} = -V_{O,S1} \cdot C_1 \tag{9}$$

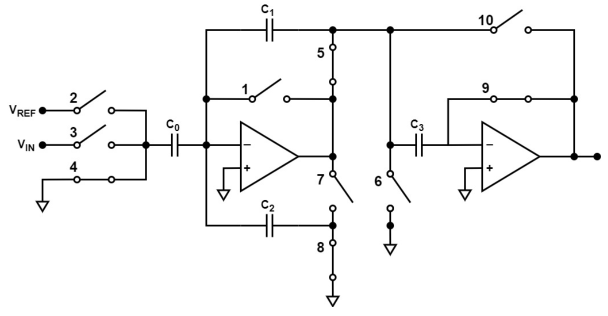
$$Q_{C_3} = V_{O,S1} \cdot C_3 \tag{10}$$



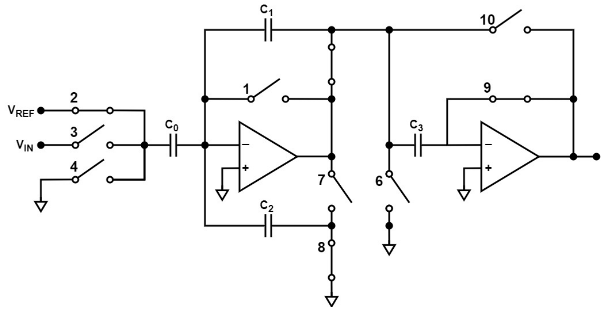
(a) b(i)=0. (b) b(i)=1.

Figure 3.25: Phase 1 of b(i+1).

Based on the MSB, the input is either connected to V_{REF} or GND in the first phase. The charge on C_3 in the previous phase is sampled onto C_1. The external feedback loop (switch 10) on the second amplifier reconnects to C_3, restoring the V_{in} there. The first amplifier is 'reset' with its feedback loop switch (switch 1) closed.



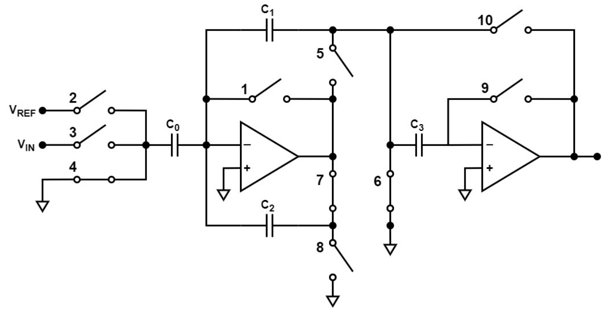
(a) b(i)=0.



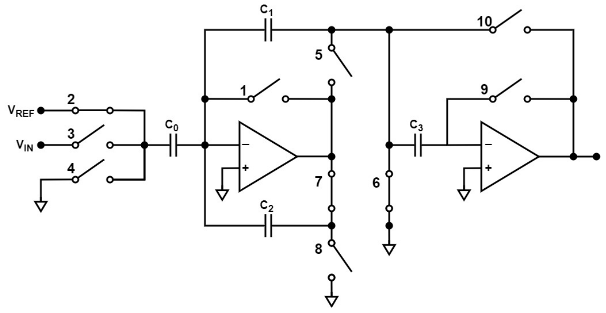
(b) b(i)=1.

Figure 3.26: Phase 2 of b(i+1).

The charge stored on C_2 is transferred to C_1 by first connecting the feedback through C_1 and then closing switch 8, to double the V_{in} voltage. V_{ref} is then either added (figure 3.26b) or subtracted (figure 3.26a) by switching the input from V_{ref} to ground or vice versa.



(a) b(i)=0.



(b) b(i)=1.

Figure 3.27: Phase 3 of b(i+1).

A bit is now obtained in the final phase of the sequence. The next decision is obtained by switching the second amplifier to open loop configuration to get the bit. The present input is sampled onto C2 by first opening the feedback loop through C_1 (switch 5) and then closing the feedback through C_2.

Comparing the ratio-dependent and the independent, it's obvious that the cost of using a ratio-independent circuit is higher than a ratio-dependent one, as it uses two extra capacitors, one extra operational amplifier, more switches and digital circuitry which all contribute to more area and power consumption. However, there is no error contribution from capacitor ratio mismatch, and it is insensitive to offset voltage in the operational amplifiers [ref]. With that being said, the ratio-independent circuit replaces capacitors that are larger in size in the ratio-dependent implementation. So one would likely only have the area of one unit capacitor more in terms of capacitors.

Conventional Cyclic/Algorithmic ADCs are often subject to conversion errors because of inaccuracies in the residue voltages. This residue error might occur if the comparator offset is too high and/or because of loop-gain non-idealities. Because the residue error gets amplified, the error will increase for each cycle. This will increase the conversion error. Overall accuracy is then usually limited by the last conversion cycle.

4 Switch-capacitor 3D implementation

The design proposed in 3.2 requires 9 different clocks to control the switches, as shown in figure 3.21. Each conversion takes 3 phases, but depending on if its the MSB-conversion or a conversion of a residual, the design also requires three different sequences. This causes the clock control circuitry to be a lot more complex than for an ratio-dependent circuit. The clock control circuitry also needs to be controlled by a counter, that resets the conversion.

4.1 ADC core

As the tape-out, the improved implementation of the cyclic ADC will have analog cells in the bottom tier, and digital in top. The switches mentioned in chapter 3 are transmission gates with low on-resistance. The operational amplifiers does not take up a lot of space in the bottom tier, but the capacitors are quite large. Therefore, the transmission gates and all the digital clock circuitry will go in the top tier to maximize area efficiency. By placing the transmission gates in the top tier, the interconnects in the core are also shortened.

As mentioned in chapter 3, one bit-conversion takes 3 phases to complete. This means that the sampling rate is equal to the clock frequency divided by the number of phases required to complete one conversion. To increase the sampling rate, a more efficient clock circuit that completes two phases per clock cycle, as compared to 1 phase per clock cycle in this circuit, would double the sampling rate.

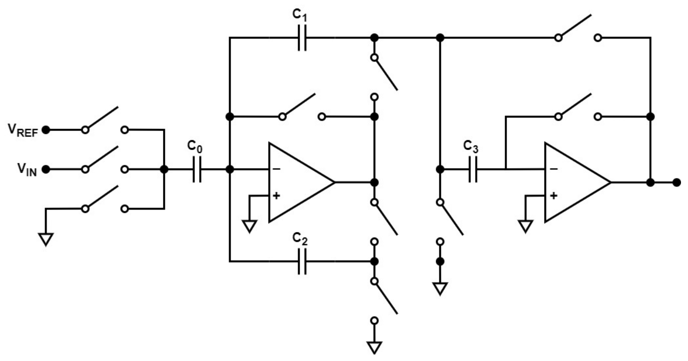


Figure 4.28: Circuit schematic of the ADC core.

As the mentioned in chapter 3, the overall accuracy of the ADC is limited by the last conversion cycle. Therefore the specifications for the ADC were defined after multiple simulations of the gain offset as showed in figure 4.29. After each simulation the parameters in the operational amplifier, capacitor sizes and clock speed were changed and tuned to find the best resolution and highest sampling rate. Through a substantial amount of back and forth between simulations and parameter-tweaking the best results were a 6 bit conversion at a 55 KHz sampling rate.

The specifications for the ADC is shown in table 4 below.

Table 4: Specifications for the ADC

Parameter	Value
$C_{0,1,2,3}$	$7.78\ pF$
Clock speed	$1\ MHz$
Sampling rate	$55\ kHz$
Resolution	6 bit

Figure 4.29 shows the output voltage of the first and second amplifier in the ADC. The simulation runs a 5-bit conversion with the reference voltage set to zero to estimate the gain of each cycle. The input is set to 20 mV. The markers show the voltage at the end of each computation phase (3.2). The upper plot shows the output of stage one with markers showing the voltage after the second phase, in which the sampled voltage should be doubled. At the end of each computation phase, the voltage should be doubled from the previous computation phase. However, the results show that the gain is not exactly 2. This might be caused by inaccurate timing of the clock flanks within the different phases.

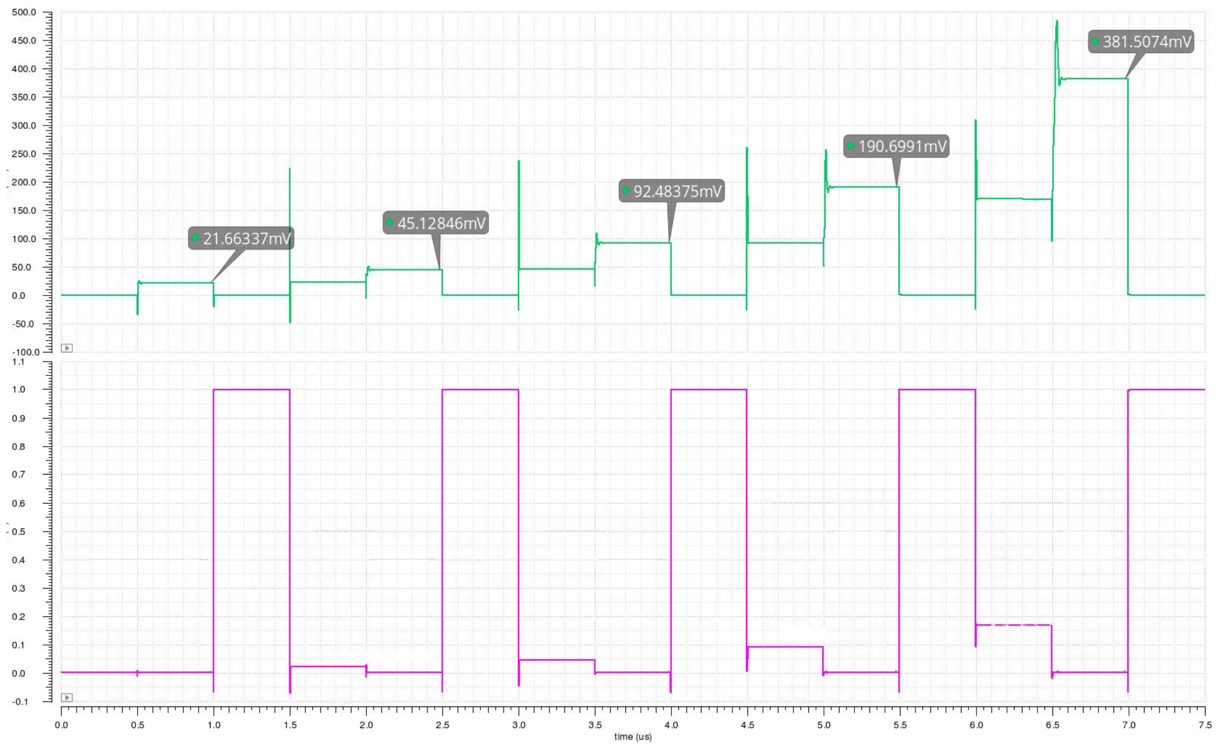


Figure 4.29: Simulation of the ADC with V_{REF} set to zero and input set to 20mV. The upper plot shows the output of stage 1 and the lower plot shows the output of stage 2.

4.2 Operational amplifier

The amplifier in figure 4.30 consists of two gain stages and current bias circuitry. The first stage is a pMOS differential pair with nMOS current mirrors [13]. The differential pair is biased by M$_5$, which is one of the two output transistors of the current mirror formed by M$_8$, M$_5$ and M$_7$. The current is fed by a reference current which is generated by the current mirror formed by M$_{10}$, M$_{11}$, M$_{12}$ and M$_{13}$. Second stage is a common-source amplifier formed by the common source transistor M$_6$ and its current load transistor M$_7$. The second stage also takes part in frequency compensation in the operational amplifier [14].

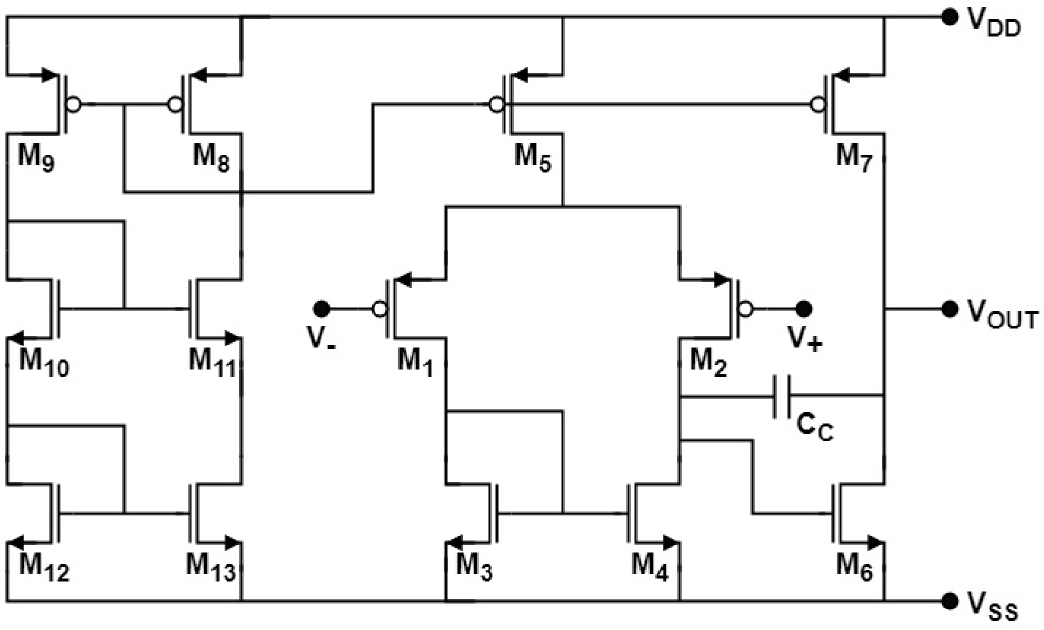


Figure 4.30: Two stage amplifier

The maximum absolute error of the 2x gain due to limitations in the DC gain is given by the following equation [9]:

$$E_{gain,max} = \frac{3}{A_V}(2^n - n - 1) + \frac{3}{A_V}(2^n - \frac{2}{3})\frac{|V_{OFF}|}{V_{REF}}\ [LSB] \tag{11}$$

To keep maximum absolute error lower than 0.5 LSB when V$_{\mathrm{OFF}}$=0, A$_{\mathrm{V}}$ needs to be bigger than 50 dB for a 6 bit conversion and bigger than 57.15dB for a 7 bit conversion. Therefore, the gain of the amplifier is the limiting factor for the ADCs resolution.

The widths and lengths of the transistors are show in the table below. These widths and lengths were first acquired through hand calculations to give an initial design, or starting point, and to easier see what parameters affected the gain. After acquiring a starting point, the final specification was established through simulation and tuning between each simulation until a satisfying result was achieved.

Table 5: Transistor specifications. All lengths are 100nm.

Transistor	**Width/size**
$M_{1,2}$	$20\mu m$
$M_{3,4}$	$5\mu m$
$M_{5,7,8}$	$35\mu m$
M_6	$10\mu m$
$M_{9,10,11,12}$	$45\mu m$
M_{13}	$55\mu m$
C_C	$242fF$

Figure 4.31 shows the gain and phase margin of the operational amplifier. Its 3dB bandwidth is approximately 1MHz, with a gain of 62 dB.

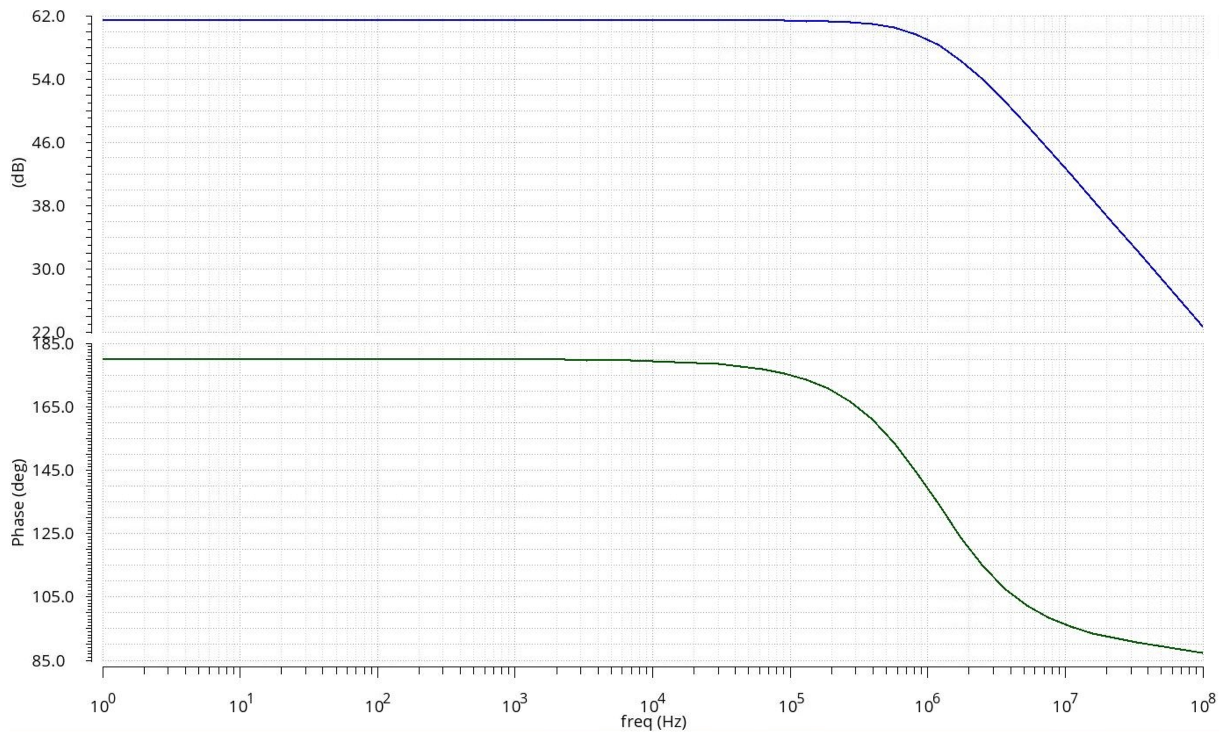


Figure 4.31: Gain and phase of the operational amplifier.

Figure 4.32 shows the layout of the operational amplifier in the bottom layer. The amplifier is 'surrounded' by a trace of metal layer 1 and metal layer 2. Metal layer 1 is ground and metal layer 2 is V_{DD}. This was implemented for easier V_{DD} and ground connection for the transistors in this cell, but also to simplify the connection to other cells in the ADC. This way the different cells can easily be connected to V_{DD} and ground with a thick trace.

The width of the operational amplifier is 28.375μm and the height is 10.842μm.

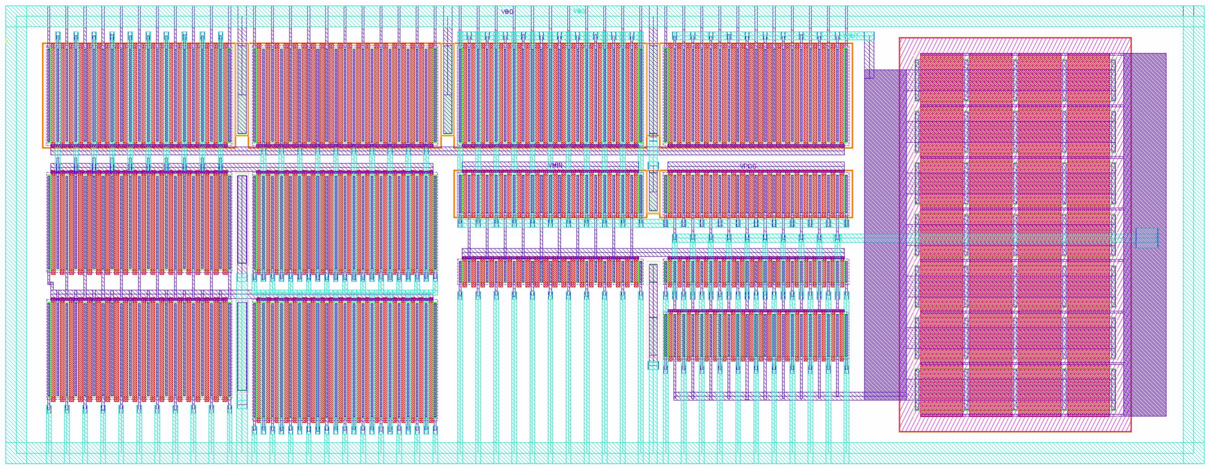


Figure 4.32: Layout of the operational amplifier.

4.3 Transmission gate

The switches described in chapter 4 are transmission gates placed in the top tier. The gates are designed to have as low drain-source resistance as possible to reduce loss. The transistors length are 100nm and the total width are 500nm. The transistors are split into multiple fingers such that the length stays the same and the width are W/k. where k is the total number of fingers. This way the drain-source resistance are in parallel, and therefore the total resistance is reduced.

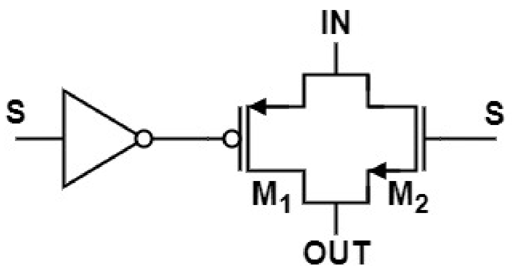


Figure 4.33: Schematic of the transmission gate.

The layout use the same design principle as the operational amplifiers with surrounding the cell with V_{DD} and ground metal layers. V_{DD} is place in metal 5 and metal 6 is ground.

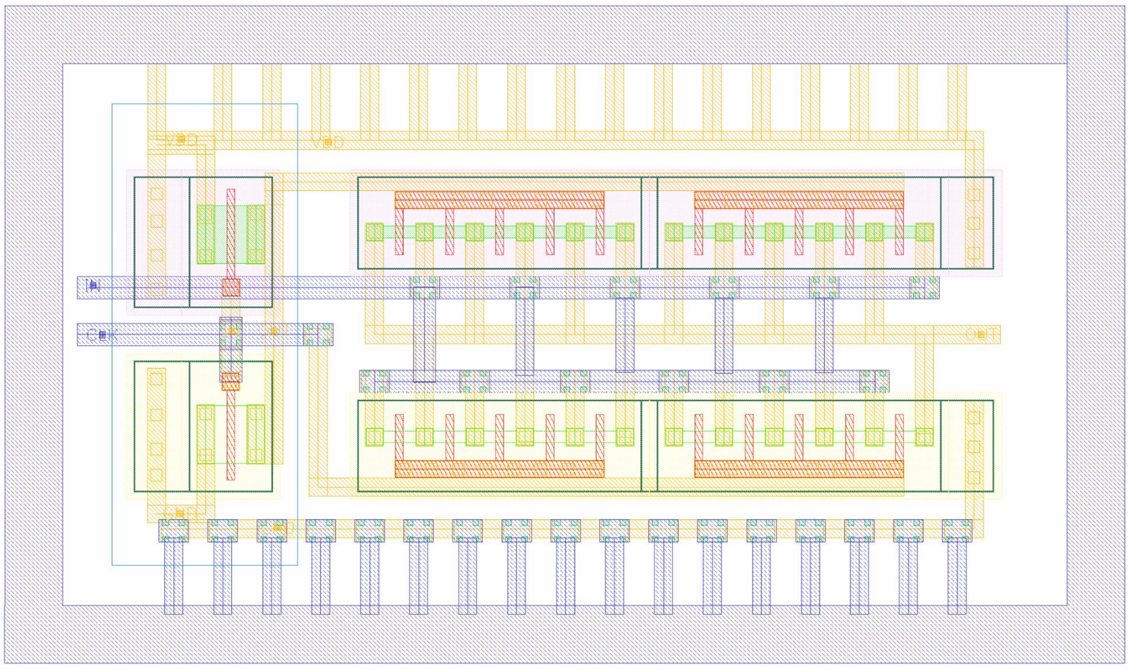


Figure 4.34: Layout of the transmission gate.

4.4 Digital logic

To implement the exact clock circuitry [12] for the switches in the switch capacitor implementation, it was thought that a state machine would be the best solution to realize this clock circuitry.

Vivado by Xilinx was used for compiling and synthesising VHDL code. Vivado is a software suite made by Xilinx for synthesis and implementation for Xilinx FPGAs [15]. Vivado has a synthesis tool that creates a circuit with latches and look-up tables based on the written VHDL code.

Based on the timings in figure 3.21 the state machine was designed. The VHDL process reacts to the rising edge of the main clock to change the timings for all the switches, which means that each phase uses one full clock cycle. Therefore, one bit conversion takes 3 clock cycles and one full 6bit conversion 18 clock cycles. After the 6th bit has been converted, the state machine resets and begins another conversion. This process runs continuously.

The VHDL code was then run through Vivados synthesis tool and the circuit was extracted A.1. After extraction the different latches and look-up tables were converted to a boolean expression via a Karnaugh map [16] that could be implemented with logic gates in Cadence. All the latches was manually placed and routed afterwards.

As previously mentioned, the clock circuitry is an exact implementation of the timing diagram published in [12]. This however is not quite sufficient as it does not implement the switches' necessary timing and order of flanks within each phase.

The layout of the digital logic is shown in figure 4.35.

4.5 Layout

This section covers the layout of the switch-capacitor circuit described in chapter 3.2. The layout was not taped out, but was done to gain experience in 3D layout and gauge advantages and disadvantages of utilizing multiple tiers and find out where the challenges are. Implementing the improved ADC in layout was seen as valuable information despite it not being sent to tape-out.

4.5.1 Top layer

The top layer consists of the transmission gates and the digital clock circuitry. It is 93.98 µm times 60.75 µm big. An effort to place the cells with connections to each other as close together as possible was made to shorten the necessary traces. Every cell uses the same layout principle of surround the cell with V_{DD} and ground metal layers. The height of the cells were determined beforehand so that every cell could easily be placed together and close to the cell which they need to connect to.

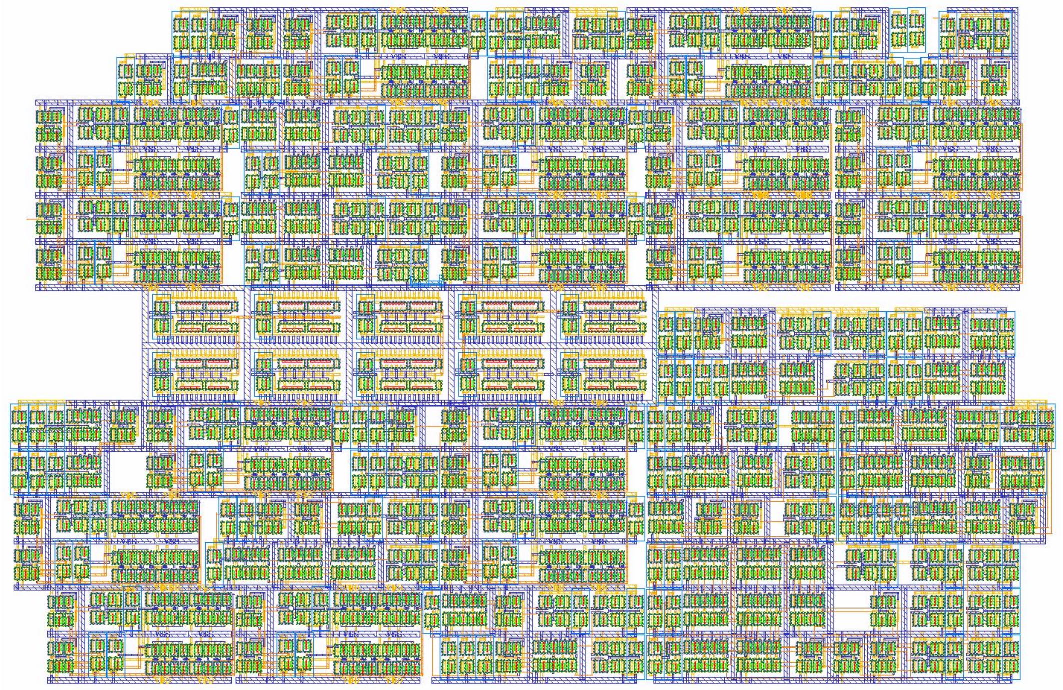


Figure 4.35: Layout of the top layer. 93.98um x 60.75um

4.5.2 Bottom layer

The bottom layer consists of the two operational amplifiers and the capacitors. It is 65.054µm times 65.406µm. The four capacitors is $C_{0,1,2,3}$ from figure 3.20. MIMCAP (Metal-Insulator-Metal Capacitor) were used. This placement of the operational amplifiers and capacitors was deemed the most space efficient solution with the shortest traces.

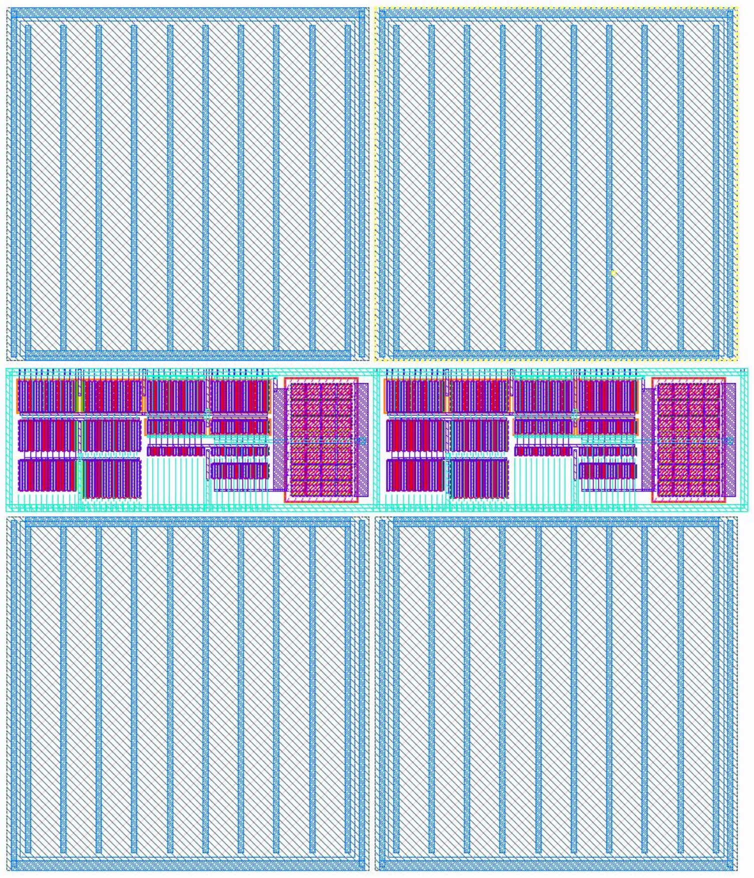


Figure 4.36: Layout of the bottom layer. 65.054um x 65.406um

4.5.3 Top and bottom combined

Figure 4.37 shows the placement of the bottom and top tier in regards to each other. The tiers were placed in this configuration to minimize the necessary trace length between the top and bottom tiers. The transmission gates in the top layer are placed directly over the operational amplifiers.

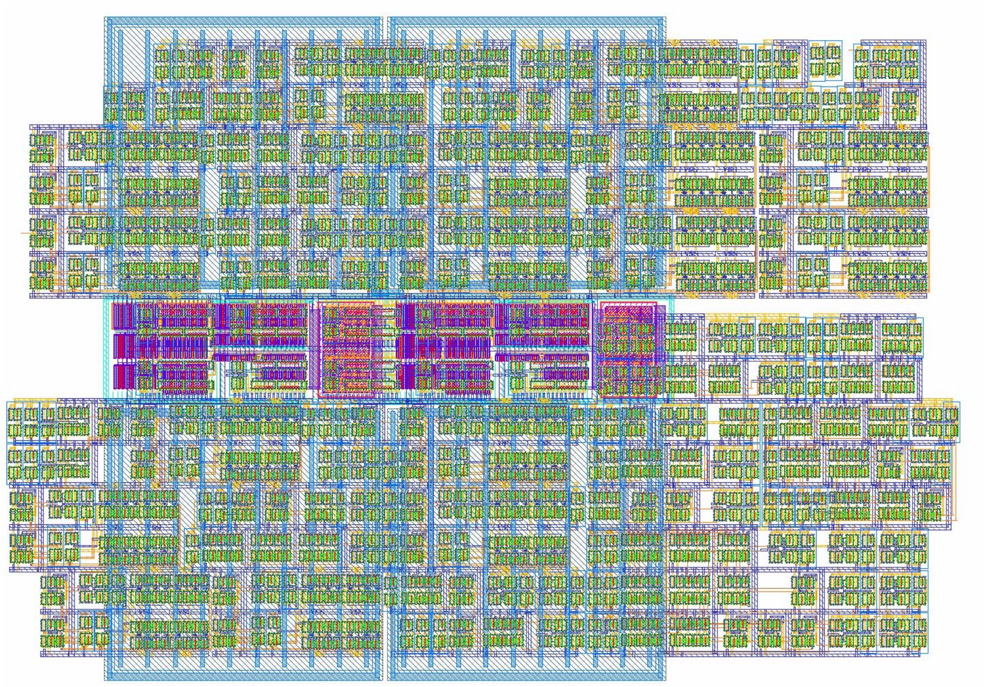


Figure 4.37: Complete layout with top and bottom layer combined. 65.406um x 93.98um

Figure 4.38 shows the top and bottom tier next to each other to illustrate the size difference. In this figure it also easier to visualize how much longer the interconnects would need to be if this were to be implemented in a 2D design.

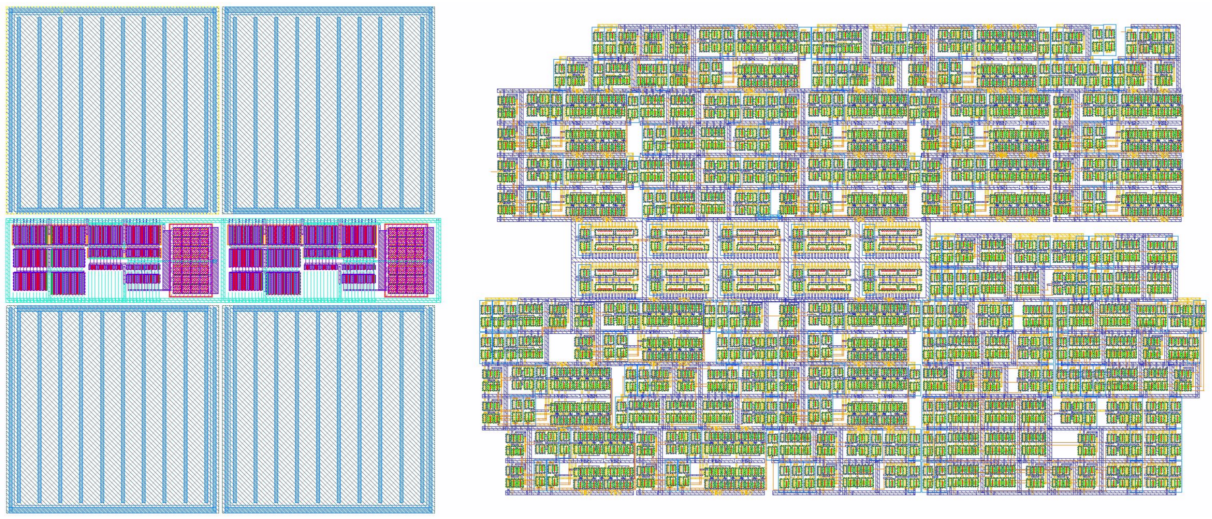


Figure 4.38: A side by side comparison of the layers.

4.6 Discussion

As the bottom and top tier utilizes different technologies, 3D vs 2D implementation are difficult to compare. If the tiers were to use the same technology and size, one could analyze and compare area efficiency, interconnect lengths, power consumption, time delay and parasitic capacitance more efficiently. In order to maximize space efficiency with these technologies, the digital should be placed in the bottom. But as mentioned before, the placement decision was made due to a higher reliability rate in the bottom tier in regards to the tape-out and further testing on the chip.

Although difficult to analyze, one could do a bit of educated speculation. Implementing all the cells in the top tier would have been bigger than the combined areas of the bottom and top tier. In comparison, implementing all the cells in the bottom tier might be smaller than the combined areas, but only if the scaling of the digital circuits would make it smaller than the the top area minus the bottom area. In other words, if the digital circuits would shrink down to a factor of $(94 - 65)/94 = 30\%$. Since the technology in the top tier is 65nm and the bottom tier is 28nm, one could approximately expect a digital shrinkage down to $28^2/65^2 = 19\%$. This would then certainly occupy less area in 2D in the bottom tier technology. Nonetheless, if one were to ignore the low yield in the top tier it would have made more sense to place the analog in the 65nm top tier and digital in the 28nm bottom tier in terms of area optimization, as analog circuits does not scale the same way digital circuits do. However, in this project, the fear of low yield and inaccurate simulation models restricted the tier division. Had it not been for these restrictions, the analog would have been placed in the top tier and digital in the bottom tier.

If the designs were placed side by side in one layer, the 3D interconnection between the layers would have to be replaced by traces. As the transmission gates are placed in the top tier in the 3D implementation, the analog signals would have to travel over a substantial amount of digital circuitry in a 2D implementation. This could potentially add a lot of coupling noise. One might go for a more mixed placement of digital and analog in 2D, but that would also increase the risk of coupling between analog and digital. Based on these concerns, one can assume that the interconnection length would be bigger in 2D than in 3D and additionally the noise might be more prominent. However, the noise could also be an issue in 3D when placing analog right below digital circuitry. Nonetheless, there are some possibilities of using inter-tier metal layers for shielding and the 3D sequential integration process features another polysilicon layer that can be placed between the tiers. The shield properties and effectiveness remains to be verified until the chip is finished and tested.

Another issue to address is the heat dissipation of the different tiers. With multiple transistors stacked vertically and consequently, thermal resistance from the heat source to heat sink increases [17]. Ideally, the largest heat sources should be placed in the top tier for lowest thermal resistance. If, however, the largest heat sources is to be placed in the bottom tier, thermal pillars or other solutions should be considered to dissipate the heat to the top[18].

4.7 Further design

To furthermore attain advantages, or disadvantages, in 3D sequential integration of cyclic ADCs, it would be beneficial to quantify how much 'saved' interconnect length a 3D implementation would provide. This would require a more comparable implementation in both 2D and 3D, meaning the comparison would be close to 1:1. With a comparable 2D and 3D implementation it would be easier to find which solution that have the least parasitic coupling and best noise protection.

Further design suggestions would be to maximize space efficiency further by placing digital in the bottom tier and analog in the top tier. Furthermore, mixing the digital and analog between the tiers could possibly be more efficient.

As this was a single-ended implementation, the accuracy of the ADC is limited. A differential implementation would be the preferred implementation in regards of accuracy. It would also be beneficial to implement an redundant signed bit algorithm to increase accuracy [19].

References

[1] Chang J.P. Chen F.F. Lecture Notes on Principles of Plasma Processing. Springer, 2003. ISBN: 978-0-306-47497-2.

[2] 3D-MUSE. URL: `https://www.3dmuse.eu/`.

[3] P. Brunet L. Batude and M. Vinet. "First demonstration of a CMOS over CMOS 3D VLSI CoolCube integration on 300mm wafers". In: IEEE Symposium on VLSI Technology (2016). DOI: `10.1109/VLSIT.2016.7573428`.

[4] F. Andrieu P. Batude L. Brunet and Et Al. "3D Sequential Integration: Application-driven technological achievements and guidelines". In: IEEE Solid-State Circuits Magazine (2018). DOI: `10.1109/IEDM.2017.8268316`.

[5] Jessy Micout. "Fabrication et caractérisation de transistors réalisés à basse température pour l'intégration 3D séquentielle". Communauté Université Grenoble Alpes, 2016.

[6] Behzad Razavi. Principles of Data Conversion System Design. Wiley, 1995. ISBN: 9780470545638.

[7] Zhiheng Wei. "A Study on Column-Parallel ADCs Using DMOS Capacitors for CMOS Image Sensors". Shizuoka University, 2014.

[8] Ajith kumar Puppala. "Design of a Low Power Cyclic/Algorithmic Analog-to-Digital Converter in a 130nm CMOS Process". Linköping Institute of Technology, 2012.

[9] Andreas Dreyfert. "Design and measurement of a cyclic ADC in 65 nm CMOS". Lund University, 2015.

[10] Behzad Razavi. "The StrongArm Latch". In: IEEE Solid-State Circuits Magazine 7 (2 2015), pp. 12–17. DOI: `10.1109/MSSC.2015.2418155`.

[11] Pradeep Singla et al. An Optimized Design of Reversible Sequential Digital Circuits. 2013. URL: `https://arxiv.org/abs/1306.2556`.

[12] Keikichi Tamaru Hidetoshi Onodera Testuo Tateishi. "A Cyclic A/D Converter That Does Not Require Ratio- Matched Components". In: IEEE Solid-State Circuits Magazine 23 (1 1988), pp. 152–158. DOI: `10.1109/4.272`.

[13] K. C. Smith A. S. Sedra. Microelectronic Circuits. Oxford University Press, 2016. ISBN: 9780199339143.

[14] Ching-Yuan Yang. Basic OpAmp Design and Compensation Chapter 6. URL: `http://cc.ee.nchu.edu.tw/~aiclab/teaching/Electronics3/lect09.pdf`.

[15] Xilinx Inc, Form 8-K, Current Report, Filing Date Apr 25, 2012. 2012. URL: `http://edgar.secdatabase.com/846/119312512182086/filing-main.htm`.

[16] Maurice Karnaugh. "The Map Method For Synthesis of Combinational Logic Circuits". In: Transactions of the American Institute of Electrical Engineers, Part I: Communication and Elec 72 (5 1953), pp. 593–599. DOI: `10.1109/TCE.1953.6371932`.

[17] Satya Vendra and Malgorzata Chrzanowska-Jeske. Thermal Management in 3D IC Designs for Nano- 2018. DOI: `10.1109/NMDC.2018.8605929`.

[18] Md Arif Iqbal et al. Thermal Management in Fine-Grained 3-D Integrated Circuits. 2018.

[19] R.L. Geiger B. Soufi S.Q. Malik. “A capacitor sharing technique for RSD cyclic ADC”. In: IEEE Solid-State Circuits Magazine (2005), 859–862 Vol. 1. DOI: `10.1109/MWSCAS.2005.1594237`.

A Appendix

A.1 VHDL code for clock circuitry

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;
------------------------------------------------------------
------------------------------------------------------------
ENTITY main IS
        PORT (
                S1, S2, S3, S4, S5, S6, S7, S89, S10 : OUT std_logic;
                CLK, B1B0                            : IN std_logic
        );
END main;
------------------------------------------------------------
------------------------------------------------------------
ARCHITECTURE Behavioral OF main IS
        SIGNAL phase_count : unsigned(1 DOWNTO 0) := "00";
        SIGNAL check_bit   : std_logic := '0';
        TYPE State_type IS (MSB, B1_1, B0_1, B1_2, B0_2, B1_3, B0_3, B1_4, B0_4, B1_5, B0_5);
        SIGNAL State : State_Type := MSB;
        ------------------------------------------------------------
BEGIN
        ------------------------------------------------------------
        ------------------------------------------------------------
        PROCESS (CLK)
        ------------------------------------------------------------
        BEGIN
                IF (rising_edge(CLK)) THEN
                        IF (check_bit = '1' AND B1B0 = '1') THEN
                                check_bit <= '0';
                                IF (State = MSB) THEN
                                        state <= B1_1;
                                ELSIF (State = B1_1) OR (State = B0_1) THEN
                                        state <= B1_2;
                                ELSIF (State = B1_2) OR (State = B0_2) THEN
                                        state <= B1_3;
                                ELSIF (State = B1_3) OR (State = B0_3) THEN
                                        state <= B1_4;
                                ELSIF (State = B1_4) OR (State = B0_4) THEN
                                        state <= B1_5;
                                END IF;
                        ELSIF (check_bit = '1' AND B1B0 = '0') THEN
                                check_bit <= '0';
                                IF (State = MSB) THEN
                                        state <= B0_1;
                                ELSIF (State = B1_1) OR (State = B0_1) THEN
                                        state <= B0_2;
                                ELSIF (State = B1_2) OR (State = B0_2) THEN
                                        state <= B0_3;
                                ELSIF (State = B1_3) OR (State = B0_3) THEN
                                        state <= B0_4;
                                ELSIF (State = B1_4) OR (State = B0_4) THEN
                                        state <= B0_5;
                                END IF;
                        END IF;
                        CASE State IS
                                ------------------------------------------------------------
                                WHEN MSB =>
                                        IF (phase_count = "00") THEN
                                                S1          <= '1';
                                                S2          <= '0';
                                                S3          <= '1';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        check_bit   <= '1';
                END IF;
                ------------------------------------------------------------
        WHEN B1_1 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                phase_count <= "00";
                check_bit   <= '1';
        END IF;
        -----------------------------------------------------------
WHEN B0_1 =>
        IF (phase_count = "00") THEN
                S1          <= '1';
                S2          <= '1';
                S3          <= '0';
                S4          <= '0';
                S5          <= '0';
                S6          <= '0';
                S7          <= '0';
                S89         <= '0';
                S10         <= '1';
                phase_count <= "01";
        ELSIF (phase_count = "01") THEN
                S1          <= '0';
                S2          <= '0';
                S3          <= '0';
                S4          <= '1';
                S5          <= '1';
                S6          <= '0';
                S7          <= '0';
                S89         <= '1';
                S10         <= '0';
                phase_count <= "10";
        ELSIF (phase_count = "10") THEN
                S1          <= '0';
                S2          <= '0';
                S3          <= '0';
                S4          <= '1';
                S5          <= '0';
                S6          <= '1';
                S7          <= '1';
                S89         <= '0';
                S10         <= '0';
                phase_count <= "00";
                check_bit   <= '1';
        END IF;
        -----------------------------------------------------------
WHEN B1_2 =>
        IF (phase_count = "00") THEN
                S1          <= '1';
                S2          <= '0';
                S3          <= '0';
                S4          <= '1';
                S5          <= '0';
                S6          <= '0';
                S7          <= '0';
                S89         <= '0';
                S10         <= '1';
                phase_count <= "01";
        ELSIF (phase_count = "01") THEN
                S1          <= '0';
                S2          <= '1';
                S3          <= '0';
                S4          <= '0';
                S5          <= '1';
                S6          <= '0';
                S7          <= '0';
                S89         <= '1';
                S10         <= '0';
                phase_count <= "10";
        ELSIF (phase_count = "10") THEN
```

```
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        check_bit   <= '1';
                END IF;
                ------------------------------------------------------------
        WHEN B0_2 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        check_bit   <= '1';
                END IF;
                ------------------------------------------------------------
        WHEN B1_3 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        check_bit   <= '1';
                END IF;
                ------------------------------------------------------------
        WHEN B0_3 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        check_bit   <= '1';
                END IF;

                ------------------------------------------------------------
        WHEN B1_4 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        State       <= MSB;
                END IF;
                ------------------------------------------------------------
        WHEN B0_4 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        State       <= MSB;
                END IF;

        WHEN B1_5 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
                        S1          <= '0';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '1';
                        S7          <= '1';
                        S89         <= '0';
                        S10         <= '0';
                        phase_count <= "00";
                        State       <= MSB;
                END IF;
                ------------------------------------------------------------
        WHEN B0_5 =>
                IF (phase_count = "00") THEN
                        S1          <= '1';
                        S2          <= '1';
                        S3          <= '0';
                        S4          <= '0';
                        S5          <= '0';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '0';
                        S10         <= '1';
                        phase_count <= "01";
                ELSIF (phase_count = "01") THEN
                        S1          <= '0';
                        S2          <= '0';
                        S3          <= '0';
                        S4          <= '1';
                        S5          <= '1';
                        S6          <= '0';
                        S7          <= '0';
                        S89         <= '1';
                        S10         <= '0';
                        phase_count <= "10";
                ELSIF (phase_count = "10") THEN
```

```
                                                S1          <= '0';
                                                S2          <= '0';
                                                S3          <= '0';
                                                S4          <= '1';
                                                S5          <= '0';
                                                S6          <= '1';
                                                S7          <= '1';
                                                S89         <= '0';
                                                S10         <= '0';
                                                phase_count <= "00";
                                                State       <= MSB;
                                        END IF;
                        END CASE;
                END IF;
                ------------------------------------------------------------
                ------------------------------------------------------------
        END PROCESS;
END Behavioral;
```

A.2 Synthesis from VHDL code

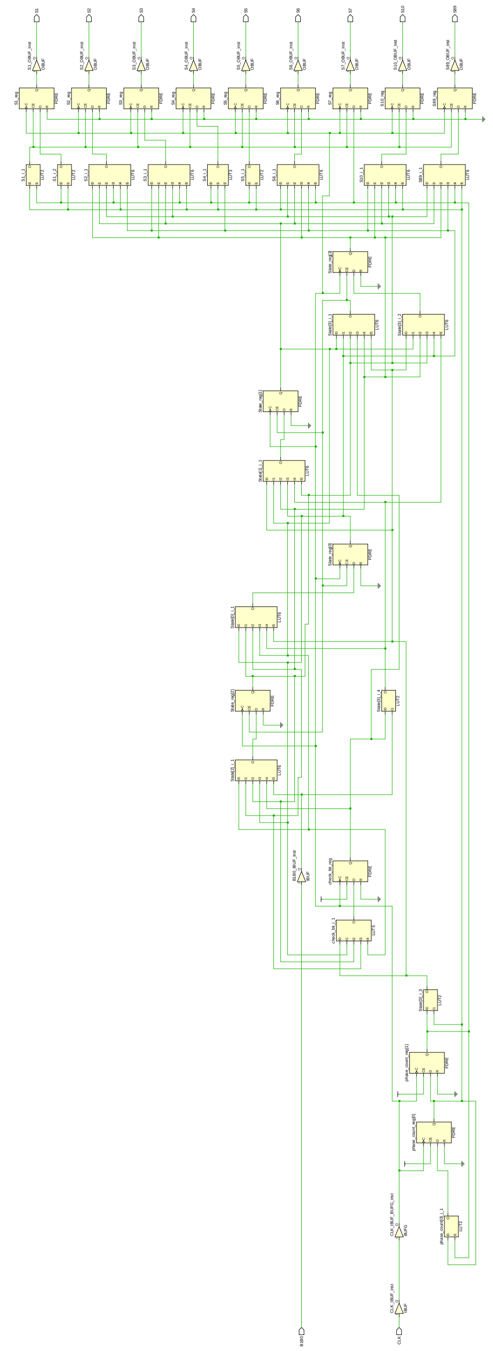


Figure A.1: Circuit extracted from synthesis.

A.3 Enlarged layout images

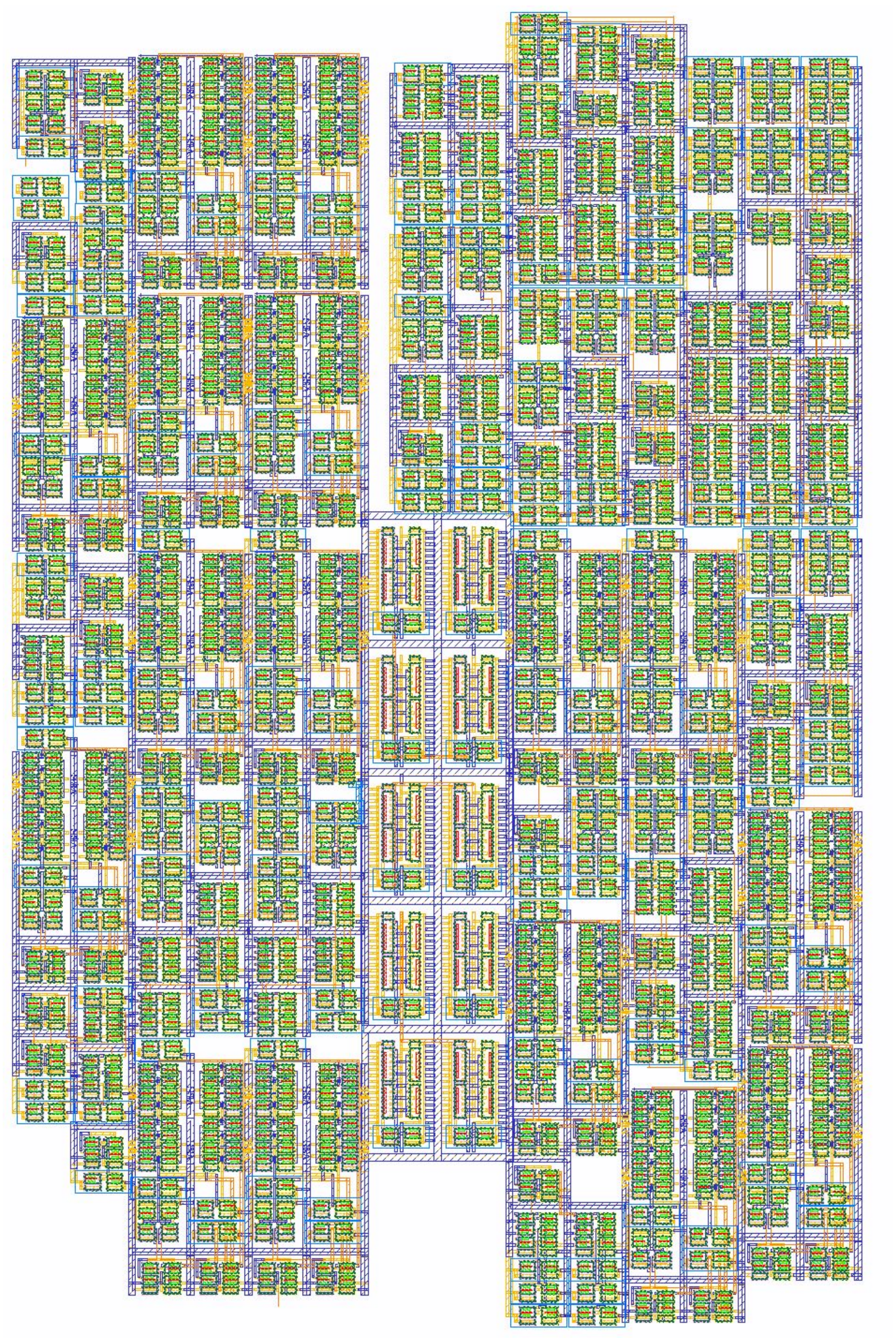


Figure A.2: Enlarged picture of the top layer.

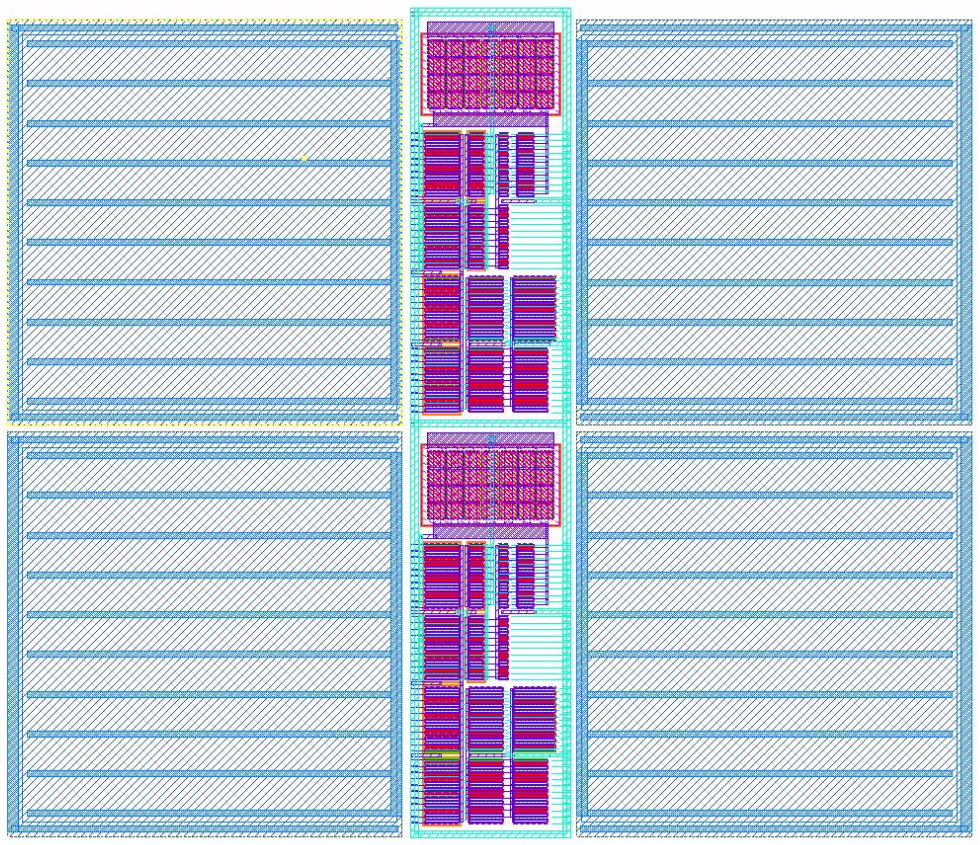


Figure A.3: Enlarged picture of the bottom layer..

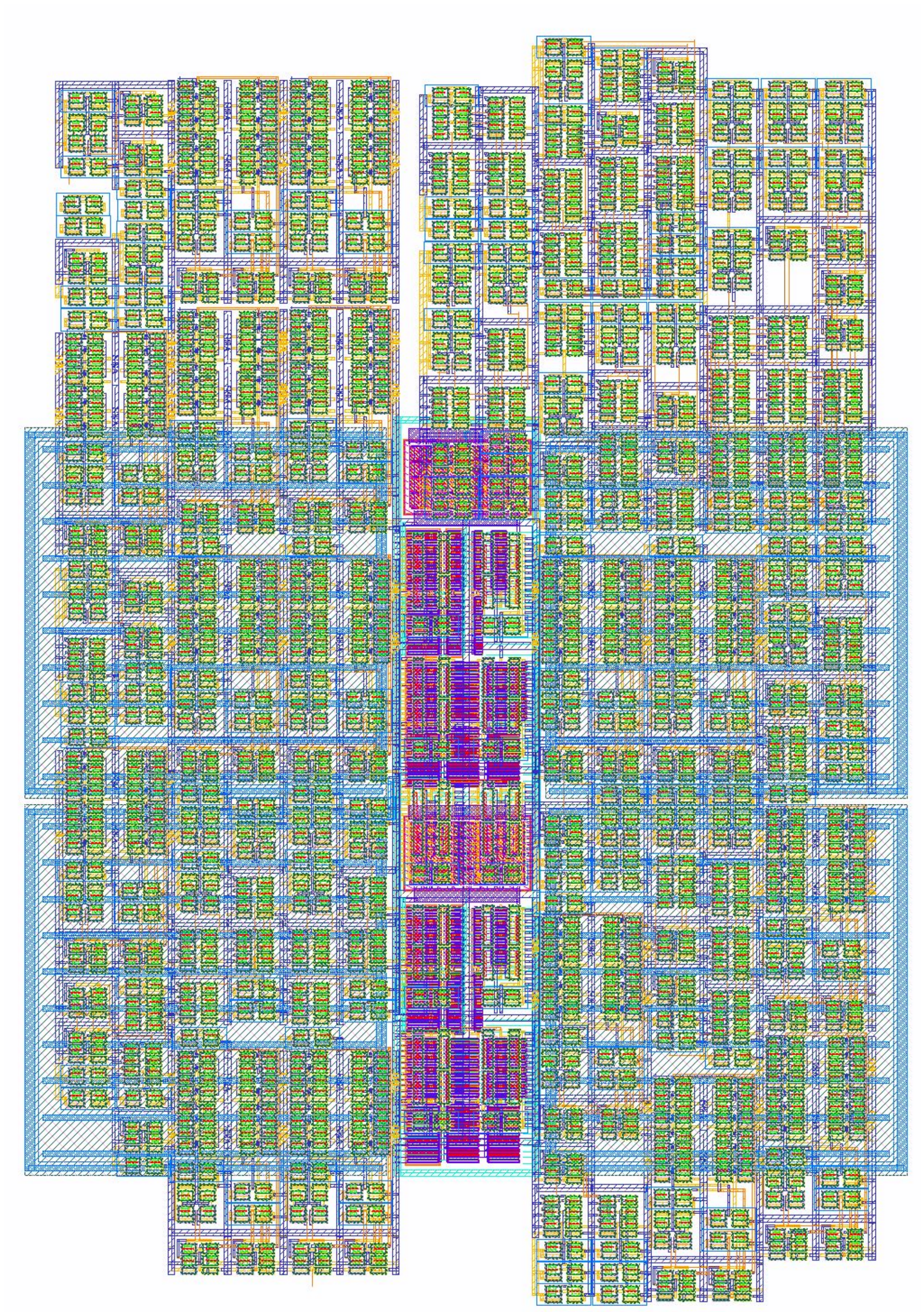


Figure A.4: Enlarged picture of top and bottom layer combined.

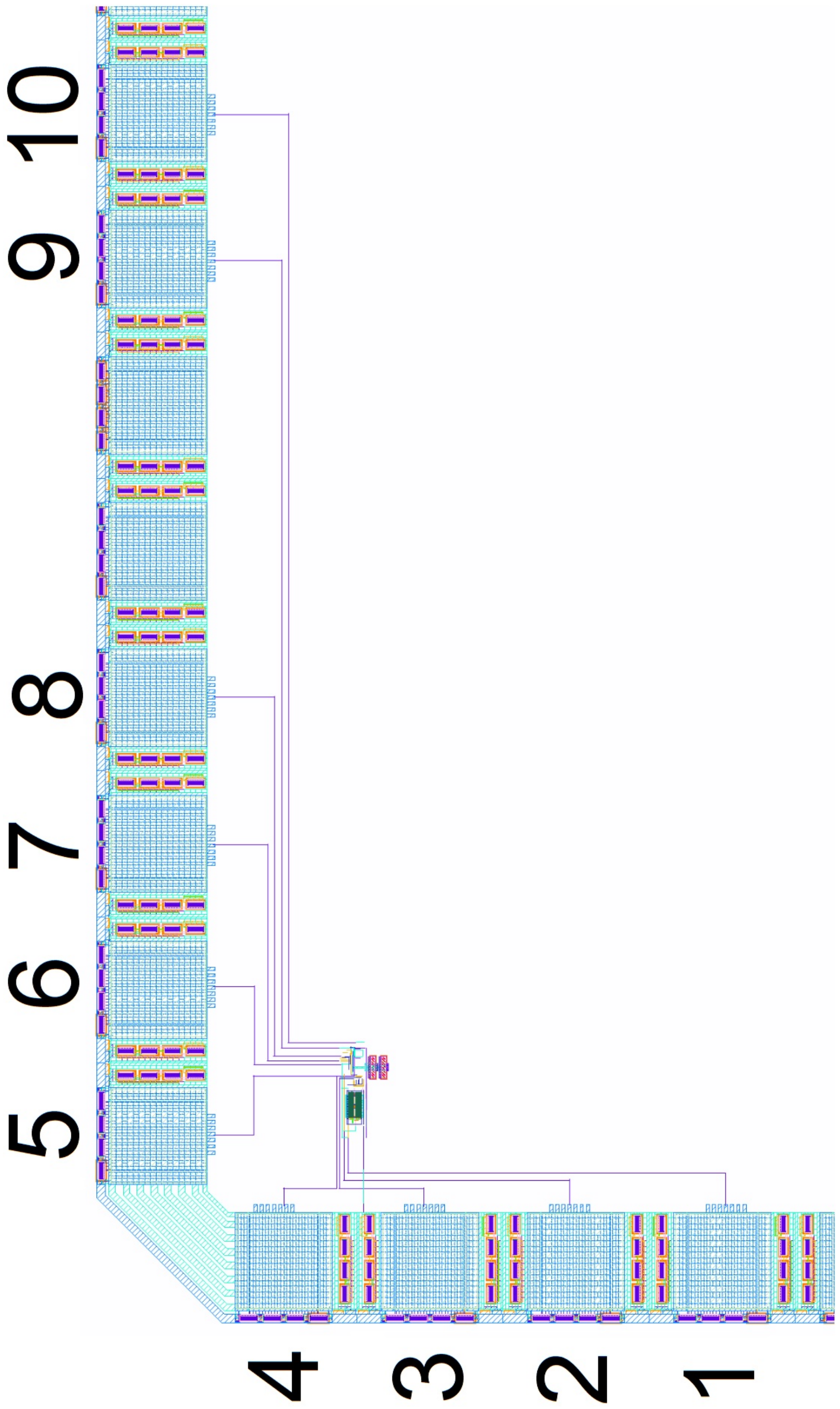


Figure A.5: Enlarged picture of the pinout illustration of the tape-out.

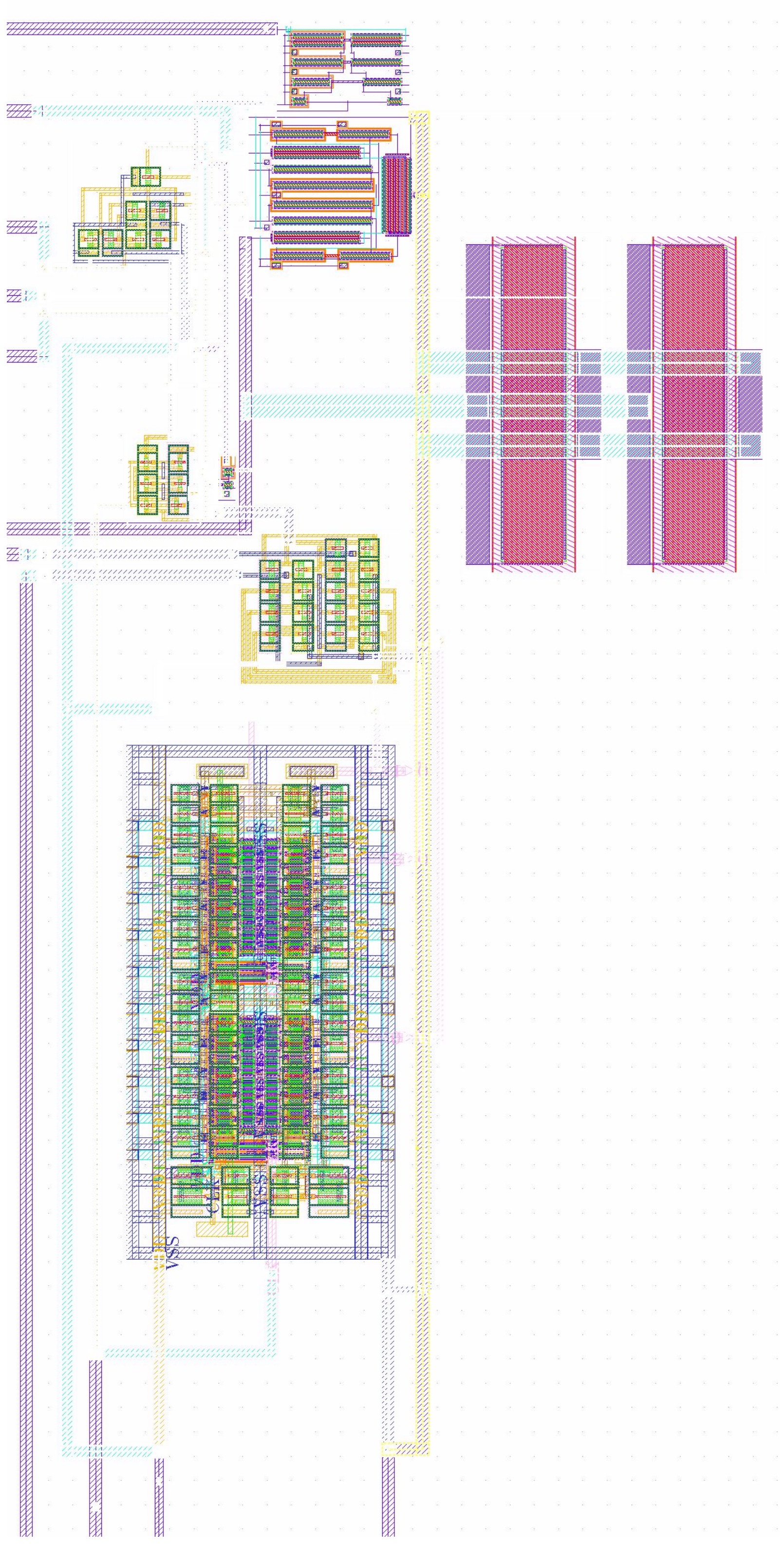


Figure A.6: Enlarged picture of the tape-out.