Design of a 16 666 frames/second CMOS Digital Pixel Sensor

Andreas S. Bakke, Øyvind M. Bjærum *2021-11-19*, v1.0.0

***Abstract*—In this paper we will design both analog and digital parts of a 2x2 CMOS digital pixel sensor with per pixel ADC and memory using $0.13\,\mu\mathrm{m}$ CMOS technology. In SPICE the sensor, comparator and memory inside each pixel sensor is implemented and tested. A model of the digital circuits is created in verilog for verification and modeling, a well as implementing a finite state machine for state control. The sensor includes parallel 8-bit A/D conversion and readout at 16 666 frames per second.**

I. Introduction

THE EVOLUTION of imaging technology has happened for several decades at this point, and the edges are pushed further and further as CMOS and the building blocks necessary to create high frame rate pixel sensors have evolved at a rapid pace. Today state of the art digital pixel sensors deliver sensors on a whole new level. A paper from 2020 by Liu, Chiao et al. titled "A 4.6μm, 512×512, Ultra-Low Power Stacked Digital Pixel Sensor with Triple Quantization and 127dB Dynamic Range" describe a digital pixel sensor capable using modern methods such as stacking CMOS image sensors, to meet ultra-low power, ultra-wide dynamic range requirements for always-on mobile computer vision applications. The fundamentals of this state of the art digital pixel sensor is still the same CMOS technology that was used 20 years ago. To showcase this evolution we therefore take inspiration in the soon to be 20 year old paper "A 10 000 Frames/s CMOS Digital Pixel Sensor" by Kleinfelder et al. [1] to design a 16 666 frames per second CMOS Digital Pixel Sensor with some fundamental changes.

In addition to implementing the described pixel sensor from the paper. We will also describe the design of a finite state machine to control the exposure, conversion, read and erase of the pixel sensor for ease of use improvements. This Moore machine connects to our 2x2 pixel array, to reduce the number of digital inputs in the system to just a clock and reset, in addition to our analog signals.

II. Theory

A. CMOS image sensor

We have two main types of electronic image sensors, charge-coupled devices (CCD) and active-pixel sensors (CMOS sensor). Both are based on MOS technology, but have some differences, mainly the use of MOS capacitors in CCD sensors, and use of MOSFET amplifiers in CMOS sensors. Both of these sensors accomplish the same task of capturing different amounts of light and converting it into electrical signals we can process as images. CCD sensors have some shortcomings in readout speeds, as Technical Advisor to Canons Professional Engineering and Solutions Division points out, "You can't get the data off the [CCD sensor] quickly enough, because there is a limit to the number of readout channels," [4]. More specifically this paper is about designing a digital pixel sensor (DPS). It is different from other CMOS image sensor in that it has a per-pixel ADC making readout of the pixel values resistant to analog noise. DPS sensors can also reach much higher frame rates because of the digital readout [1]. The drawback with DPS is that more transistors are required for each pixel. For CMOS technologies below $0.18\,\mu\mathrm{m}$. In this paper we will use $0.13\,\mu\mathrm{m}$.

B. SPICE

SPICE is the software used to simulate the analog circuits in this paper. It is used for computing transient solutions of the implemented circuits' node voltages. From these we can get plots of the voltages over time to see if the implemented circuits work as intended. The CMOS models used can be found here: http://ptm.asu.edu/modelcard/2006/130nm_bulk.pm.

C. Verilog

Another tool used to design and verify the CMOS digital pixel sensor in this paper is Verilog. Verilog is a hardware description language (HDL) which is most commonly used to model circuits, design and verify digital circuits on register transfer level and verify analog circuits as well as generic circuits.

D. FSM

There are multiple ways to control digital pixel sensors such as the one we will design and discuss in this paper. One of these controllers, which we will reference later on is a finite state machine (FSM). We understand a finite state machine to be a machine that can only can be in a specific state from a finite set of states. It transitions between states by inputs and by using a set of rules to determine what state to be in at all times. We have two main types of FSMs, Moore machines and Mealy machines. The main difference between these types of FSMs is the input-dependency of the machines output.

III. Implementation

The analog part of the implementation addresses everyting inside the pixel sensor. The digital part of the implementation takes care of the 2x2 array of pixel sensors.

A. Analog

The pixel sensor is made up of four parts. The sensor itself, a comparator, an 8-bit ADC and 8 bits of memory. The ADC is not implemented here. Block diagram of pixel sensor is shown in figure 1. The ADC and the readout of the pixel share the same data lines. VRAMP is an analog ramp which goes from a low value of $0.5\,\mathrm{V}$ to a high of $1.4\,\mathrm{V}$. ERASE, EXPOSE, CONVERT and READ are control signals for the different parts the analog circuit, these are implemented as ideal voltage sources. The period for one frame capture is $53\,\mu\mathrm{s}$. VBN1 is a bias line to calibrate the comparator, it is implemented in the appendix VIII-B. Lastly the supply voltages are VDD $= 1.5\,\mathrm{V}$ and VSS $= 0\,\mathrm{V}$.

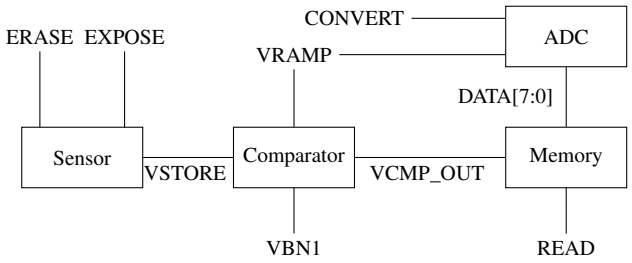

Fig. 1. Block diagram of single pixel.

1) Sensor: The circuit scheme of just the sensor is shown in figure 2. The photogate is implemented as the resistor $Rphoto$. There are no different photogates implemented in this design, meaning this pixel sensor cannot capture different colors. This implementation is simpler and different from the one in [1]. We expose the pixel when the EXPOSE signal goes high to make M1 active. The voltage from the photogate is then stored at the capacitor C_1. The transistor M2 is used as a switch to reset the voltage on C_1 to VRESET when ERASE goes high. Both transistors are working as switches, so they are as small as possible to make their rise and fall times short. VSTORE is then used as one of two inputs to the comparator. SPICE netlist of the sensor is in VIII-C.

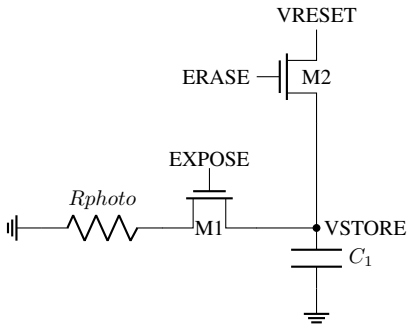

Fig. 2. Circuit scheme of sensor.

2) Comparator: The comparator, shown in figure 3, is meant to be a switch between the two input voltages VSTORE and VRAMP. The output of the comparator, VCMP_OUT, is high when VSTORE $>$ VRAMP and low when VRAMP $>$ VSTORE. To compare the two input voltages you need two transistors, M5 and M6 at the same operating point so they are the same size and fed by a current mirror. The PMOSes M7 and M8 make up the current mirror, so it is important these transistors are the same size. The bias line VBN1 calibrates the comparator gets a high or low input, this can typically be chosen in fabrication of the device. VBN1's value is set so that the pull down of M4 is stronger than the pull up of M9 when the drain voltage of M5 is bigger than the drain voltage of M6. M9 is a PMOS so we get a low VCMP_OUT when the drain voltage of M6 is lower, and vice versa. The last two transistors, M10 and M11 is just an inverter so the comparator provides the right write signal for the memory cells. The size of these two transistors are determined so the inverter is not skewed. SPICE netlist of the comparator is in VIII-D.

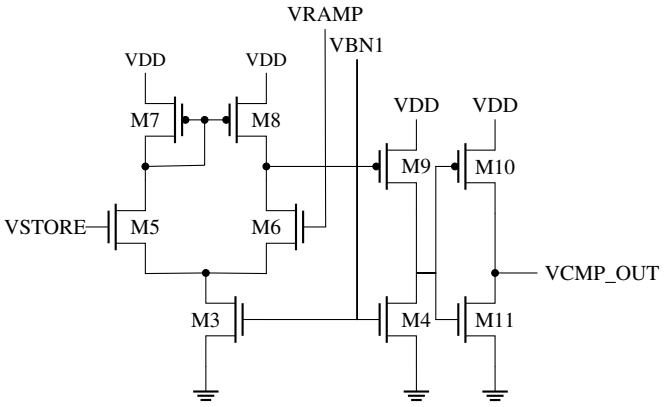

Fig. 3. Circuit scheme of comparator.

3) 8-bit memory: A single memory cell is shown in figure 4. M12 is a switch for putting the DATA output of the ADC on the gate of M14. M13 is a switch that connects the DATA line to the drain of M14. We want VCMP_OUT to be low when the memory cells should stop updating their value. The memory cells get the output from an 8-bit ADC which takes in the analog ramp voltage VRAMP and outputs a digital 8-bit value to the memory cells. and when this goes low we stop updating the voltage the gate of M14, because M12 is no longer active. Now we have stored the DATA line from when VCMP_OUT went low on the gate of M14. This transistor is wide to make the gate bigger to store this charge, also a $1\,\mathrm{pF}$ capacitor C_2 is added to make the memory cell able to store the charge for longer. When READ goes high M13 just connects the DATA line to the drain of M14. If this stored gate voltage is high then M14 is active and pulls the DATA line low. If the M14 had a low gate voltage the DATA line would be read as high. To shortly summarize, the ADC output value is sampled once to the memory. SPICE netlist for single memory cell is found in VIII-E, the netlist for all 8 bits of memory is found in VIII-F.

4) Pixel sensor: The pixel sensor is implemented as the sensor connected to the comparator, which is again connected to the memory like in the block diagram in figure 1. SPICE netlist for the pixel sensor is found in VIII-G.

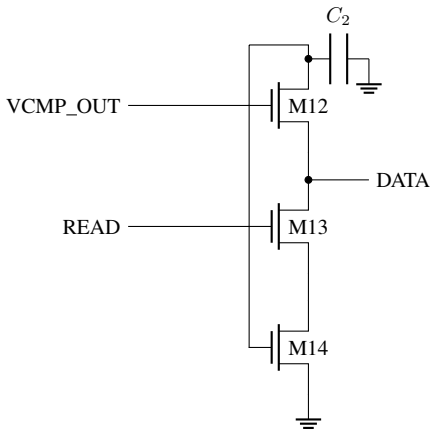

Fig. 4. Circuit scheme of a single memory cell.

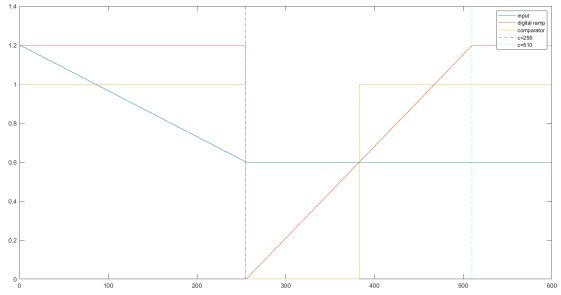

Fig. 5. Pixel sensor signals during one cycle of expose and convert with $dv_{pixel} = 0.5$. In blue, our simulated analog input, in red our simulated analog ramp, in yellow the comparator output, and the blue and cyan dashed lines indicate clock cycle number 255 and 510 respectively.

B. Digital

The digital design is divided into four modules. The pixel sensor, a 2x2 array of pixel sensors, a finite state machine for pixel array control and a top module to connect the fsm and array.

1) Pixel Sensor: The design of the pixel sensor is identical the Pixel Sensor modeled by Wulff [3]. The pixel sensor is modelled digitally by dividing it into the same four parts as in the analog design. A sensor, a comparator, an ADC and a memory latch. The sensor behaviour during exposure is modelled by setting the input to v_{erase}, then lowering the input by $\frac{v_{erase}}{c_{exposure}} \cdot dv_{pixel}$ every clock cycle. Here $c_{exposure}$ is the number of clock cycles during exposure and dv_{pixel} is the photocurrent (between 0 and 1). This gives our comparator an input voltage between 0 and v_{erase} dependant on dv_{pixel}.

The comparator and ADC during conversion is modelled similarly to the sensor, by replacing the analog ramp signal with a digital ramp that increases by a factor of $\frac{v_{erase}}{c_{exposure}}$ every clock cycle until its value is larger than the input received from the comparator. At this point the comparator output goes from 0 to 1, as opposed to going from 1 to 0 as discussed in the paper. While the comparator signal is low, we get a counter input, which latches into our memory when the comparator output goes high. The memory is simply a new register.

We can observe this from figure 5, where the signals in a pixel sensor with $dv_{pixel} = 0.55$ is plotted, for one cycle of expose and convert. The expose and convert cycle last 255 clock cycles each. As we would expect, the input reaches 0.6 after 255 clock cycles, and the comparator goes high after an additional 128 clock cycles, see section IV for verification.

2) Pixel Array: In the Pixel Array, we create an array consisting of 4 pixel sensors in a 2x2 grid, we also implement a counter, and control the data input/outputs of the pixel sensors. We implement a binary counter, as opposed to a Gray Counter, that we provide as an input to our pixel sensors. The counter has a maximum value equal to the number of clock cycles during exposure. To be able to provide the counter as an input, and later read from memory using the same databus, we need to control the read and write. This is simply done by giving the input a "High Impedance State"; Z. This allows us to drive the bus as an output from the pixel sensor, by assigning the bus the latched value from within the sensor when reading. Similarly we assign the output Z within the sensor when driving the counter as an input. The verilog model of the pixelArray can be found in appendix VIII-H.

The Pixel Array takes in two read signals as inputs, and two 8 bit databusses as outputs as we want to limit the number of databusses used for reading the pixel sensors. Each column of pixel sensors in the array share one of the 8 bit databusses. While the read signals are shared row-wise to allow simultaneous reading of two pixel sensors. This design corresponds with the 2x2 example provided at page 3 in [1].

3) Pixel State: To control the different states of the pixel sensor, we have chosen to design a finite state machine (FSM), more specifically a Moore machine, hereby refereed to as Pixel State. The verilog model of Pixel State can be found in appendix VIII-I. We have chosen to design a Moore machine to reduce the amount of signals in our design, and to have outputs solely depending on which state the FSM is in. Pixel State has five states, ERASE, EXPOSE, CONVERT, READ0, and READ1 and provides these states as output wires with values 1 or 0. Figure 6 shows a state diagram of the FSM. When changing state, we load the next counter value into a register, and also move the current value into our counter. This value depends on which state is following. The counter value decrements for each positive clock edge and when the counter reaches 0, the state changes unless a positive reset input R is present. The following negative clock edge the Pixel State outputs are updated to correspond to the current state. If the Reset input signal ever equal 1, we go back to our base state, ERASE. In this implementation the state machine is implemented with the counter values from table I. This gives us 8 bit resolution.

c_{erase}	5
c_{expose}	255
$c_{convert}$	255
c_{read}	5

TABLE I
COUNTER VALUES USED IN THIS IMPLEMENTATION OF PIXEL STATE.

4) Pixel Top: Finally we implemented a module to connect the FSM to the 2x2 pixel array, hereby refereed to as Pixel Top,

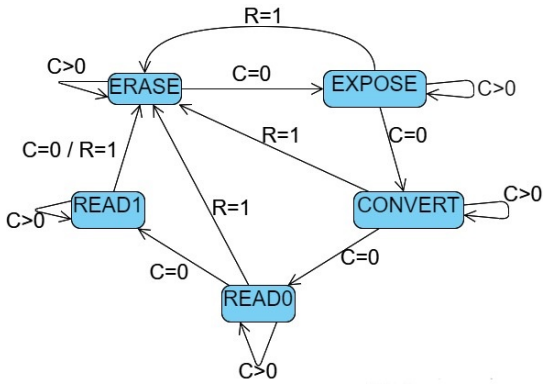

Fig. 6. State diagram of the finite state machine, Pixel State with five states. The initial state is ERASE, and whenever the internal counter c hits zero, the state changes in the order EXPOSE-CONVERT-READ0-READ1-ERASE-EXPOSE unless a positive reset R is present.

which can be found in appendix VIII-J. The block diagram for Pixel Top can be seen in figure 7. In addition to connecting the FSM to the array, we also control the final output, and drive the aforementioned analog Ramp and Bias signals. As mentioned the ramp and bias follow the clock signal whenever our state machine outputs convert = 1 and expose = 1 respectively.

Finally we control the readout from the two data busses onto our final 32 bit databus/storage. Since the first row from the top is read during the state READ0, we also drive the two outputs from the array into the first 16 bits, 0-15, during this state. In the following READ1 state we can drive the same outputs to the next 16 bits, 16-31.

IV. Result

A. Analog

All testbenches used to produce these results can be found at VIII-A.

1) Sensor: The output of the sensor is pretty simple. We see that the charge from the photogate is moved over to VSTORE when EXPOSE is high because the voltage is the same on VSTORE as on the photogate. When EXPOSE goes low the VSTORE voltage trickles through the capacitor C_1, so it does not stay completely constant. In the simulation shown in figure 8 VSTORE drops from $0.91\,\mathrm{V}$ to $0.88\,\mathrm{V}$. When RESET goes high we see an abrupt change in VSTORE where it reaches $1.1\,\mathrm{V}$. Here the sensor is tested with no load on VSTORE.

2) Comparator: The comparator is tested with a VSTORE input voltage modeled with an ideal voltage source. In figure 9

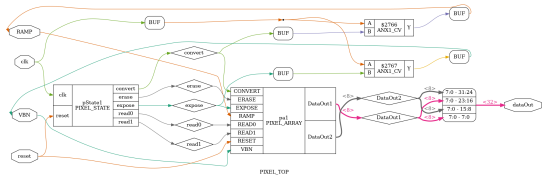

Fig. 7. Block diagram of the module Pixel Top, connecting the finite state machine to our pixel Array. As well as driving the analog ramp and bias signals. The behaviour of blocks pState1 and ps1 are described in subsections III-B3 and III-B2 respectively.

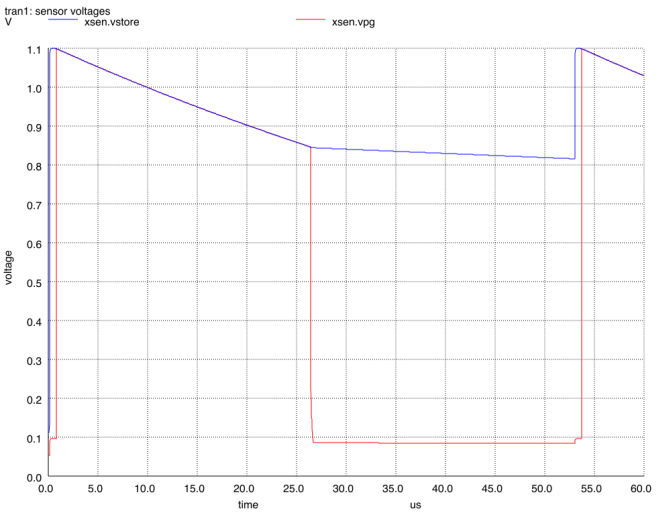

Fig. 8. Plot of sensor voltages.

VCMP_OUT goes from high to low as VRAMP $>$ VSTORE. This takes $1.1\,\mu\mathrm{s}$. No load is put on VCMP_OUT in this test.

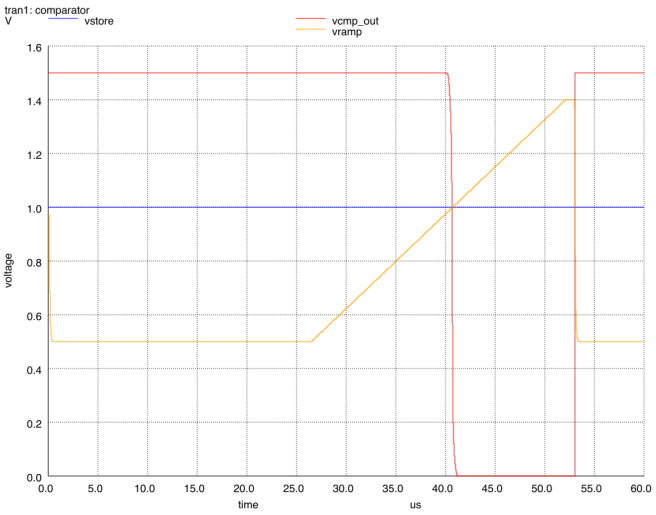

Fig. 9. Plot of comparator with the two inputs VRAMP and VSTORE.

3) Memory cell: A single memory cell is tested with two different VCMP_OUT signals, one that should write a low value to the memory cell and one that should write a high value to the memory cell. The difference between these two is when VCMP_OUT goes from high to low. For both tests there is no load on DATA when reading from the memory cell. In figure 10 we see that the low value of the gate voltage, VG, is kept even though DATA goes high because READ is low. The low value of VG, $0.05\,\mathrm{V}$, is then put on DATA after reading. In figure 11 we see a high VG when VCMP_OUT goes low when DATA is high. VG is then stored in C_2 and the gate of M14 so it decays from a high of $1.7\,\mathrm{V}$ to $1.2\,\mathrm{V}$ after the read. A voltage of $1.2\,\mathrm{V}$ is also put on DATA after reading.

4) 8-bit memory: For testing 8 bits of memory the 8-bit digital value of VRAMP is compared to the values stored the 8 bits of memory. To make the test simple VCMP_OUT is modeled as a ideal voltage source. For testing multiple values

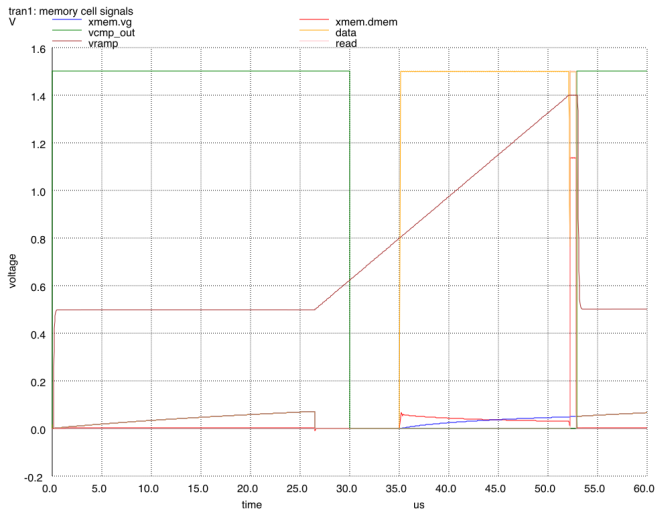

Fig. 10. Plot of writing a digital 0 to a memory cell.

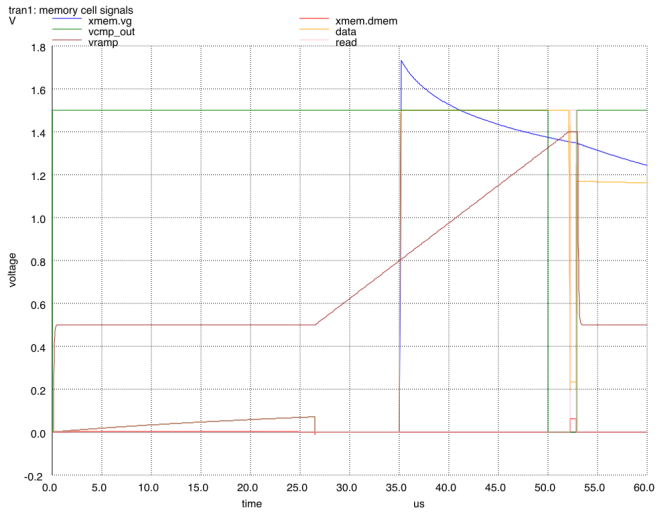

Fig. 11. Plot of writing a digital 1 to a memory cell.

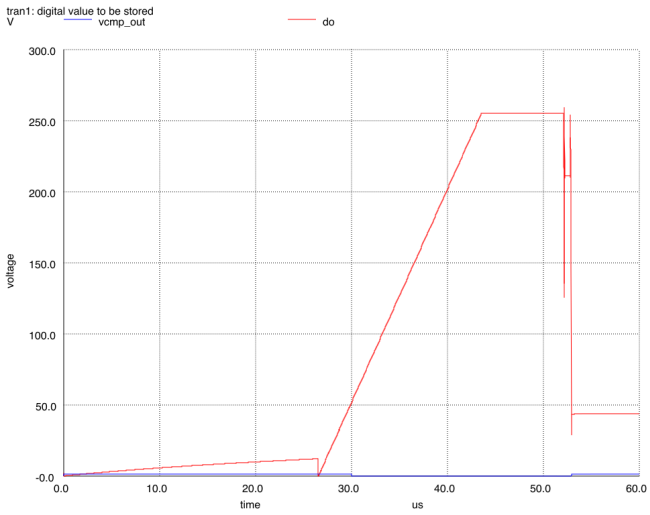

Fig. 12. Plot of 8-bit digital VRAMP values and VCMP_OUT.

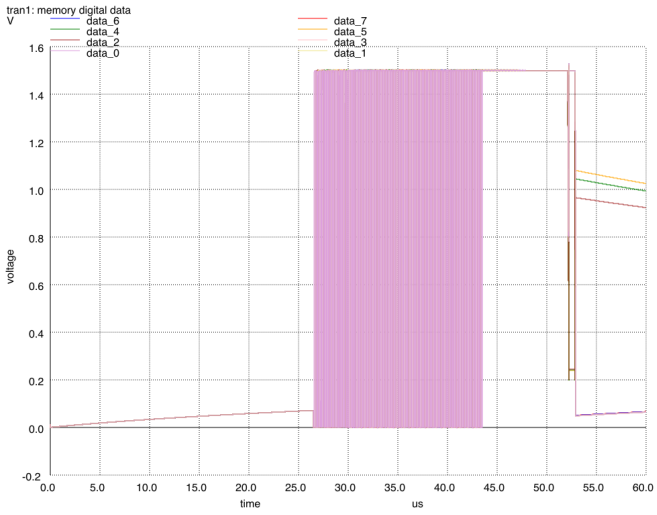

Fig. 13. Plot of memory values with VCMP_OUT going low after 30 µs.

of VRAMP, VCMP_OUT will go low after 30 µs, 40 µs and 50 µs. With VCMP_OUT going low after 30 µs we get the digital value of 52 shown in figure 12. The corresponding memory values are shown in figure 13. DATA_5, DATA_4 and DATA_2 is high, this is $2^5 + 2^4 + 2^2 = 52$.

With VCMP_OUT going low after 40 µs the digital value of VRAMP is 202, as shown in figure 14. The memory in figure 15 shows DATA_7, DATA_6, DATA_3 and DATA_1 is high, corresponding to a digital value of $2^7 + 2^6 + 2^3 + 2^1 = 202$.

With VCMP_OUT going low after 40 µs the digital value of VRAMP is maxed to 255, shown in figure 16. In the memory all DATA lines are high, shown in figure 17. This corresponds to a digital value of 255.

5) Pixel sensor: To test that all the subcircuits work when connected together the comparator switching from high to low and value stored in memory is checked. The comparator and control signals are shown in figure 18. The memory values for this test is shown in figure 19. Here the values of the DATA_1, DATA_2 and DATA_3 are read somewhere between a logical high and low. The other bits are either low or high.

B. Digital design

As mentioned we implemented a testbench to verify our modules work as intended. This testbench may be found in appendix VIII-A. We use our analog simulations as a basis for selecting the clock period in our testbench. As we can observe in figure 8, one cycle of our pixel sensor is simulated to last approximately 60μs, which results in a clock period of 114ns as our state machine uses a total of 525 clock cycles to erase, expose, convert and read from all four pixelsensors. This gives our CMOS digital pixel sensor a theoretical 16 666 frames/ second.

We have two main interests in evaluating the results of our digital design. Whether or not our output values are correct, and try different cases of inputs where the system may fail. We believe the system has the highest chance of failing when we input a reset signal during changes in states, with varying

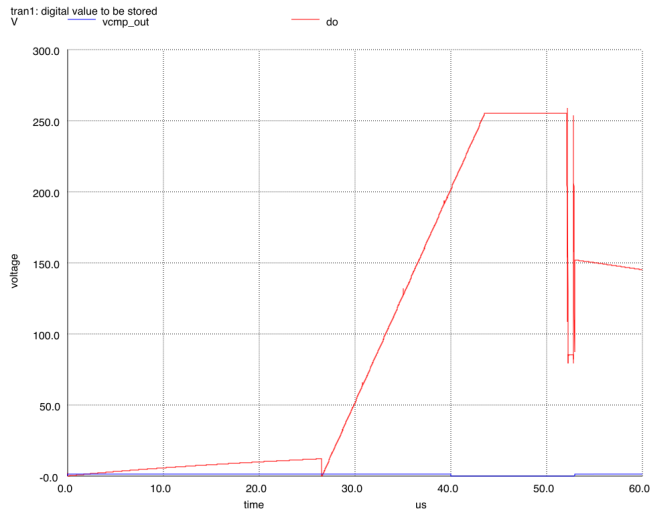

Fig. 14. Plot of 8-bit digital VRAMP values and VCMP_OUT.

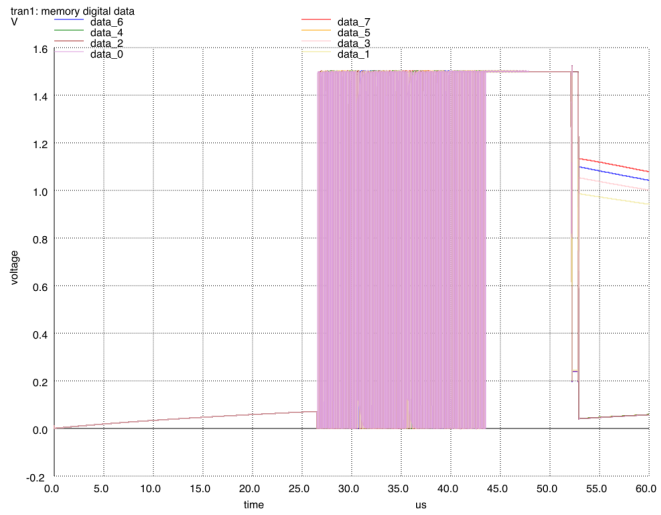

Fig. 15. Plot of memory values with VCMP_OUT going low after 40 µs.

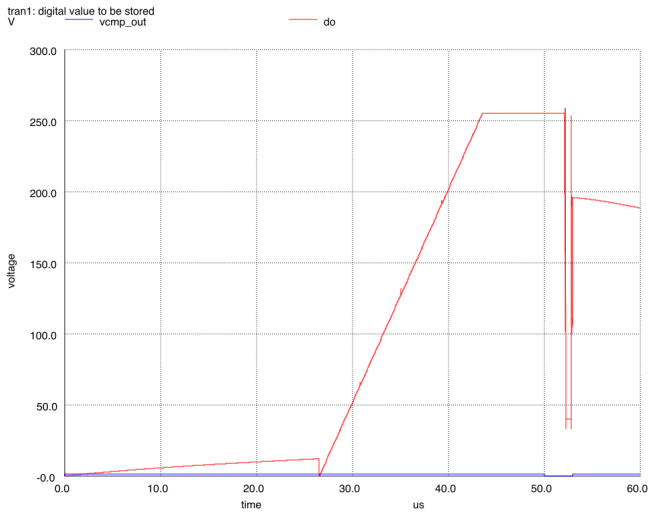

Fig. 16. Plot of 8-bit digital VRAMP values and VCMP_OUT.

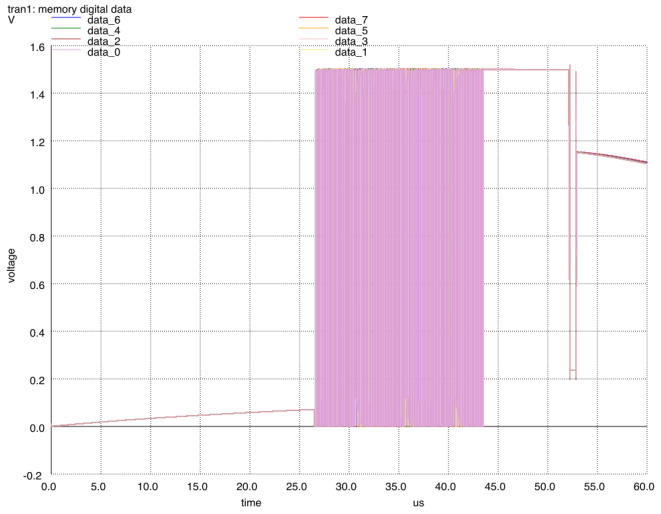

Fig. 17. Plot of memory values with VCMP_OUT going low after 50 µs.

intervals and lengths. We therefore do some tests to rule out any system failures.

1) Output Values: Figure 20 shows the 32 bit databus from our system, the 8 bit pixel array busses, as well as the data in and out of the individual pixels. The four pixel sensors were assigned four different photocurrents dv_{pixel} as seen in table II along with the captured values for each sensor. Notice that the databus does not change value when we are finished reading, but rather keeps the latched values from the sensors until the value changes and is driven to the bus during the following read states.

Pixel	$\mathbf{dv_{pixel}}$	**Output (HEX)**
1	0.48	85
2	0.5	80
3	0.52	7B
4	0.54	76

TABLE II
OUTPUTS FROM OUR FOUR PIXEL SENSORS WITH VARYING dv_{pixel}.

2) Resets: We test the reset by driving the signal at a rising clock edge, falling clock edge, in the middle of a clock period or at the exact same time that the state machine changes state as we can see in figure 21, 22, 23 and 24 respectively. When resetting during a falling clock edge or between two edges, we observe that the internal pixel memory is 01. From figure 25 we observe the results of driving a high reset input for a long time. The output value from our bus is 00000000 and the state is set to ERASE in all cases.

V. DISCUSSION

A. Analog

The sensor is not easy to model correctly as there is no way to simulate a stream of photons in SPICE. A more sophisticated implementation of the photo gate could have made results from testing the entire pixel sensor more informative, in this implementation the VSTORE voltage barely changed throughout the expose time. Also the capacitor that stores the VSTORE voltage is implemented as a transistor in [1], but we

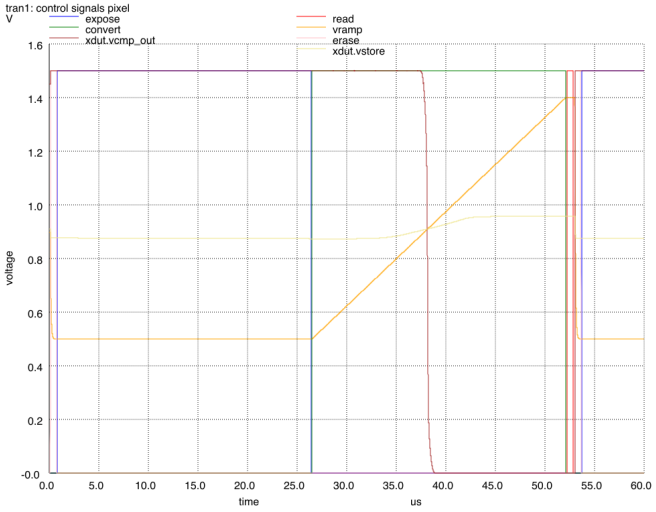

Fig. 18. Plot of control signals and inputs and outputs of the comparator with all subcircuits connected.

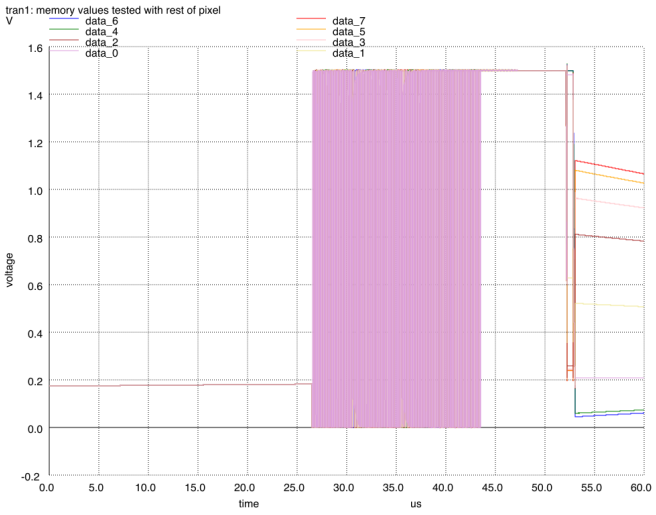

Fig. 19. Memory data when tested with rest of pixel.

were not able to get that working with the models used in this paper.

The comparator works as intended, but there were many things we did not test. Output impedance, output current would be valuable information to further optimise the implementation. Slow rise and fall times might be the reason why some of the memory bits stored values that were not clear digital 1 or 0. What rise time required for this not to happen is unknown.

The memory implementation is simple, but functioning. All

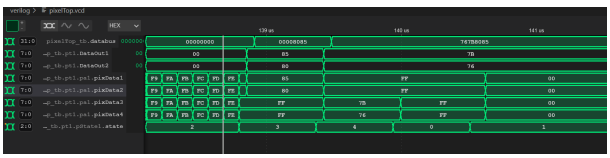

Fig. 20. Value change dump visualised. From top to bottom we observe the output of the Pixel Top module, the two column pixel Array databusses, the individual data readout of the pixel sensors and the FSM state.

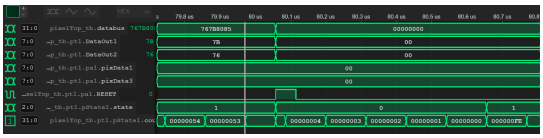

Fig. 21. Results of applying a reset pulse during a rising clock edge.

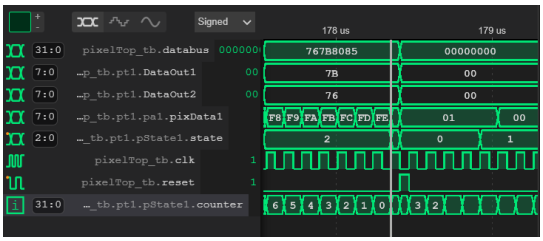

Fig. 22. Results of applying a reset pulse during a falling clock edge.

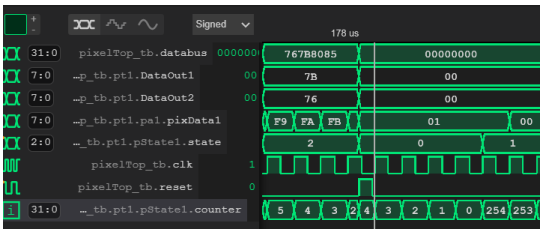

Fig. 23. Results of applying a reset pulse between two edges.

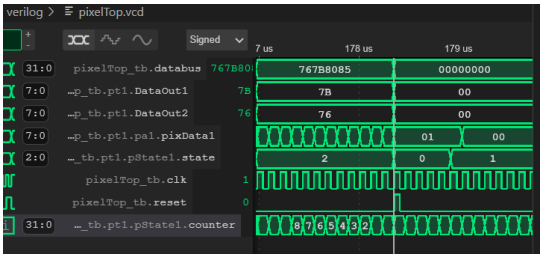

Fig. 24. Results of applying a reset pulse during a rising clock edge, and while the FSM is switching states.

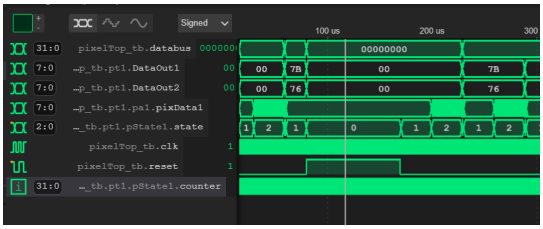

Fig. 25. Results of applying a wide reset pulse.

3 tests show the memory storing the correct digital values. The digital value that was put on to the DATA lines after reading was lower than the value written to the memory. This could be because the voltage stored in memory was slowly trickling through C_2 or the simple design of the DAC used in the testbench.

B. Digital design

When testing our reset pulses we notice that it works exactly as expected when the pulse is placed on a rising clock edge. However the internal pixel memory is wrongfully read as 01 during just after the reset when driving the reset at a falling clock edge, or between the two clock edges. This does however not impact the rest of the system, nor the output after the cycle is finished. We still end up with our expected 767B8085 output as the counter resets at the beginning of our exposure. From the wide reset pulse we observe that the FSM does not transition to the next state until the reset goes back down to 0 as expected. The duration of resets therefore have a impact on how many frames we can expect per second with regular resets.

We need to determine whether or not our outputs are accurate and displays the correct value. We do this mathematically by evaluating the second pixel in the first row, PixelSensor2 (ps2). We have defined ps2 to have a photocurrent of 0.5, and v_erase = 1.2V. After one expose state of 255 clock cycles, we get

$$input = v_{erase} - 0.5 \cdot \frac{v_{erase}}{255} \cdot 255 = 0.6V \tag{1}$$

We have the model

$$ramp = ramp_0 + lsb \cdot x \tag{2}$$

where

$$lsb = \frac{v_{erase}}{255} \tag{3}$$

ramp$_0$ is the initial ramp voltage and x is the counter value. As we want to find which latched value x when the ramp exceeds 0.6V we see that:

$$x > \frac{0.6 \cdot 255}{1.2} = 127.5 \tag{4}$$

as our inital ramp voltage is 0. Since we do not count decimals, the theoretical output from PixelSensor2 should be 128 as confirmed by figure 20 since HEX80 = 128.

This can be repeated for all the sensors and the the output signal in the digital verification always ends up at our expected 767B8085 as we can see from table III. We may also observe that the row-wise readout works as intended. During READ0 DataOut1 stores the correct output from PixelSensor1, while DataOut2 stores the correct output from PixelSensor2 immediately after the first falling clock edge. Equivalently for READ1, we store the correct value in our two 8 bit databusses as well as our final 32bit databus.

Pixel	$\mathbf{dv_{pixel}}$	Expected output (HEX)	Actual output (HEX)
1	0.48	85	85
2	0.5	80	80
3	0.52	7B	7B
4	0.54	76	76

TABLE III
EXPECTED AND ACTUAL OUTPUTS FROM OUR FOUR PIXELSENSORS WITH VARYING dv_{pixel}.

VI. FUTURE WORK

An implementation of the ADC and bias line would be a natural next step. This part is important for ensuring the correct values are written to memory. Changing the digital value from binary to gray code could be a big improvement, because with gray code only a single bit changes for each value. This means the data lines for the least significant bits would be more stable. The current implementation had a problem with this, writing values to memory that could be between the thresholds for a digital 1 and 0. An improvement to the current design of the sensor would be more realistic and different photogates to emulate different colors. Making sure the analog circuits can handle being in a bigger array would also be useful. Implementing a grid of read and write for the memory, and testing interference between the pixel sensors is key to see if the design is scaleable for bigger image sensors.

For future work more controls could be implemented to more accurately simulate modern CMOS pixel sensors. This includes, but is not limited to, variable exposure control, and obviously higher resolution of pixel sensors. Expanding the pixel array only requires additional read signals for each row, and additional busses for each column. The state machine could simply be modified by a new state for each read signal.

VII. CONCLUSION

We have successfully designed a 2x2 CMOS digital pixel sensor capable of capturing 16 666 frames per second. All subcircuits implemented in the analog design are working as intended both when tested individually and together. The individual pixel sensors show limited value-errors during digital verification and design, but these do not propagate to our sensor output, and we may consider the design to have accurate data representation and 8-bit black and white resolution.

VIII. APPENDIX

A. GitFront

Testbenches and all mentioned verilog and spice files with comments may be found at https://gitfront.io/r/user-2142794/f6b313a790913c78954d6b3609f49e780854d5e8/ICProsjekt/

B. Bias line implementation

```
IPB1 0 VBN1 dc 1u
XMNB0 VBN1 VBN1 VSS VSS NCHCM2
```

C. Sensor implementation

```
.SUBCKT SENSOR RESET STORE ERASE EXPOSE VDD VSS
*Capacitor to model gate-source capacitance
C1 VSTORE VSS 100f

M1 RESET ERASE VSTORE VSS nmos W=0.2u
L=0.13u

M2 VPG EXPOSE VSTORE VSS nmos W=0.2u
L=0.13u

* Model photocurrent
Rphoto VPG VSS 1G
.ENDS
```

D. Comparator implementation

```
.param p_wp = 2.56
.SUBCKT COMP VCMP_OUT VSTORE VRAMP VBN1 VDD VSS
* Current mirror
M7 VMIRROR VMIRROR VDD VDD pmos
W = {0.65u*p_wp} L=0.13u
M8 VDM2 VMIRROR VDD VDD pmos
W = {0.65u*p_wp} L=0.13u

* Comparing VRAMP to VSTORE
M5 VMIRROR VSTORE VDM1 VSS nmos W=0.65u L=0.13u
M6 VDM2 VRAMP VDM1 VSS nmos W=0.65u L=0.13u

M3 VDM1 VBN1 VSS VSS nmos W=0.65u L=0.13u
M4 VINV VBN1 VSS VSS nmos W=0.65u L=0.13u
M9 VINV VDM2 VDD VDD pmos
W = {0.65u*p_wp} L=0.13u

*inverter
M10 VCMP_OUT VINV VDD VDD pmos W = {0.65u*p_wp} L=0.13u
M11 VCMP_OUT VINV VSS VSS nmos W=0.65u L=0.13u
.ENDS
```

E. Memory cell implementation

```
.SUBCKT MEMCELL READ WRITE DATA VSS
M12 VG WRITE DATA VSS nmos  W=0.2u
L=0.13u
M13 DATA READ DMEM VSS nmos  W=0.4u
L=0.13u
M14 DMEM VG VSS VSS nmos  W=1u  L=0.13u
C2 VG VSS 1p
.ENDS
```

F. 8-bit memory array implementation

```
.SUBCKT MEMORY READ VCMP_OUT
+ DATA_7 DATA_6 DATA_5 DATA_4 DATA_3
+ DATA_2 DATA_1 DATA_0 VSS
XM1 READ VCMP_OUT DATA_0 VSS MEMCELL
XM2 READ VCMP_OUT DATA_1 VSS MEMCELL
XM3 READ VCMP_OUT DATA_2 VSS MEMCELL
XM4 READ VCMP_OUT DATA_3 VSS MEMCELL
XM5 READ VCMP_OUT DATA_4 VSS MEMCELL
XM6 READ VCMP_OUT DATA_5 VSS MEMCELL
XM7 READ VCMP_OUT DATA_6 VSS MEMCELL
XM8 READ VCMP_OUT DATA_7 VSS MEMCELL

.ENDS
```

G. Pixel sensor implementation

```
.SUBCKT PIXEL_SENSOR VBN1 VRAMP VRESET ERASE
+ EXPOSE READ DATA_7 DATA_6 DATA_5 DATA_4
+ DATA_3 DATA_2 DATA_1 DATA_0 VDD VSS

XS1 VRESET VSTORE ERASE EXPOSE VDD VSS SENSOR

XC1 VCMP_OUT VSTORE VRAMP VBN1 VDD VSS COMP

XM1 READ VCMP_OUT DATA_7 DATA_6 DATA_5 DATA_4
+ DATA_3 DATA_2 DATA_1 DATA_0 VSS MEMORY
.ENDS
```

H. pixelArray.v

```
`timescale 1 ns / 1 ps

module PIXEL_ARRAY(
    input logic VBN,
    input logic RAMP,
    input logic RESET,
    input logic ERASE,
    input logic EXPOSE,
    input logic READ0,
    input logic READ1,
    input logic CONVERT,
    output  reg [7:0] DataOut1,
    output  reg [7:0] DataOut2
);

    wire [7:0] pixData1;
    wire [7:0] pixData2;
    wire [7:0] pixData3;
    wire [7:0] pixData4;

    PIXEL_SENSOR #(.dv_pixel(0.48))
     ↪  ps1(VBN, RAMP, ERASE, EXPOSE,
     ↪  READ0, pixData1);
    PIXEL_SENSOR #(.dv_pixel(0.5))
     ↪  ps2(VBN, RAMP, ERASE, EXPOSE,
     ↪  READ0, pixData2);
    PIXEL_SENSOR #(.dv_pixel(0.52))
     ↪  ps3(VBN, RAMP, ERASE, EXPOSE,
     ↪  READ1, pixData3);
    PIXEL_SENSOR #(.dv_pixel(0.54))
     ↪  ps4(VBN, RAMP, ERASE, EXPOSE,
     ↪  READ1, pixData4);


    //-------------------------
    //  ADC/DAC
    //-------------------------
    logic[7:0] data;
    always_ff @(posedge RAMP or posedge
     ↪  VBN or posedge RESET) begin
        if(RESET) begin
            data=0;
        end
        if(CONVERT) begin
            data= data + 1;
        end
        else begin
            data= 0;
        end
    end

    output && 8'bZ = output
    assign pixData1 = READ0 ? 8'bZ: data;
    assign pixData2 = READ0 ? 8'bZ: data;
    assign pixData3 = READ1 ? 8'bZ: data;
    assign pixData4 = READ1 ? 8'bZ: data;

    always_ff @(posedge READ0 or posedge
     ↪  READ1 or posedge RESET) begin
       if(RESET) begin
            DataOut1=0;
            DataOut2=0;
       end
       else begin
          if(READ0)begin
            DataOut1 <= pixData1;
            DataOut2 <= pixData2;
          end
          else if (READ1) begin
            DataOut1 <= pixData3;
            DataOut2 <= pixData4;

          end
       end
    end
endmodule
```

I. pixelState.v

```
`timescale 1 ns / 1 ps

module PIXEL_STATE (
    input   logic   clk,
    input   logic   reset,
    output  logic   erase,
    output  logic   read0,
    output  logic   read1,
    output  logic   expose,
    output  logic   convert
);
    parameter integer c_erase = 5;
    parameter integer c_expose = 255;
    parameter integer c_convert = 255;
    parameter integer c_read = 5;

//--------------------------------------
// State Machine
//--------------------------------------
    parameter   ERASE=0, EXPOSE=1,
    ↪  CONVERT=2, READ0=3, READ1 =4;
    logic [2:0] state;
    integer counter=c_erase;
    integer next_counter=c_expose;

    always_ff @(negedge clk) begin
        case(state)
            ERASE: begin
                erase   <= 1;
                expose  <= 0;
                convert <= 0;
                read0   <= 0;
                read1   <= 0;
            end
            EXPOSE: begin
                erase   <= 0;
                expose  <= 1;
                convert <= 0;
                read0   <= 0;
                read1   <= 0;
            end
            CONVERT: begin
                erase   <= 0;
                expose  <= 0;
                convert <= 1;
                read0   <= 0;
                read1   <= 0;
            end
            READ0: begin
                erase   <= 0;
                expose  <= 0;
                convert <= 0;
                read0 <= 1;
                read1 <= 0;
            end
            READ1: begin
                erase   <= 0;
                expose  <= 0;
                convert <= 0;
                read0 <= 0;
                read1 <= 1;
            end
        endcase
    end


    always_ff @(posedge clk or posedge
    ↪  reset) begin
        if(reset)begin
            state = ERASE;
            counter = c_erase;
            next_counter=c_expose;
            convert=0;
        end
        else begin
            if (!counter) begin
                case (state)//Kontrollere
                ↪  hva som er neste
                ↪  Counter, slik at dette
                ↪  endres neste gang.
                    ERASE: begin
                        state       =
                        ↪  EXPOSE;
                        counter     =
                        ↪  next_counter;
                        next_counter=
                        ↪  c_convert;
                    end
                    EXPOSE: begin
                        state       =
                        ↪  CONVERT;
                        counter     =
                        ↪  next_counter;
                        next_counter=
                        ↪  c_read;
                    end
                    CONVERT: begin
                        state       =
                        ↪  READ0;
                        counter     =
                        ↪  next_counter;
                        next_counter=
                        ↪  c_read;
                    end
                    READ0: begin
                        state       =
                        ↪  READ1;
                        counter     =
                        ↪  next_counter;
                        next_counter=
                        ↪  c_erase;
                    end
                    READ1: begin
```

```
                        state       =
                        ↪  ERASE;
                        counter     =
                        ↪  next_counter;
                        next_counter=
                        ↪  c_expose;
                    end
                endcase
            end //end if(!counter)
        end//end else
        counter = counter -1;
    end //end always_ff

endmodule
```

J. pixelTop.v

```
`timescale 1ns / 1ps

module PIXEL_TOP (
    input   logic   clk,
    input   logic   reset,
    input   logic   VBN,
    input   logic   RAMP,
    output  logic [31:0]  dataOut
);
    logic [7:0] DataOut1, DataOut2;
    wire erase, read0, read1, expose,
    ↪  convert;
    PIXEL_STATE pState1(clk, reset, erase,
    ↪  read0, read1, expose, convert);
    PIXEL_ARRAY pa1(VBN, RAMP, reset,
    ↪  erase, expose, read0, read1,
    ↪  convert, DataOut1, DataOut2);

    always_ff @(posedge clk or posedge
    ↪  reset)begin
        if(reset)begin
            dataOut =0;
        end
        else begin
            if(read0)begin
                dataOut[7:0]    <=
                ↪  DataOut1;
                dataOut[15:8]   <=
                ↪  DataOut2;
            end//Her leser vi ut til
            ↪  databussen avhengig av
            ↪  hvilken read-state vi er i
            else if (read1) begin
                dataOut[23:16]  <=
                ↪  DataOut1;
                dataOut[31:24]  <=
                ↪  DataOut2;
            end
        end
    end

    //"Driver" de analoge signalene
    assign RAMP = convert ? clk : 0; //Så
    ↪  lenge denne kjører, øker adc i
    ↪  pixelsensoren frem til adc>tmp
    assign VBN = expose ? clk : 0; //Clk
    ↪  eller 0 avhengig av om
    ↪  state=expose - da synker tmp på
    ↪  hver posedge clk

endmodule
```

REFERENCES

[1] Kleinfelder, Lim, Liu, Gamal “A 10 000 Frames/s CMOS Digital Pixel Sensor”, JSSC, VOL 36, NO 12, 2001

[2] Liu, C., Bainbridge, L., Berkovich, A., Chen, S., Gao, W., Tsai, T.-H., … Nakamura, J. (2020). A 4.6μm, 512×512, Ultra-Low Power Stacked Digital Pixel Sensor with Triple Quantization and 127dB Dynamic Range. Presented at the 2020 IEEE International Electron Devices Meeting (IEDM). https://doi.org/10.1109/iedm13553.2020.9371913

[3] Wulff, Carsten, Design of Integrated Circuits Examples, (2021), https://github.com/wulffern/dicex

[4] Moyhnihan, Tim, ”CMOS Is Winning the Camera Sensor Battle, and Here’s Why” (2011). Retrived 18.11.2021 from https://www.techhive.com/article/246931/cmos-is-winning-the-camera-sensor-battle-and-heres-why.html