TFE4152 - Lecture 6

Project

Source

Goal for today

Make it easier to understand the project

Project

- 30 % of final grade
- Groups of 2 people. Find a partner soon. Sign up on blackboard.
- Deadline: 19'th November before 12:00 (24 hour format).
- Strict deadline, $t > 12:00 \equiv fail$. Both members in group must submit report.

Goal

Be inspired by the ISSCC paper, and design a similar system.

Design analog circuits in SPICE

Design digital circuits in SystemVerilog

A 10 000 Frames/s CMOS Digital Pixel Sensor

IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 36, NO. 12, DECEMBER 2001

A 10 000 Frames/s CMOS Digital Pixel Sensor

Stuart Kleinfelder, SukHwan Lim, Xinqiao Liu, and Abbas El Gamal, *Fellow, IEEE*

***Abstract*—A 352 × 288 pixel CMOS image sensor chip with per-pixel single-slope ADC and dynamic memory in a standard digital 0.18-μm CMOS process is described. The chip performs "snapshot" image acquisition, parallel 8-bit A/D conversion, and digital readout at continuous rate of 10 000 frames/s or 1 Gpixels/s with power consumption of 50 mW. Each pixel consists of a photogate circuit, a three-stage comparator, and an 8-bit 3T dynamic memory comprising a total of 37 transistors in 9.4 × 9.4 μm with a fill factor of 15%. The photogate quantum efficiency is 13.6%, and the sensor conversion gain is 13.1 μV/e$^-$. At 1000 frames/s, measured integral nonlinearity is 0.22% over a 1-V range, rms temporal noise with digital CDS is 0.15%, and rms FPN with digital CDS is 0.027%. When operated at low frame rates, on-chip power management circuits permit complete powerdown between each frame conversion and readout. The digitized pixel data is read out over a 64-bit (8-pixel) wide bus operating at 167 MHz, i.e., over 1.33 GB/s. The chip is suitable for general high-speed imaging applications as well as for the implementation of several still and standard video rate applications that benefit from high-speed capture, such as dynamic range enhancement, motion estimation and compensation, and image stabilization.**

***Index Terms*—ADC, CMOS image sensor, digital pixel sensor, high-speed imaging, image sensor, memory.**

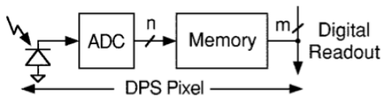


Fig. 1. Simple DPS pixel block diagram.

rate applications that require high-speed capture such as sensor dynamic range enhancement and motion estimation [8]–[10].

The main drawback of DPS is that it uses more transistors per pixel than conventional analog image sensors and therefore can have larger pixel sizes. Since there is a lower bound on practical pixel size imposed by the wavelength of light, imaging optics, and dynamic range, this drawback quickly disappears as CMOS technology scales down to 0.18 μm and below. Designing image sensors in such advanced technologies, which will be needed for implementing true camera-on-chip systems, is challenging due to the scaling of supply voltage and the increase in leakage currents [12].

In this paper, we describe a 352 × 288 CMOS DPS with per-pixel ADC and digital memory fabricated in a standard

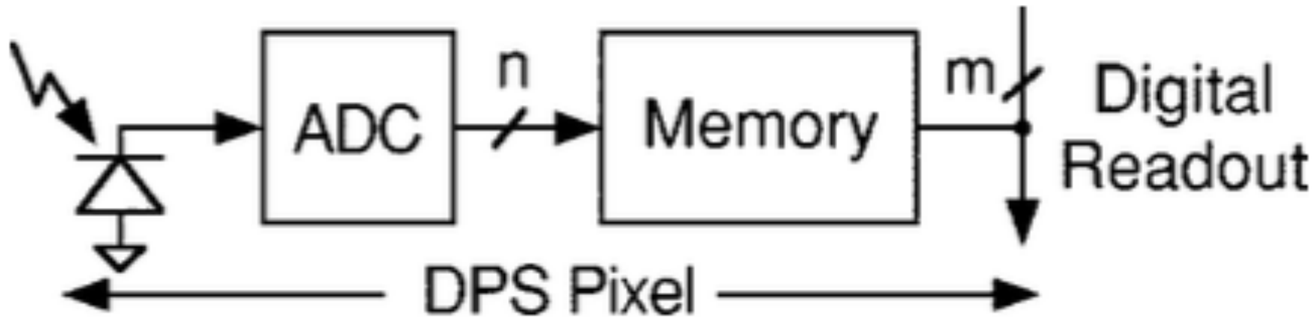


Fig. 1. Simple DPS pixel block diagram.

photon sensor $\Rightarrow$ local analog to digital converter $\Rightarrow$ local memory

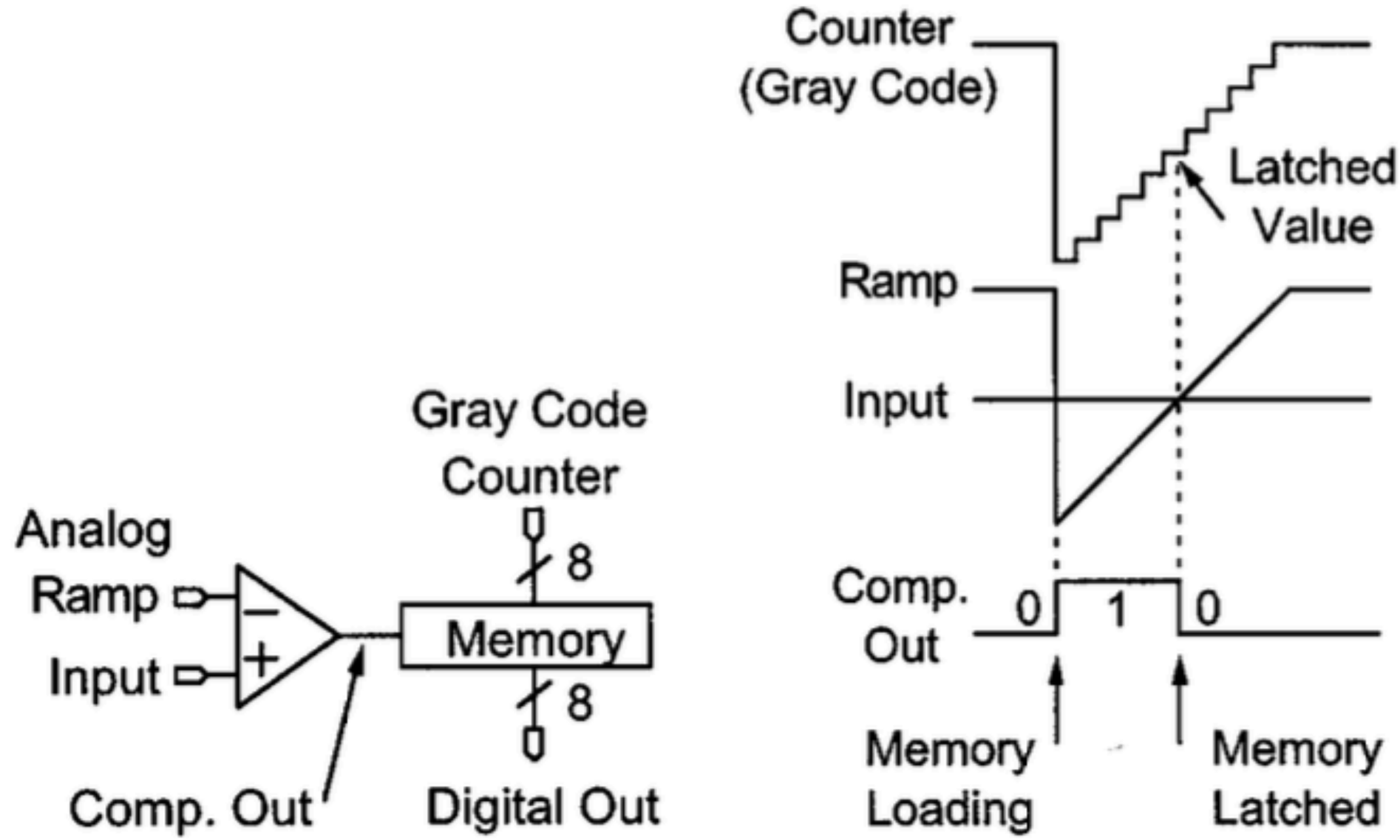
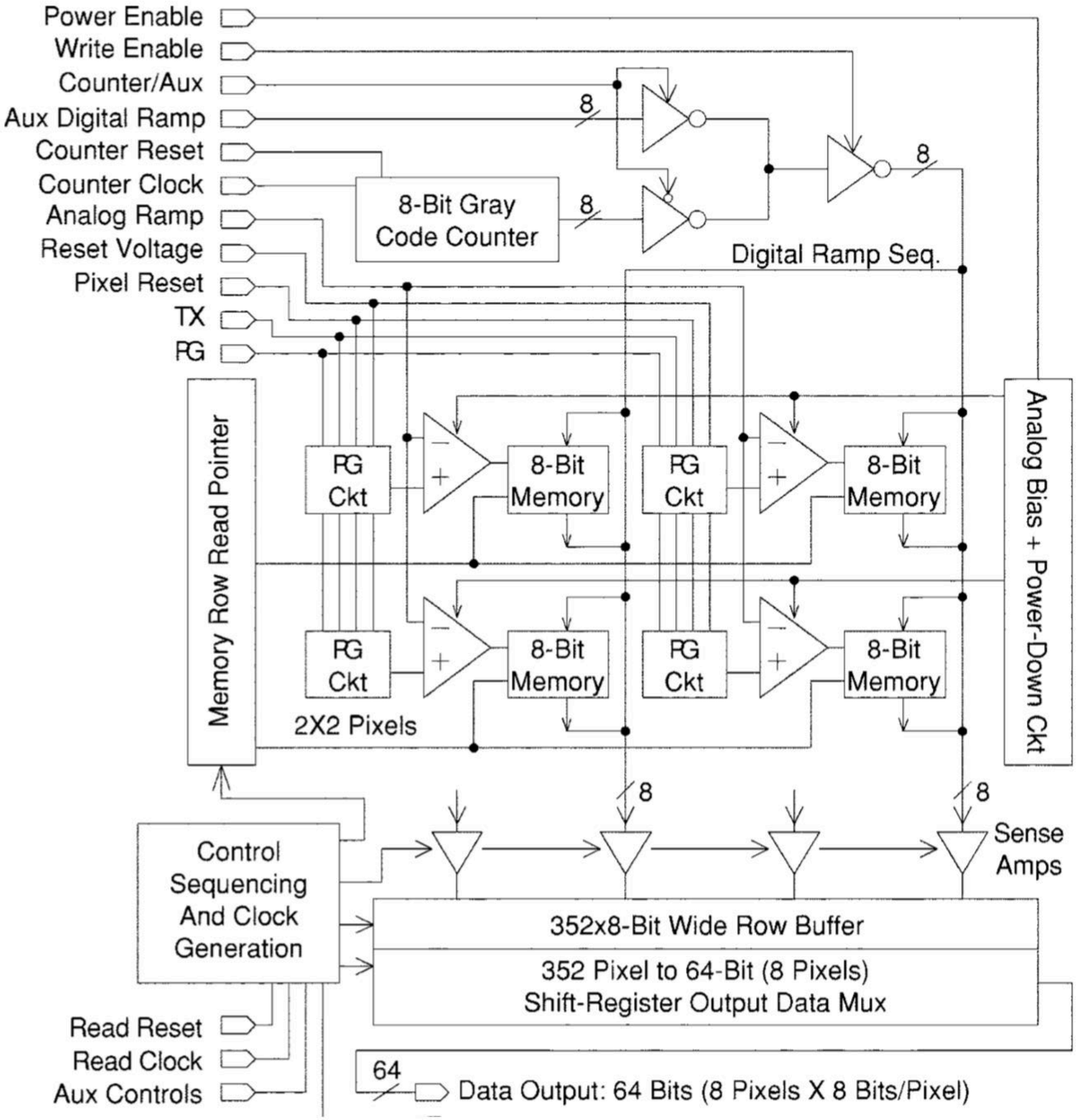
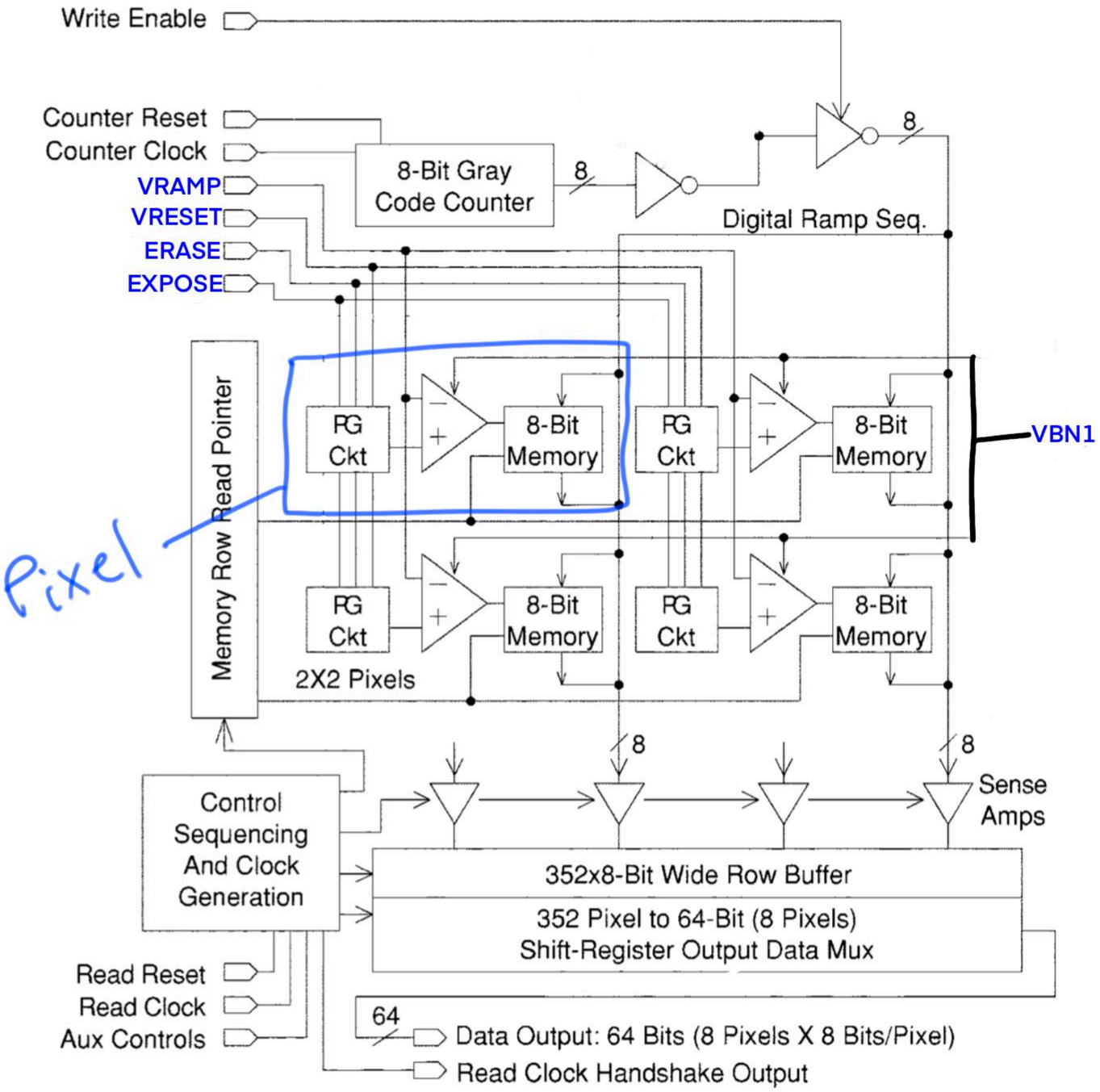


Fig. 6. Single-slope ADC operation.


Power Enable
Write Enable
Counter/Aux
Aux Digital Ramp
Counter Reset
Counter Clock
Analog Ramp
Reset Voltage
Pixel Reset
TX
PG
8
8-Bit Gray Code Counter
8
8
Digital Ramp Seq.
Memory Row Read Pointer
PG Ckt
8-Bit Memory
PG Ckt
8-Bit Memory
PG Ckt
8-Bit Memory
PG Ckt
8-Bit Memory
2X2 Pixels
Analog Bias + Power-Down Ckt
8
8
Sense Amps
Control Sequencing And Clock Generation
352x8-Bit Wide Row Buffer
352 Pixel to 64-Bit (8 Pixels) Shift-Register Output Data Mux
Read Reset
Read Clock
Aux Controls
64
Data Output: 64 Bits (8 Pixels X 8 Bits/Pixel)


Write Enable
Counter Reset
Counter Clock
VRAMP
VRESET
ERASE
EXPOSE
8-Bit Gray Code Counter
8
8
Digital Ramp Seq.
Memory Row Read Pointer
PG Ckt
8-Bit Memory
VBN1
Pixel
2X2 Pixels
8
8
Sense Amps
Control Sequencing And Clock Generation
352x8-Bit Wide Row Buffer
352 Pixel to 64-Bit (8 Pixels) Shift-Register Output Data Mux
Read Reset
Read Clock
Aux Controls
64
Data Output: 64 Bits (8 Pixels X 8 Bits/Pixel)
Read Clock Handshake Output

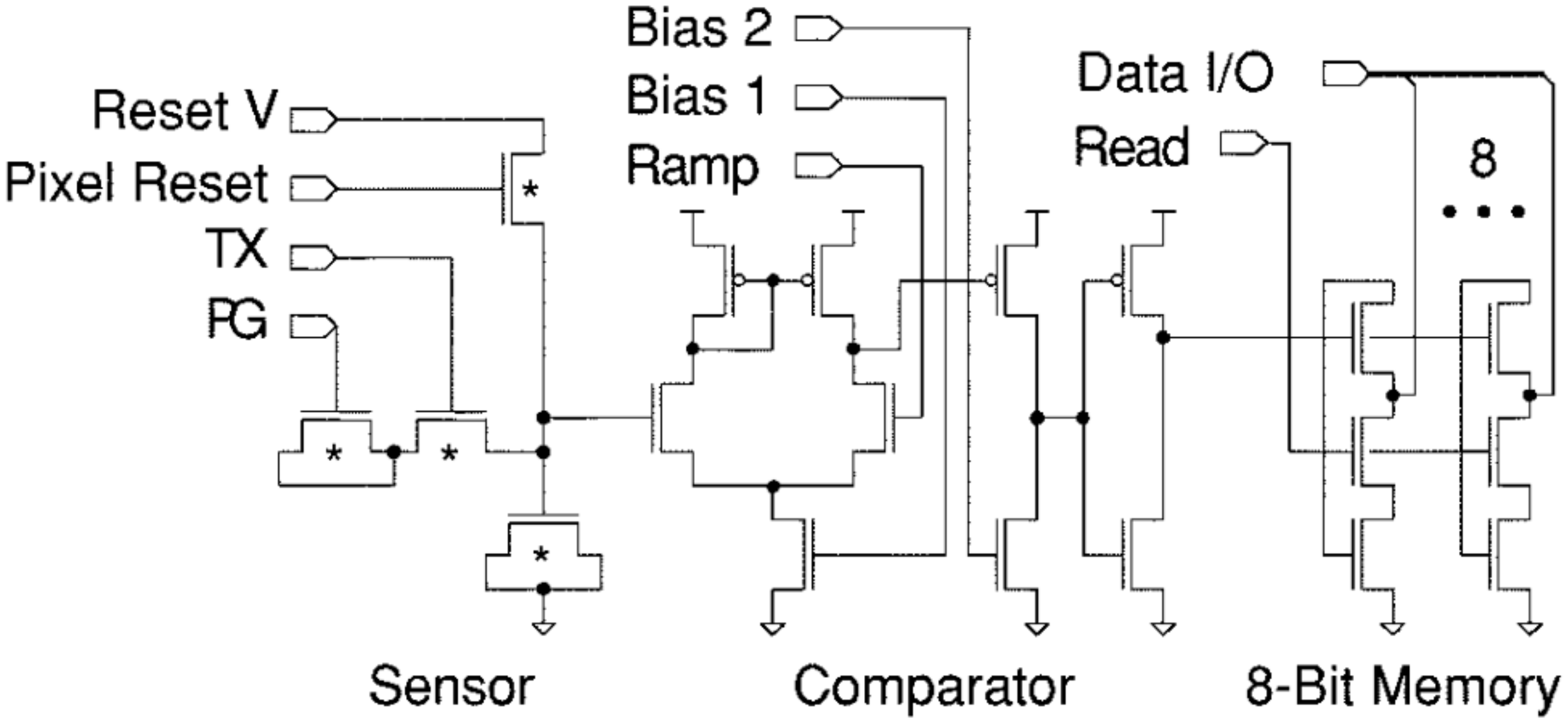
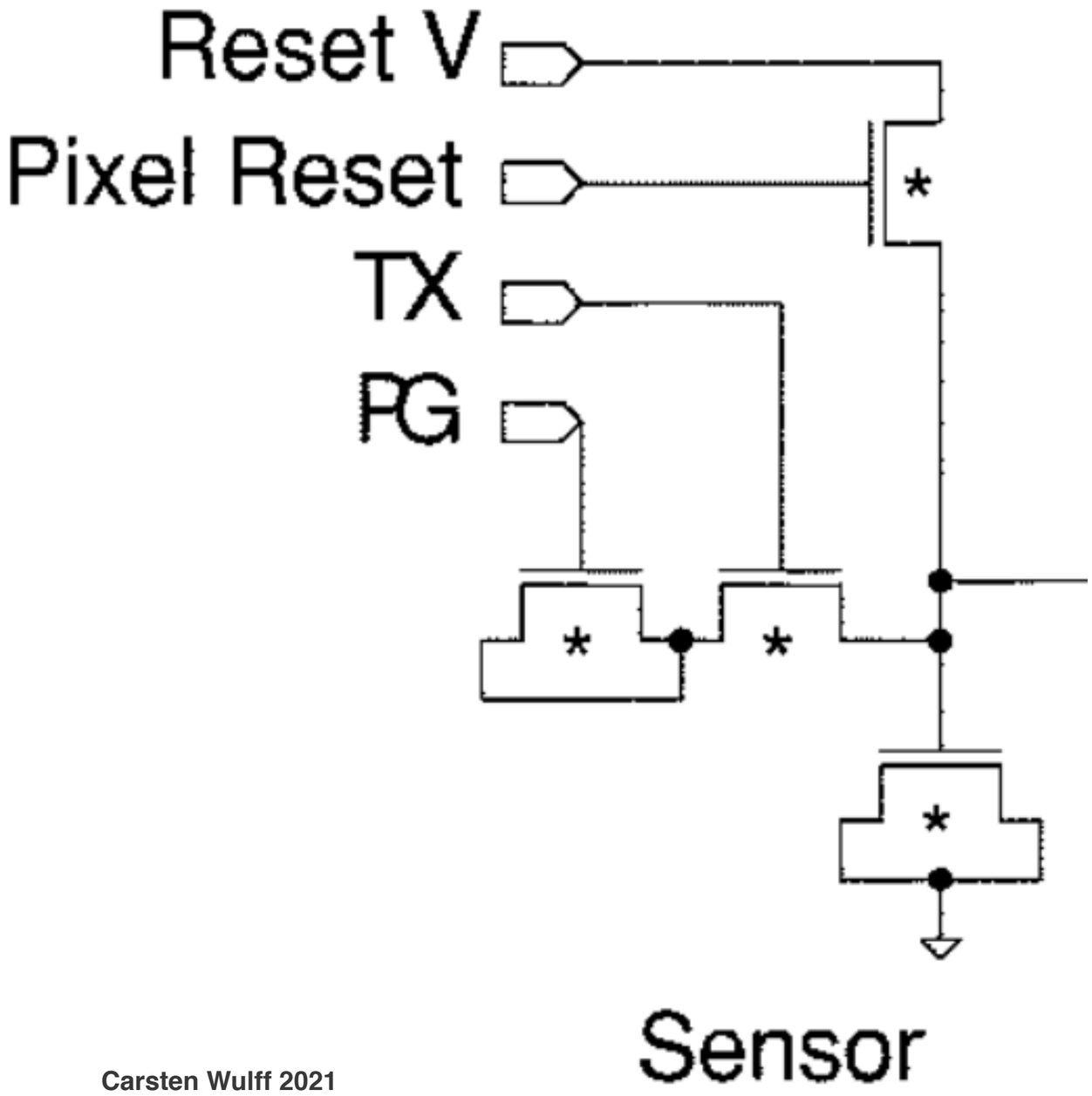
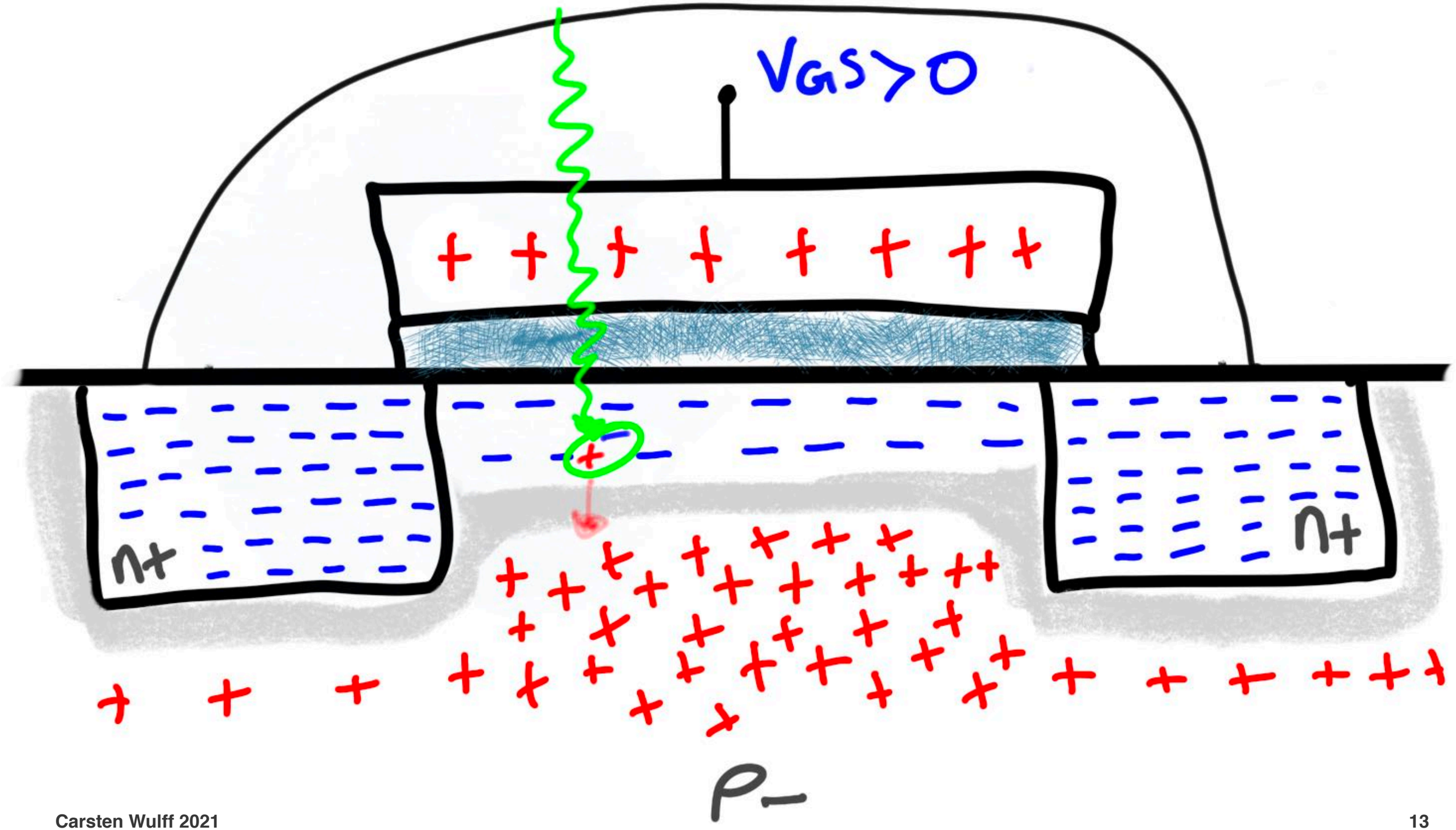
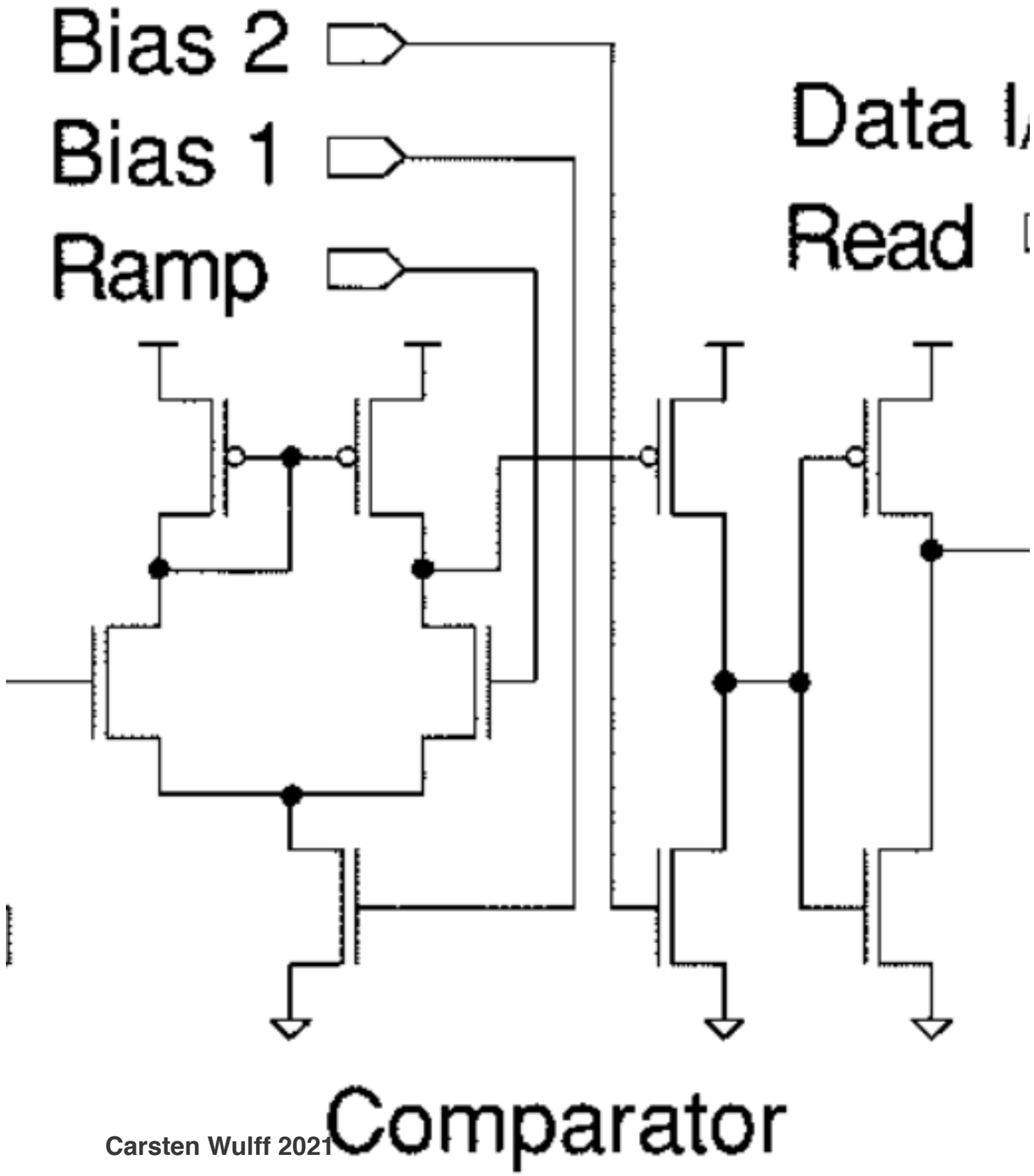


Fig. 4. Pixel schematic.

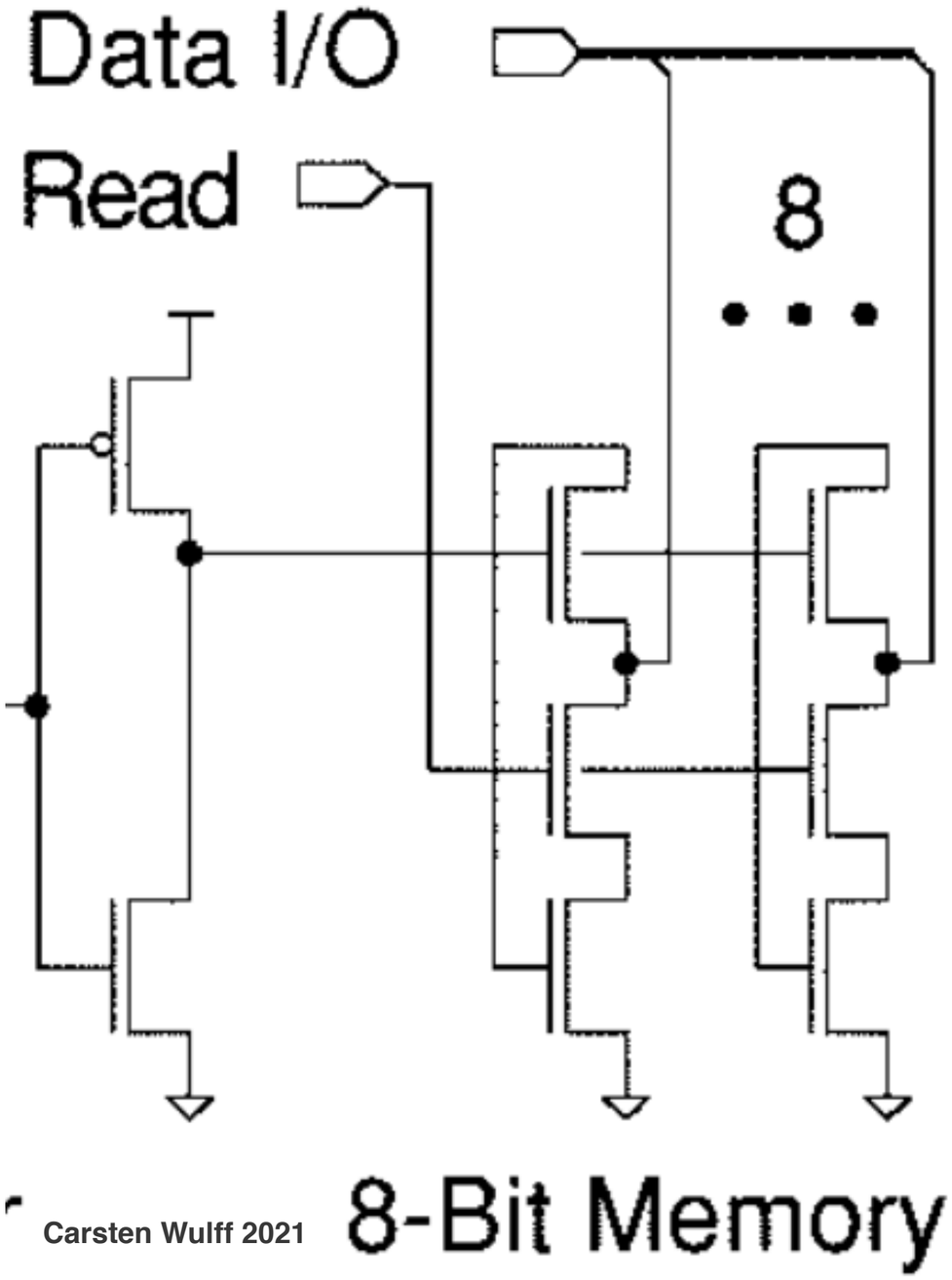


SENSOR


VGS>0
n+
n+
p-



COMP



MEMORY

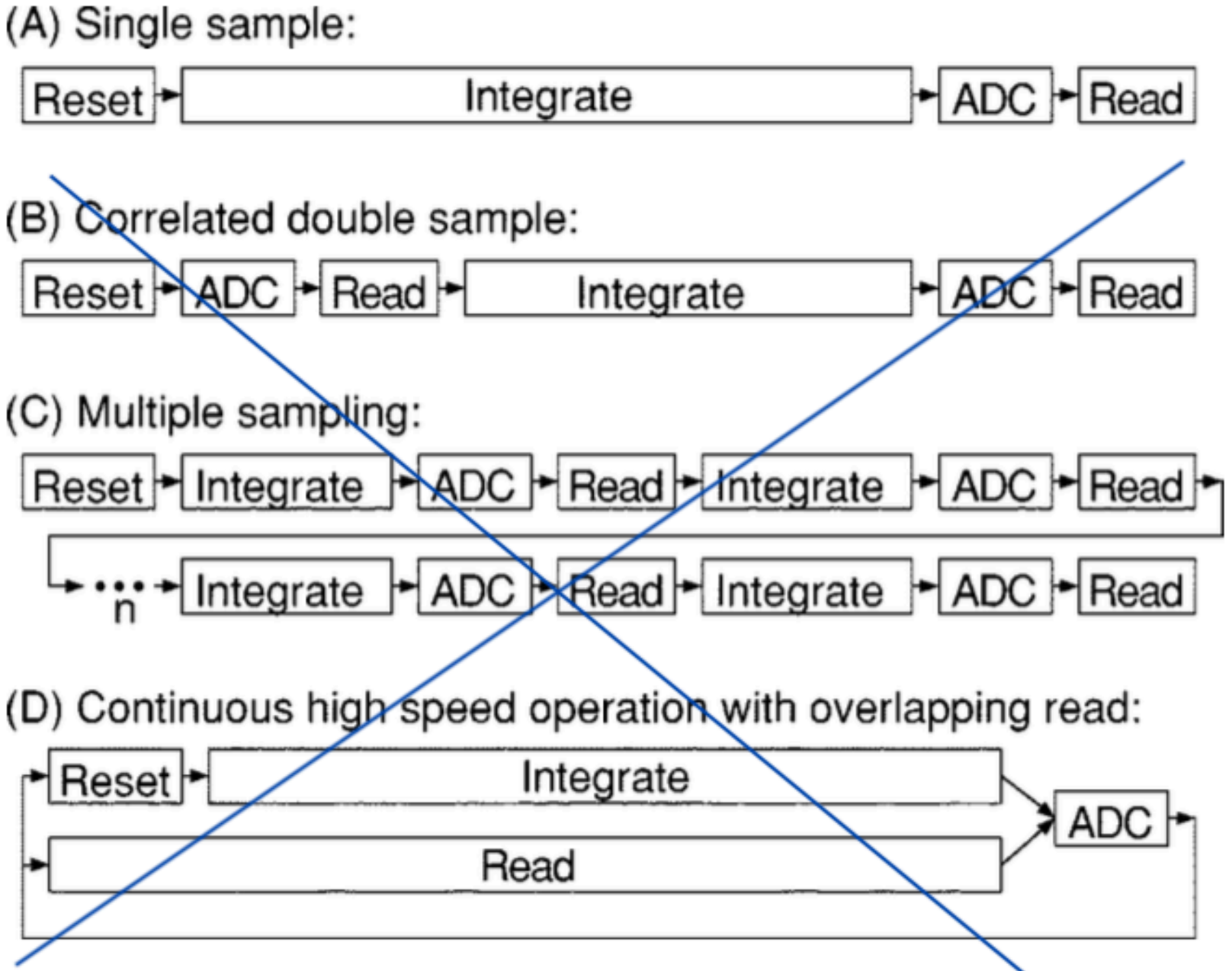
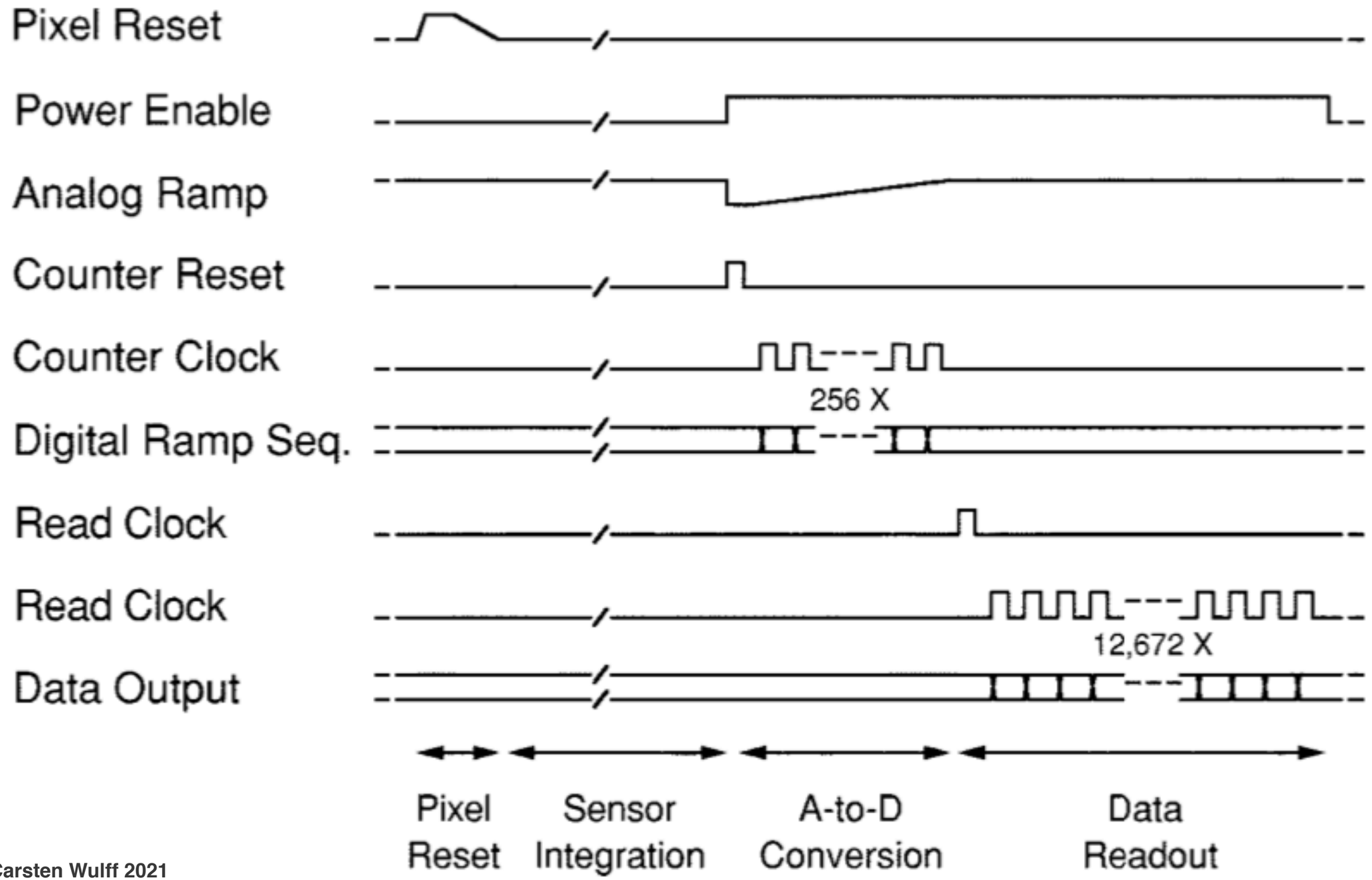


Fig. 9. Basic DPS operation schemes.


Pixel Reset
Power Enable
Analog Ramp
Counter Reset
Counter Clock
256 X
Digital Ramp Seq.
Read Clock
Read Clock
12,672 X
Data Output
Pixel Reset
Sensor Integration
A-to-D Conversion
Data Readout

Minimum implementation

- Model of 2 x 2 pixel array in SystemVerilog
- State machine to control reset, exposure, analog-to-digital conversion, and readout of the pixel array
- SPICE of pixel sensor (sensor, comparator)
- Report documenting that the circuits (analog and digital) work as designed

Things it's OK to ignore

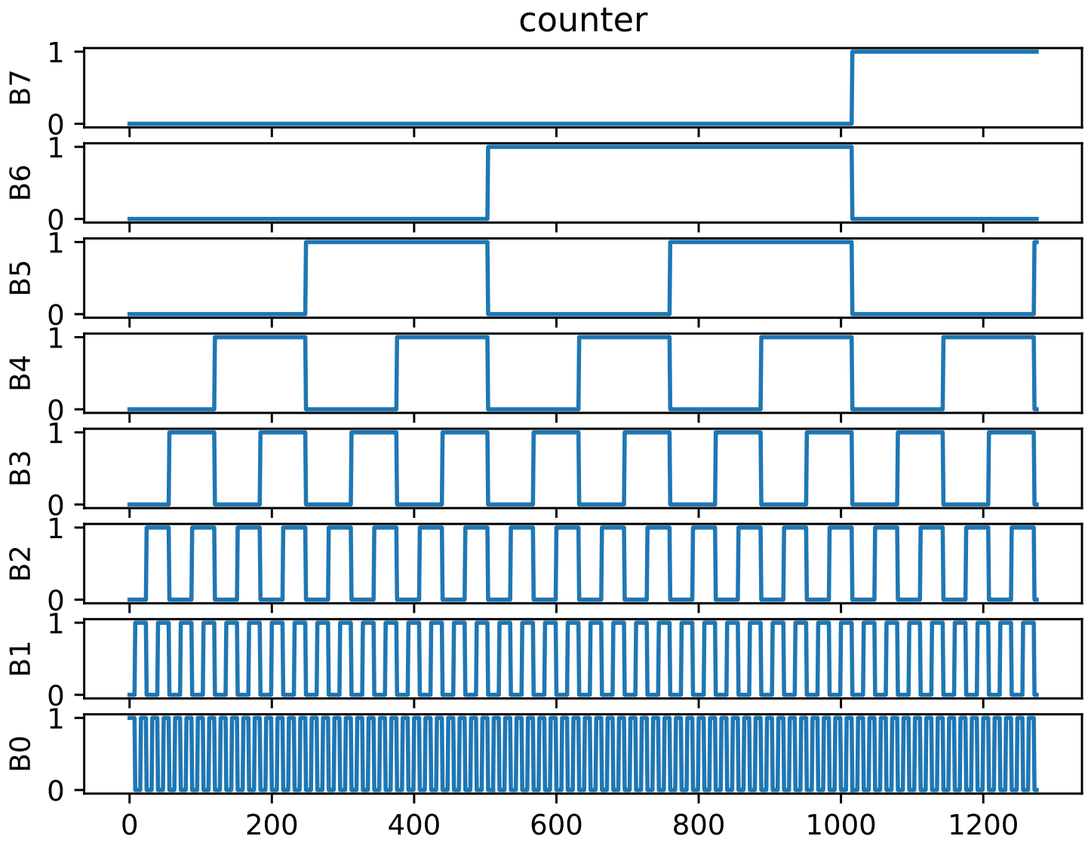
- Transistor corners
- Gray counter (but if you do, then it will be hard to get the ADC accurate)
- Voltage variation
- Temperature variation
- "nice to have features", like power optimalization, testability

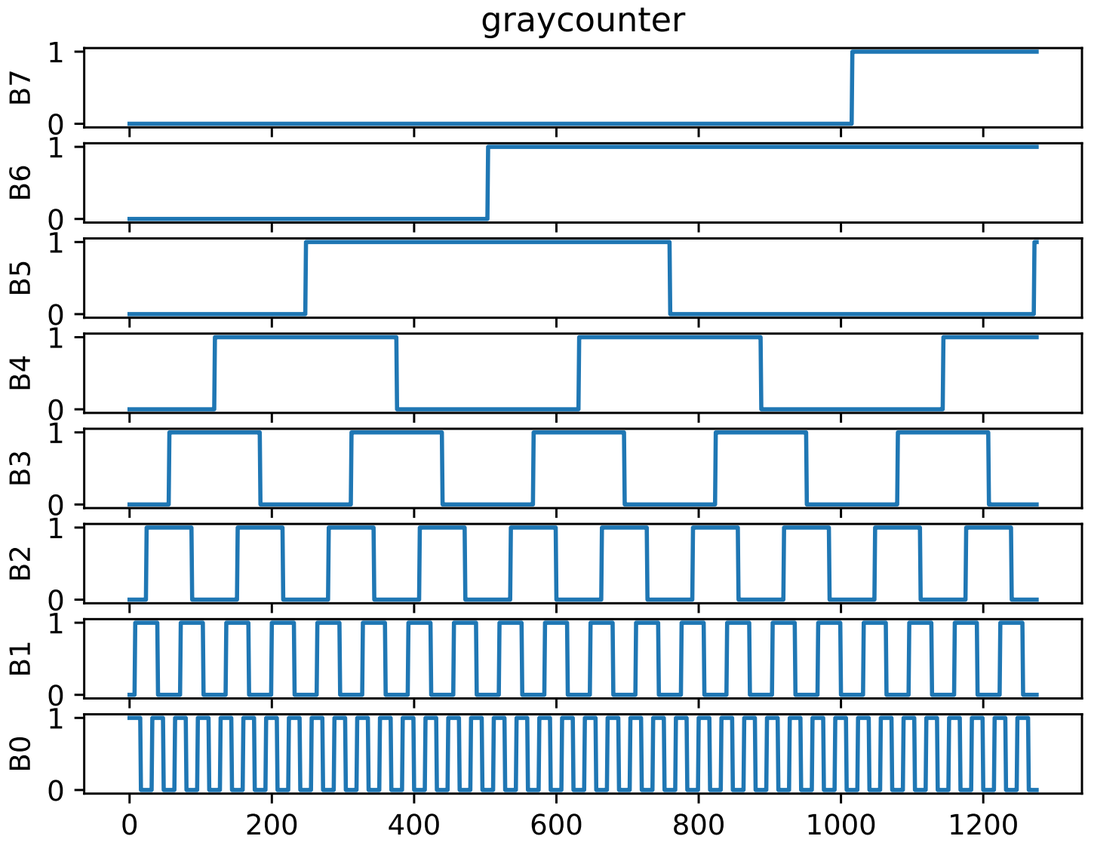
Gray codes

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

dicex/sim/verilog/graycounter.v

```
module graycounter(out, clk, reset);

   parameter WIDTH = 8;

   output [WIDTH-1 : 0] out;
   input                clk, reset;

   logic [WIDTH-1 : 0]  out;
   wire                 clk, reset;

   logic [WIDTH-1 : 0]  q;


   always @(posedge clk or posedge reset) begin
      if (reset)
        q <= 0;
      else begin
         q <= q + 1;
      end
      out <= {q[WIDTH-1], q[WIDTH-1:1] ^ q[WIDTH-2:0]};
   end

endmodule // graycounter
```

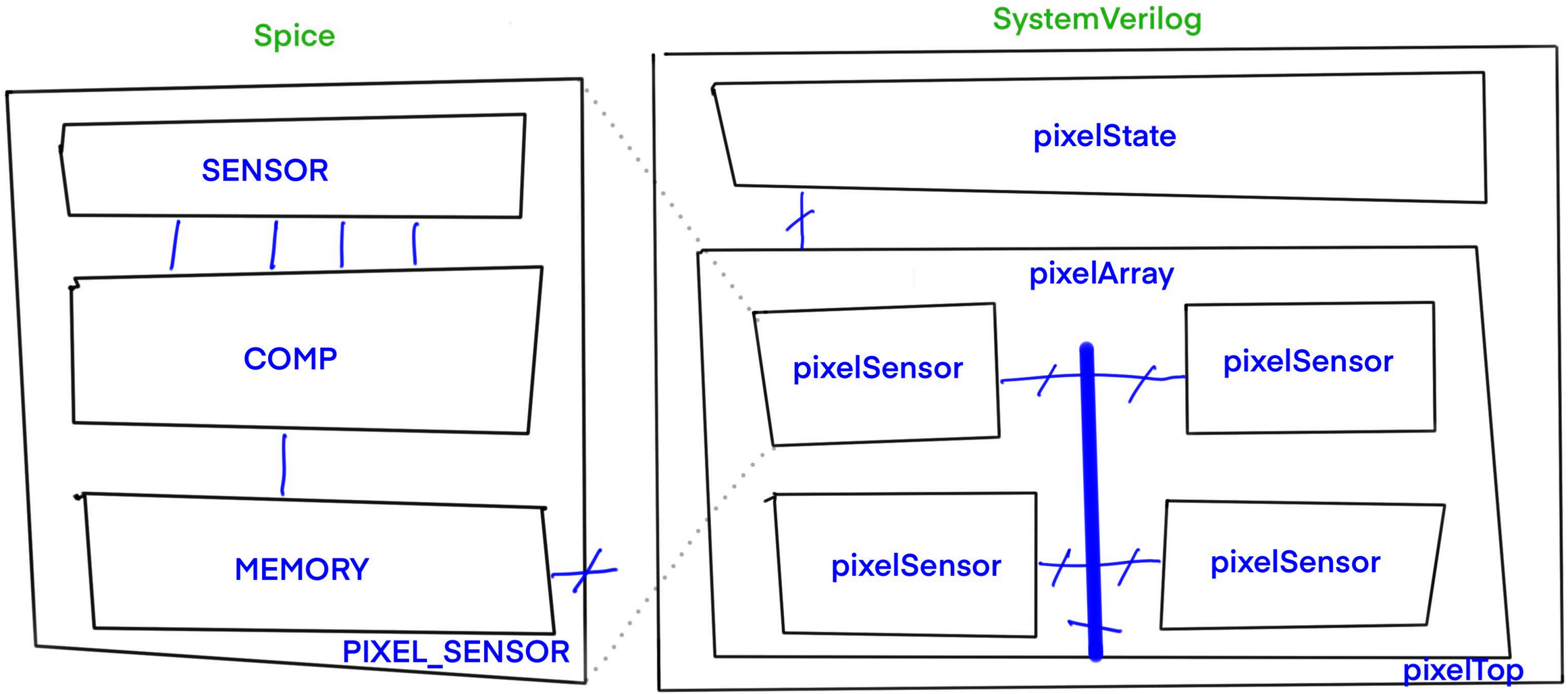

counter
B7
B6
B5
B4
B3
B2
B1
B0
1
0
0
200
400
600
800
1000
1200


graycounter
B7
B6
B5
B4
B3
B2
B1
B0
1
0
0
200
400
600
800
1000
1200

What you get

github.com/wulffern/dicex

```
project/
├── spice/
│   ├── Makefile                # See https://www.gnu.org/software/make/manual/html_node/Introduction.html
│   ├── pixelSensor.cir         # Almost empty circuit for pixelSensor
│   └── pixelSensor_tb.cir      # SPICE testbench for pixelSensor, emulates verilog
└── verilog/
    ├── Makefile
    ├── pixelSensor.fl          # Verilog file list
    ├── pixelSensor_tb.gtkw     # Save file for GTKWave
    ├── pixelSensor_tb.v        # Verilog testbench for pixelSensor
    └── pixelSensor.v           # Verilog model of analog pixelSensor circuit
```


Spice
SystemVerilog
SENSOR
COMP
MEMORY
PIXEL_SENSOR
pixelState
pixelArray
pixelSensor
pixelSensor
pixelSensor
pixelSensor
pixelTop

Analog Simulators

aimspice, ngspice, spectre, eldo, hspice

Time evolution (transient analysis) is a numerical analysis to differential equations for voltage and current. For example Newton's method

1. Take a small time step, iterate numerical analysis until error is low enough, if error is too large, chose shorter time step
2. Go to 1

Can simulate digital circuits, but very slowly

Digital Simulators

iverilog, questa, xcelium, vcs

Time evolution with time steps, and delta-time

1. Next time step (i.e clock cycle)
2. What signals change at this time step?
3. Compute new signal values, and schedule changes in future
4. Should any of the signals that changed now affect other signals now? If yes, then take delta time step
5. If delta timestep, then go to 2. If no more delta timestep, go to 1.

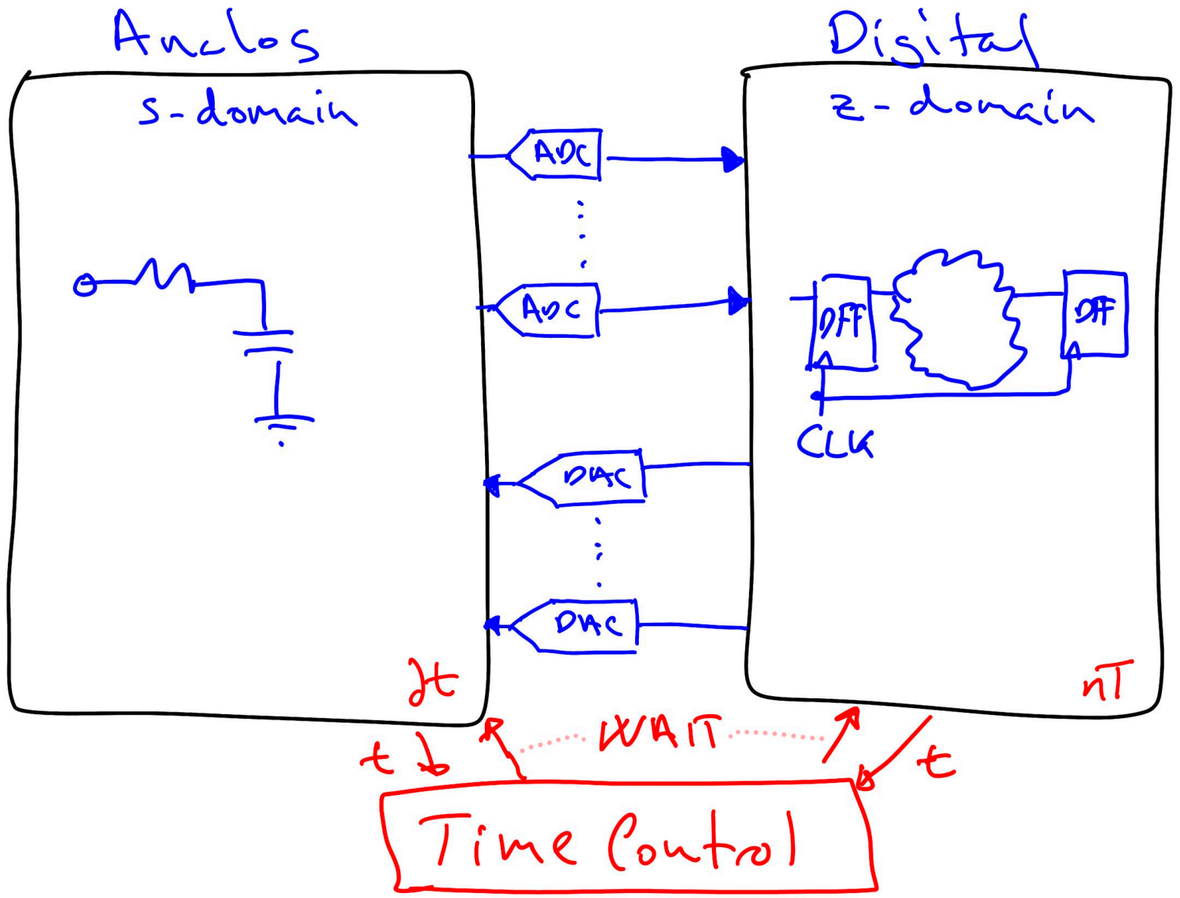
Cannot simulate analog differential equations!

Mixed-Signal Simulators

Control time in both analog and digital simulator

Provide analog-to-digital and digital-to-analog converters to "mirror" signals in the other simulator

Not sure there is an open source mixed-signal simulator



What you should do

Model of 2 x 2 pixel array in SystemVerilog

Make a SystemVerilg module (i.e `pixelArray.v`), that use `pixelSensor.v`

Figure out which signals need to be a bus, and what signals are common for the pixels

Make a testbench `pixelArray_tb.v` to check that the `pixelArray.v` compiles and do some rudementary tests to check that you've hooked things up correctly

State machine

Make a SystemVerilog module `pixelState.v` that can connect to `pixelArray.v`

Make a state machine to control reset, exposure, analog-to-digital conversion, and readout of the pixel array

Make a testbench `pixelState_tb.v` to test the state machine

Top verilog

Make a SystemVerilog module `pixelTop.v` that connects `pixelState.v` to `pixelArray.v`

Make a testbench `pixelTop_tb.v` to test the statemachine and readout of the 2 x 2 array

SPICE of pixel sensor

Copy `pixelSensor.cir` to another name

Copy `pixelSensor_tb.cir` to another name

Make the design from the paper (Fig. 4)
- Sensor (SENSOR)
- Comparator (COMP)
- Memory (MEMORY)

Add something (like `Rphoto` in `pixelSensor.tb`) to model the photocurrent.

Add a testbench for each subcircuit (COMP, SENSOR, MEMCELL, MEMORY)

Report 1/2

- **Introduction** = Why?
- **Theory** = How?
  - As little information as possible. Give references to sources. Assume that the reader has read the paper.
- **Implementation** = What?
  - Describe what you designed
  - State diagrams, with explanation
  - Circuit diagrams, with explanation
  - One sub-chapter per block

Report 2/2

- **Verification** = Are you sure it works?
  - Describe your testbenches, and how you verified your design
  - Describe key results
  - Describe known problems (incase something does not work)
- **Discussion and conclusion** = Why do you deserve a good grade?
- **Appendix**
  - SPICE netlist
  - SystemVerilog netlists
  - SystemVerilog testbenches

You need to start now to be able to complete the project with a good grade

Proposed plan

Week	Plan
34	Register group
35	Read and understand paper
36	Sketch what you want to do
37	Write theory chapter in report
38	Design & simulation
39	Design & simulation
40	Design & simulation
41	Design & simulation
42	Verification
43	Verification
44	Write report
45	Write report
46	**Deadline**

What you get

github.com/wulffern/dicex

```
project/
├── spice/
│   ├── Makefile                # See https://www.gnu.org/software/make/manual/html_node/Introduction.html
│   ├── pixelSensor.cir         # Almost empty circuit for pixelSensor
│   └── pixelSensor_tb.cir      # SPICE testbench for pixelSensor, emulates verilog
└── verilog/
    ├── Makefile
    ├── pixelSensor.fl          # Verilog file list
    ├── pixelSensor_tb.gtkw     # Save file for GTKWave
    ├── pixelSensor_tb.v        # Verilog testbench for pixelSensor
    └── pixelSensor.v           # Verilog model of analog pixelSensor circuit
```

Let's check what's inside the files

Thanks!