



# **Graded Exercise VLSI Design WS23/24: Current Mode (Pipeline/Algorithmic) ADC (version 2)**

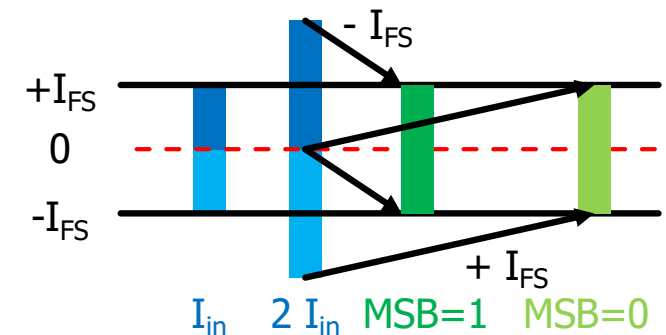
Prof. Dr. P. Fischer

Lehrstuhl für Schaltungstechnik und Simulation  
Uni Heidelberg



# Our ADC Principle

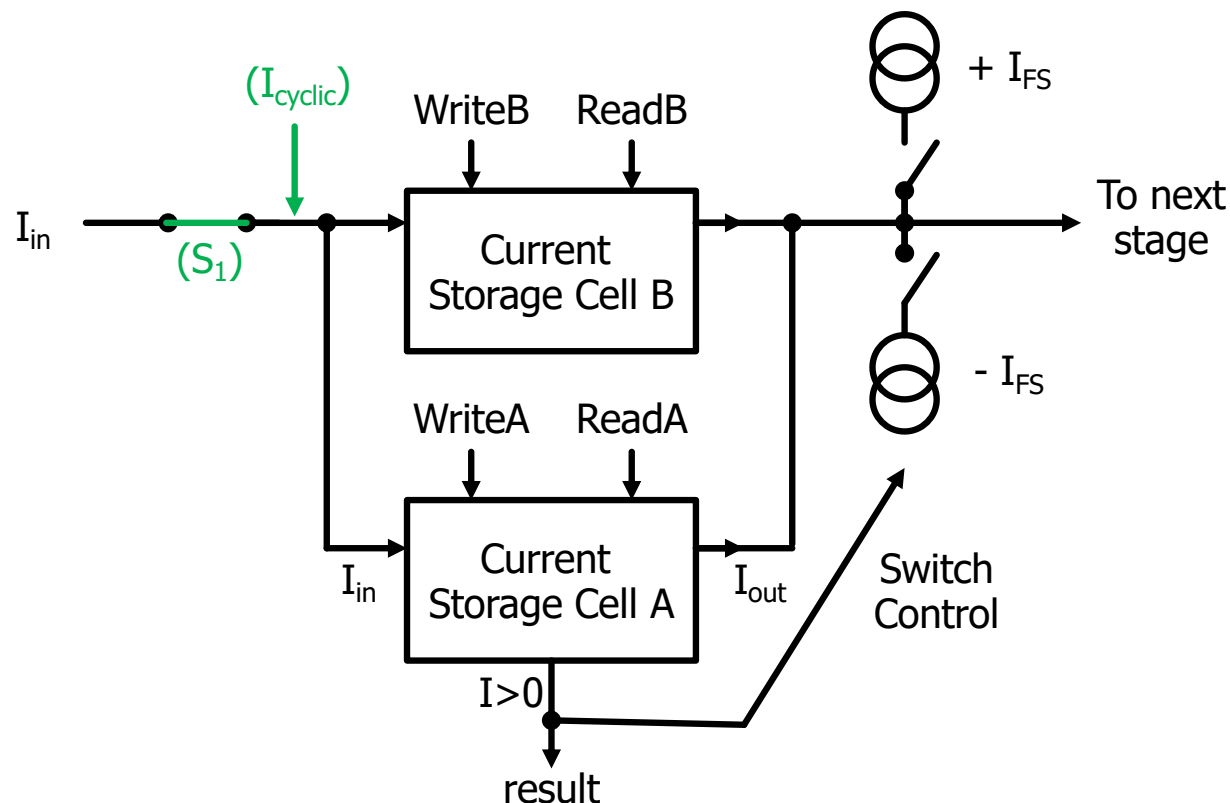
- An Analogue – Digital – Converter (ADC) converts an analogue quantity (we use a *current*) into a digital code.
- There are many different operation principles which each can be implemented in many different ways.
- Our *algorithmic / pipeline* ADC basically works like this:
  - The **input current**  $I_{in}$  ranges between  $-I_{FS}$  and  $I_{FS}$  (full scale value)
  - The **Most Significant Bit (MSB)** is obtained by comparing  $I_{in}$  to 0
  - $I_{in}$  is multiplied by 2
  - If the **MSB=1**,  $I_{FS}$  is subtracted
  - If the **MSB=0**,  $I_{FS}$  is added
  - This generates a new current again in the range  $-I_{FS} \dots I_{FS}$ .
  - These steps are repeated to produce one more bit per 'stage'





# Block diagram

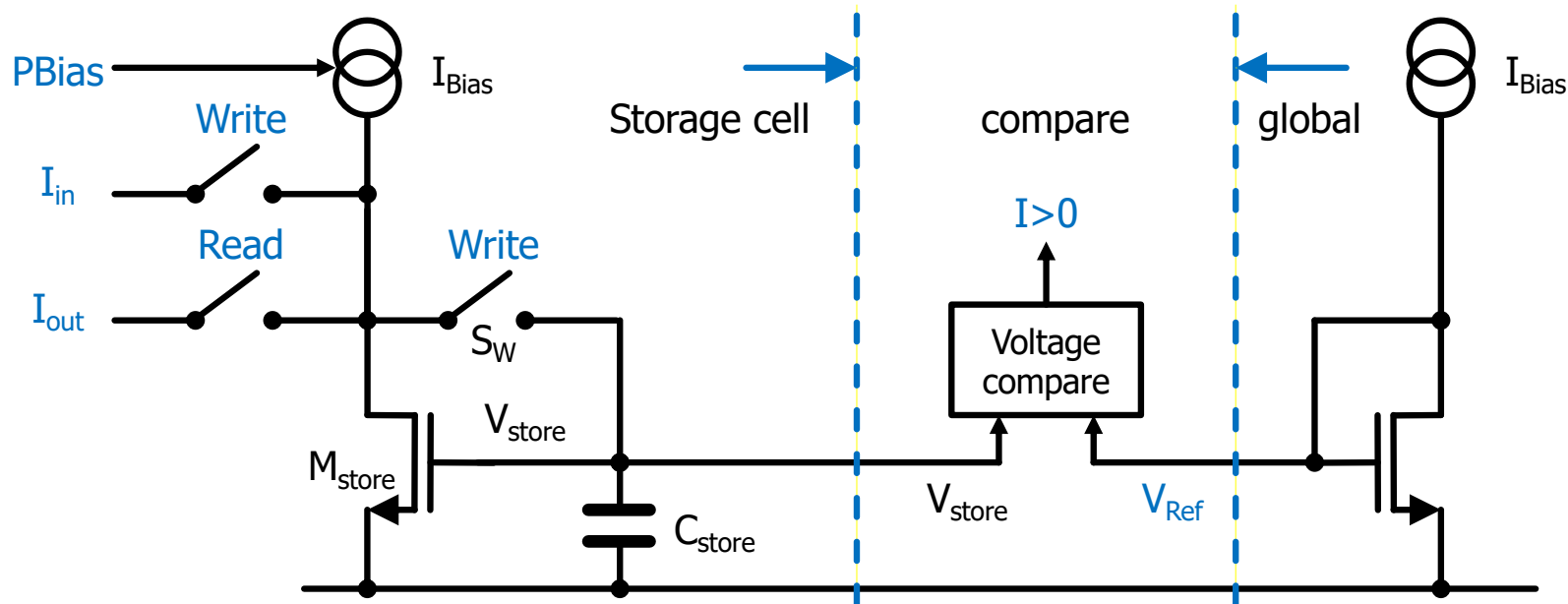
- The heart of our ADC is a *current storage cell*
  - We can send an input into the cell and 'write' it
  - The cell delivers the stored current at its output when we 'read'
  - It also tells us whether the (stored) current is  $>0$





# The Current Storage Cell

- When write is active (switches closed),  $I_{in}$  flows into the diode connected NMOS  $M_{store}$
- When writing is stopped, the gate voltage  $V_{store}$  is memorized on capacitor  $C_{store}$  (and the gate capacitance of  $M_{store}$ )
- This current can be read at  $I_{out}$  If switch 'read' is closed
- All switches can be NMOS (understand why!)
- A constant bias current  $I_{bias} > I_{FS}$  provides a standing current for  $M_{store}$  and enables bipolar operation (i.e. positive and negative input currents). Understand this!
- The stored voltage is compared to the gate voltage of an identical NMOS with *no* extra input current. This is the 'comparison to 0'. Understand this! The comparison voltage  $V_{ref}$  can be global (common for all cells, generated outside of the cell).





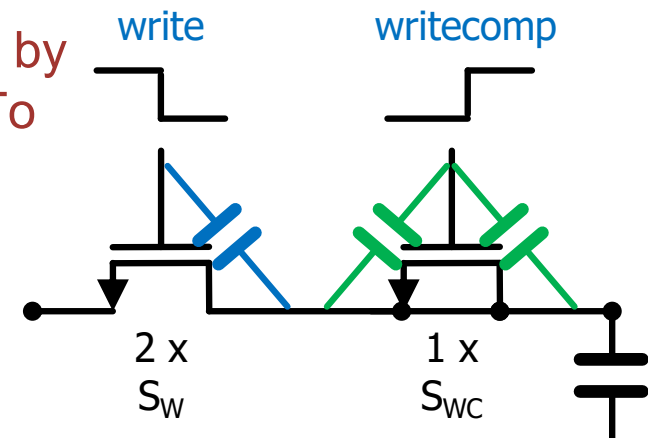
## Task 1: Simulate the Storage cell

- Use for instance  $I_{FS} = 20\mu A$ .
- Chose a sensible value for  $I_{Bias}$
- Simulate the storage cell. Inject a current  $I_{in} = -I_{FS} \dots I_{FS}$  at the input and check that you see 'roughly' this current at the output.
- Note:
  - When you inject current with an ideal 'idc' current source, make sure that the input voltage does not increase to unrealistic values when the current cannot flow!
  - When reading, the output voltage must be held at a potential which allows current flow in both direction, for instance  $VDD/2 = 0.9 V$ . Even better this could be the potential which will be present when writing another cell.
- Make sure  $V_{store}$  does not raise too high for large currents (operation of the switches!). Chose the size of  $M_{store}$  accordingly
- You will notice that  $I_{out}$  is not be *exactly*  $I_{in}$ .
  - How good is the cell?
  - What are reasons for the deviation?



## Task 2: Improve the cell

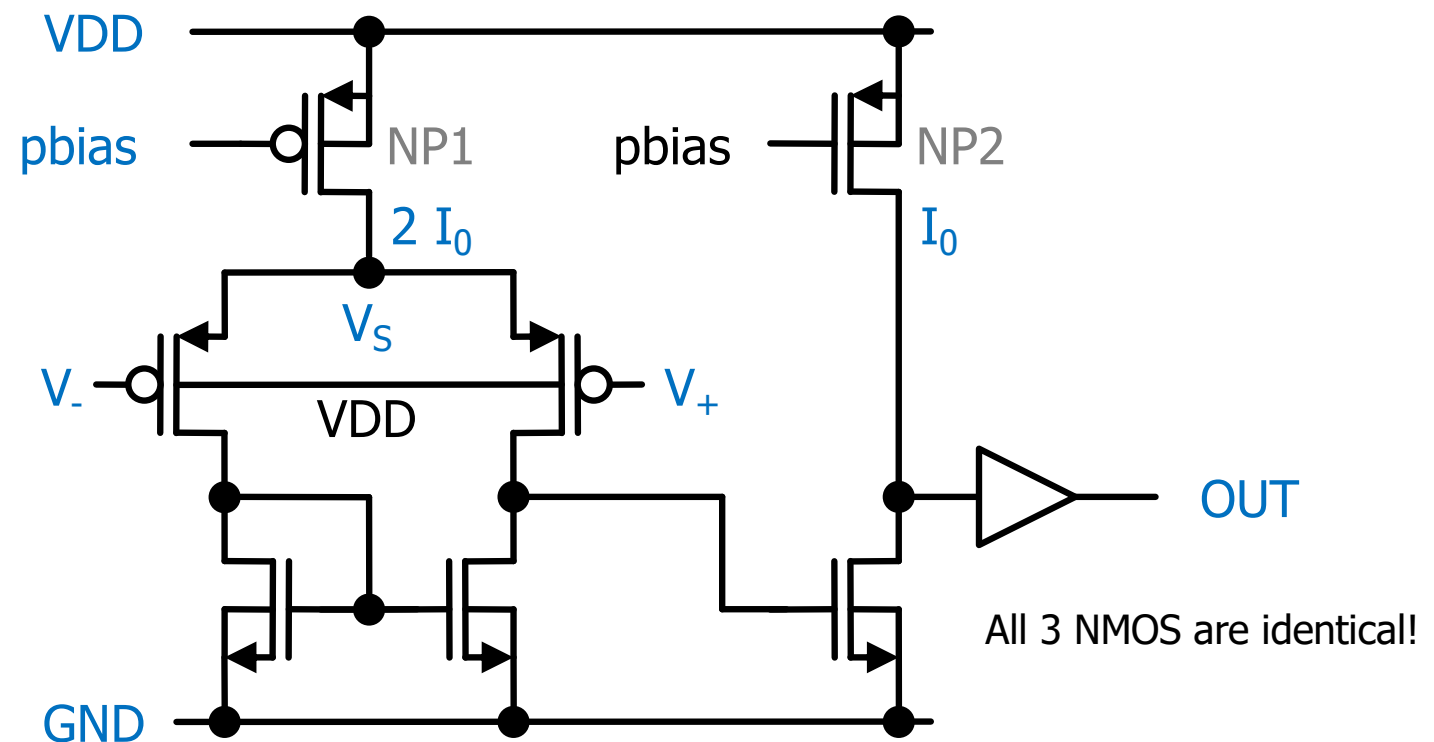
- Make  $M_{\text{store}}$  long enough so that the Early effect does not matter too much. (Make sure it remains wide enough...)
- You will notice that the falling edge of *write* couples from the (NMOS) switch  $S_W$  into the storage node. This is a significant source of error. A trivial counter-measure is a large  $C_{\text{store}}$ , but this makes the cell slow.
- A better, common trick to alleviate this 'charge injection' problem is to use a shorted *compensation* NMOS of half size clocked with a complementary edge *writecomp*. Both signal edges (*write*, *writecomp*) should be fast (say, 100ps rise/fall time). The compensation edge should best come a bit after the injection edge. (You may use a few slow inverters to achieve this...)
- (The charge injected by the Gate-Drain capacitance  $C_{\text{GD}}$  of  $S_W$  is compensated by the charges from  $C_{\text{GD}}$  and  $C_{\text{GS}}$  of  $S_{\text{WC}}$ . To make the charges equal,  $S_W$  must be twice as large as  $S_{\text{WC}}$ .)
- Does this help?





# The Comparator

- For the voltage comparison, we can use a very simple comparator with
  - a (PMOS) differential pair
  - followed by a (NMOS) gain stage
- This part needs an extra current bias.





# Comparator Details

- The comparator is biased with a 'not too short' PMOS current source NP1 ( $L \sim 1\mu\text{m}$ ).
  - Generate the bias voltage  $p_{\text{bias}}$  by a current mirror globally for all comparators of the ADC!
  - Use a current of – for instance –  $\sim 10\mu\text{A}$
  - For simplicity, generate the input current to the (global) mirror with a resistor to ground.
- The second stage should be biased with *half the current* of the first stage so that its threshold is set to the condition  $V_+ = V_-$ . Try to understand why! Look at currents and voltages just at the the switching point.
- This comparator can handle voltages at its inputs from (just) ground up to some positive voltage, but not up to  $V_{DD}$ . This is what we need (why?)





## Task 3: The comparator

- Make a symbol & schematic of the comparator
- Simulate it (dc, transient)
  - Make sure it works ok for typical input voltages  $V_{\text{ref}}$
  - Does it have a systematic voltage offset?
- Simulate the storage cell *with* the comparator and check that the digital output is correct. Try to quantify the error.



## Task 4: All together: One Stage

- Put together
  - two current storage cells
  - one without comparator
  - the extra add/subtract sources (using good mirrors).
- Generate the biases for storage, comparators and subtract sources (which are the same in all stages) in one global blocks
- Simulate the stage
  - Check that the MSB is 'more or less' correct and that the output current is again in the range  $-I_{FS}..I_{FS}$ .



## Task 5: The ADC

- Cascade several stages to generate an ADC with some ( $\geq 4$ ) bits resolution
- Make a well structured schematic hierarchy (global bias cell, storage, comparator, sources,...)
- Control everything with a Verilog code with a mixed mode simulation
  - You first have to write twice to the first stage, then read it (and add/subtract  $I_{FS}$ ). This signal then goes to the second stage and so on.
  - You may notice that the polarities get mixed up between stages...
- Try to extract a transfer characteristic, i.e. the digital output code vs. the analog input voltage.
  - What are errors of the ADC ?
  - Can you find some sources of the errors ?



## Task 7: Layout

- Make layouts of the cells.
  - Try to be reasonable compact (but you do not need to exaggerate). Where appropriate, make sure transistors match.
  - Decide on a reasonable hierarchy
  - If you need  $C_{\text{store}}$  (i.e. if the gate cap. is not sufficient), just use an extra NMOS gate in your layout.
- Then make the layout of the 'One Stage' cell such that you can cascade an arbitrary number of stages without extra cabling by placing them side by side (in x).
  - → The cell should be better by high (in y) and not so wide in x.
  - Think on how to run the various signals and on which layers.
- The global part could derive all bias voltages from one current and be placed at the side (with no extra wires).
- Your layout should be DRC and LVS clean.



# Report

- Provide a document (~ 10 pages) with
  - Explanations of some of your design choices (transistor and capacitor sizes, bias,...)
  - Schematics (just screen shots), using a decent hierarchy. Explain your hierarchy!
  - Simulation results (function, speed, errors)
  - Results of the mixed mode simulation + Verilog code
  - Answers to some of the 'understand this' questions in the text
  - A 'nice' layout (not too large, matched transistors, good placement for short connections,...) (screenshot)  
Explain your reasonings and choices!



## Bonus Exercise A

- In the described ADC, all stages operate simultaneously, starting at the MSB. Only when all stages are correctly switched, the result is available and a new value can be converted.
- By adding another 'pure' storage cell after each stage, the whole process can be pipelined, i.e.
  - The input current is by the MSB stage
  - The result is moved to the storage cell of the MSB
  - Now the MSB stage can process *another* value while the stored value is passed to the next stage
  - And so on.
- Implement this, at least in simulation
  - Note that the digitized result must be composed from time-shifted values.



## Bonus Exercise B

- If you are motivated, you could try to make a *cyclic* ADC (only schematic & simulation):
  - Such an ADC uses only *ONE* stage + storage, being used several times.
  - The stored output current is fed back into the input of the stage (this is why I have indicated switch  $S_1$  and the green arrow on page 3)
- As before, this needs add a *third* storage cell 'C':
  1.  $I_{in}$  is written to A, then to B
  2. The corrected result ( $2I_{in} \pm I_{FS}$ ) is written to C
  3. Instead of  $I_{in}$ , C is written to A, then B.
  4. For a number of cycles, go to 2.
  5. After the complete conversion, go to 1 for the next  $I_{in}$



## Bonus Exercise C

- Instead of using two storage cells, you could use a single cell which outputs the doubled current by means of a current mirror
- Try to make this work





# Grading

- You can achieve the best grade 1.0 with a very good treatment and documentation of the main exercise.
- Deficiencies in your solution can be compensated by – also partial – treatment of some bonus exercises.