Digital Pixel Sensor

Simen Domaas, Abasin Essa

v1.0

November 19, 2021

Abstract

The design and modeling of a 2x2 pixel array with CMOS technology will be described. The system uses an ADC per pixel and row-wise digital readout. Transistors with width 0.5μm and lengths in the range of 0.1-0.5μm will be used in the design. The system will be implemented and simulated using SPICE and Verilog. Simulations verify that the functionality of the analog and digital part of the system is as intended when the photoresistor in the pixel sensor circuit has a size in the range of 0.5-100GΩ.

Contents

1 Introduction **4**

2 Theory **4**
- 2.1 SPICE . . . 4
- 2.2 Verilog . . . 4
- 2.3 FSM . . . 4

3 Implementation **5**
- 3.1 Analog . . . 5
  - 3.1.1 Sensor . . . 5
  - 3.1.2 Comparator . . . 6
  - 3.1.3 Transistor sizes and inversion level . . . 8
  - 3.1.4 Memory . . . 9
- 3.2 Digital . . . 9
  - 3.2.1 State Machine . . . 10
  - 3.2.2 Readout . . . 11

4 Results **12**
- 4.1 Analog . . . 12
  - 4.1.1 Sensor circuit verification . . . 12
  - 4.1.2 Variation in expose time . . . 13
  - 4.1.3 Variation in photoresistor values . . . 15
  - 4.1.4 Stored memory values . . . 16
- 4.2 Digital . . . 16
  - 4.2.1 PixelArray . . . 16
  - 4.2.2 PixelTop . . . 16

5 Discussion **17**
- 5.1 Analog . . . 17
  - 5.1.1 Sensor circuit validation . . . 17
  - 5.1.2 Photoresistor variation . . . 17
  - 5.1.3 Expose variation . . . 18
- 5.2 Digital . . . 18
- 5.3 Readout . . . 19

6 Future work **19**

7 Conclusion **19**

References **20**

A Appendix **21**

B Code **22**

B.1 Verilog 22

B.2 SPICE 30

1 Introduction

CMOS image sensor technology allows for timers and analog-to-digital converters (ADCs) to be integrated in to a single chip. Implementing ADCs and memory for each pixel offers the advantages of a digital readout and a reduction in analog circuit demands. The readout and analog to digital conversion can be performed in parallel, which increases the speed of the system. CMOS image sensors are also useful as they can be low-powered, which is an important feature in devices such as smart phones and digital cameras.

This report is inspired by [1], a paper which presents the design and implementation of a 352x288 digital pixel sensor with per-pixel ADC and memory. The paper is from 2001, and the high speed digital readout of 10 000 frames per second and 0.18-μm CMOS process is no longer state-of-the-art. Even though the technology described in [1] has been replaced by more advanced methods, such as the ones presented in [2] and [3], the ideas and concepts can be used as fundamentals for further development.

This report describes a theoretical implementation of a 2x2 block of pixels using CMOS technology. The system uses a single slope ADC and a row-wise digital readout. It is implemented and simulated using SPICE and Verilog.

2 Theory

2.1 SPICE

SPICE, short for *Simulation Program with Integrated Circuits Emphasis,* is a modeling and computer simulation program developed in the 1970s. It is used to predict the behavior and simulate electronic circuits. SPICE offers DC, AC, transient and noise and sensitivity analysis as well as built-in models of components such as transistors and diodes.

2.2 Verilog

Verilog is a Hardware Description Language (HDL) and is used to model electronic systems [5, 41]. HDLs allow for the hardware of digital circuits to be described in textual form. Verilog is a behavioral abstraction of the circuit which unlike SPICE sacrifices accuracy for the sake of run time.

2.3 FSM

A Finite State Machine (FSM) is an abstract machine and mathematical model of computation that can be in one of a finite number of states at any given time. FSMs are used to model or recognize patterns. When a certain input is given the machine changes its state accordingly.

3 Implementation

The system is implemented as a combination of an analog and a digital system. The analog part uses the analog input that is light and converts this to a digital output. The digital combines multiples of the analog pixels and controls the readout and the different states of the system. The code used to implement the system in SPICE and Verilog can be found in [B].

3.1 Analog

Figure 1 illustrates a simple block diagram of a pixel. The analog system for a pixel consists of a sensor circuit, a comparator and an 8-bit memory. This section explains the design and the functionality of each subsystem. The various subsystems were implemented in SPICE.

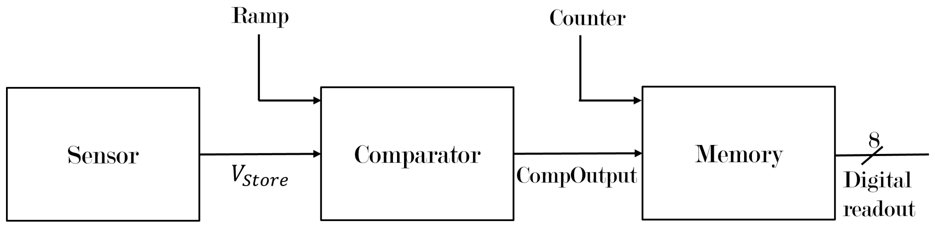


Figure 1: Complete system block diagram of a pixel.

3.1.1 Sensor

Figure 2 shows the sensor circuit. The main purpose of the circuit is to store the information from detected photons as an analog voltage value V_{Store}.

The main elements of the circuit is a photoresistor R_{Photo}, a transfer gate transistor M_{Exp}, a reset transistor M_{Erase} and a capacitor C_S. M_{Exp} and M_{Erase} are implemented as switches in figure 2. When V_{Erase} is high, a reference voltage V_{reset} is set over V_{Store} and C_S is fully charged.

When *Expose* is high C_S will discharge over time and the voltage value V_{Store} will decrease. The difference in V_{Store} before and after exposure forms a simplified model of charges generated by the emission of photons.

Table 1 shows the dimensions of M_{Exp} and M_{Erase}. W is the width and L is the length of the transistors. M_{exp} and M_{Erase} were implemented with

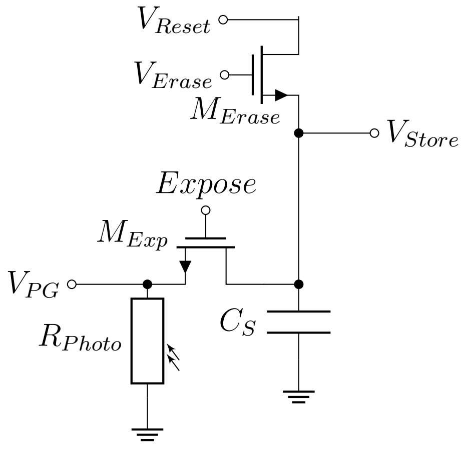


Figure 2: Sensor schematic.

NMOS models with higher oxide thickness then the rest of the transistors in the pixel. Thicker oxide minimize subthreshold and high gate leakage current. Table 2 presents the sizes of C_S and R_{Photo} in the implemented circuit.

Table 1: Transistor sizes in the implemented sensor circuit.

Transistor	W	L	W/L
M_{Exp}	0.5μm	0.1μm	5
M_{Erase}	0.5μm	0.1μm	5

Table 2: Values of components used in the pixel sensor.

Component	Value
C_s	100 F
R_{Photo}	1 GΩ

3.1.2 Comparator

Similar to [1], a linear ramp approach is used in the A/D conversion. A linear analog ramp V_{Ramp} is swept until it exceeds V_{Store}. Simultaneously, a digital counter iterates from zero. When V_{Ramp} has a higher value than V_{Store}, the comparator output *CompOutput* switches to zero, and the value of the counter is latched in the pixel's memory.

The Comparator is designed as a differential pair followed by a common-source and a CMOS inverter. Figure 3 shows the fully implemented comparator. The purpose of the differential pair is to compare the signal from the sensor V_{Store} to V_{Ramp}. It will highlight the difference between the two inputs, and it will naturally eliminate noise or interference that is present in both input signals.

The differential pair has a current mirror as an active load. This provides certain benefits in the system such as increasing the output impedance and saving area on the chip. NMOS transistors are used to implement the differential pair while the active current mirror load is implemented using PMOS transistors.

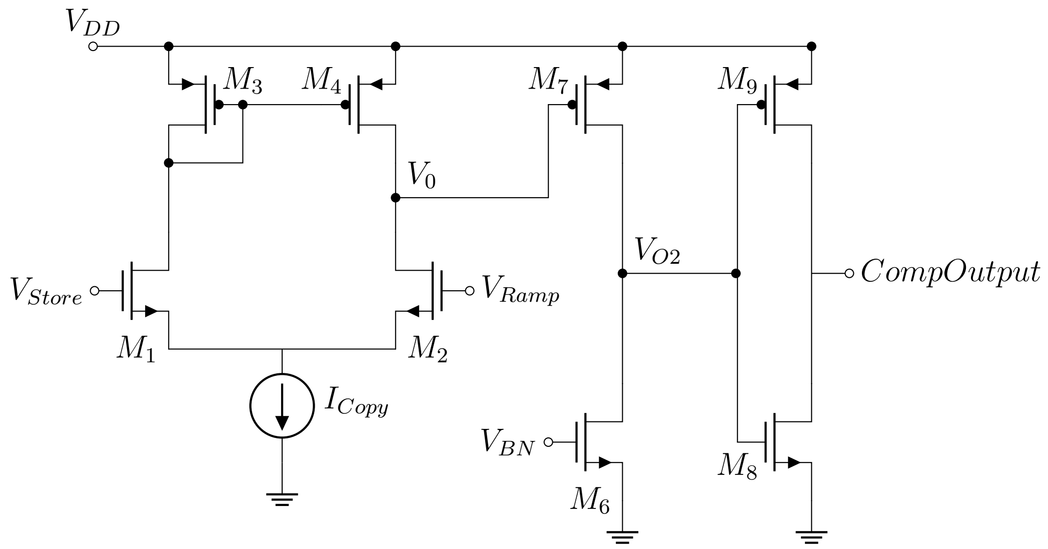


Figure 3: Implemented comparator. It consists of a differential pair, a common-source follower and an inverter.

The output of the differential pair V_O is connected to the gate of M_7. V_O will vary based on the inputs of the differential pair and therefore the current from M_7 will vary. If M_7 can provide a large enough current it can pull the node V_{O2} high. A consequence of this operation is that it will flip the polarity of V_0. Because of this the output of the common-source is connected to an inverter to flip back the polarity.

The bias current I_{Copy} of the differential pair was implemented with a current mirror. Figure 4 illustrates the current mirror that provides the current. I_{Bias} is an external part of the system and has a value of 1μA. The current mirror is implemented such that I_{Bias}=I_{Copy}.

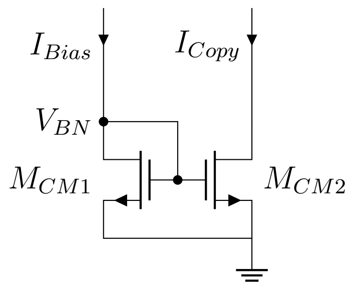


Figure 4: Current mirror that provides the current in the comparator.

3.1.3 Transistor sizes and inversion level

The transistors M_{CM1} and M_{CM2} in figure 4 were designed such that they operate in strong inversion. For current mirrors, it is known that $(\frac{\sigma_{I_D}}{I_D})^2 \propto (\frac{gm}{I_D})^2$ where $\sigma^2_{I_D}$ is the standard deviation in the current. Current mirrors achieve best matching in strong inversion because strong inversion provides a lower value of gm [9].

Transistors M_1 and M_2 in figure 3 were designed such that they operate in weak inversion. For a differential pair, it is known that $\sigma_V^2 \propto (\frac{I_D}{gm})^2$ where gm is the transconductance of a transistor and σ_V^2 is the variance in voltage. Weak inversion provides a higher gm than strong inversion and hence a better match [9]. However, weak inversion has some disadvantages. For instance it will slow down the transistor. A better matching has been prioritized over the speed of the transistors.

The transistors sizes were selected based on inversion conditions for different transistors and the current in the various parts of the circuits. Two different sizes were selected as basis sizes for all transistors in the comparator which makes the physical realization more practical. Also, SPICE has a certain limitation in the dimensions of a transistor. Table 3 shows the two different sizes A and B. A and B has the same W, but different L.

Table 3: Sizes used in implementation of the ADC.

Size	**W**	**L**	**W/L**
A	0.5μm	0.5μm	1
B	0.5μm	0.15μm	3

The information presented in [7] was used to achieve the desired inversion level. Figure A.3 in [A] illustrates the required μA/sq ratio needed to achieve a certain inversion level. The parameter sq is defined as $\frac{W}{L}$ of a transistor.

For an NMOS transistor of size A to achieve strong inversion in the current mirror in figure 4, a ratio of 6 μA/sq is needed. As I_{Bias} and I_{Copy} provide 1 μA in the circuit, 6 transistors were stacked in series to achieve strong inversion. Equation (1) shows how putting transistors in series gives the desired ratio for the current mirror.

$$\frac{I_{Bias/Copy}}{\frac{W}{6 \cdot L}} \implies \frac{1\mu A}{\frac{0.5\mu m}{6 \cdot 0.5\mu m}} = 6\mu A/sq \tag{1}$$

Differential pair transistors M_1 and M_2 were implemented with size B. If the input signals of the differential pair V_{Ramp} and V_{Store} are equal, it is expected that $I_{Copy} = 1\mu A$ divides into two equal sizes. Both M_1 and M_2 will have an I_D value of 0.5μA. This condition will result in 0.15 μA/sq which gives an NMOS transistor near weak inversion based on figure A.3. In an extreme case one can expect $V_{Store} >> V_{Ramp}$ and $I_D \approx 1\mu$A for M_1. This condition gives approximately 0.3μA/sq for M_1 and 0μA/sq for M_2. The chosen dimensions of M_1 and M_2 ensure that transistors operate in the correct inversion level.

The current mirror transistors M_3 and M_4 are PMOS transistors. Because of mobility difference between holes and electrons, the μA/sq value will be $\frac{1}{6}^{th}$ of an NMOS transistor. M_3, M_4, M_7 and M_9 have size A.

Lastly, the remaining transistors M_6 was implemented with size A and M_8 was implemented with size B.

3.1.4 Memory

The memory was implemented using 3T dynamic memory cells [1]. Figure 5 shows one of the eight memory cells of the system. Writing to a memory cell is controlled by *CompOutput*. When *CompOutput* is high the voltage value is stored on the gate of M_{10}. When *Read* is high, M_{11} is closed and the stored voltage will be latched into *DataI/O*.

3.2 Digital

The digital logic is designed using Verilog and connects multiple pixel sensors, controls the different states of the system and stores data provided by the pixel sensors as digital values. Figure 6 is a block diagram describing the top level block logic. The block pixelArray is made up of four pixelSensor blocks. The finite state machine (FSM) pixelState controls the different states of the system and pixelTop is the top level block connecting the FSM and pixelArray.

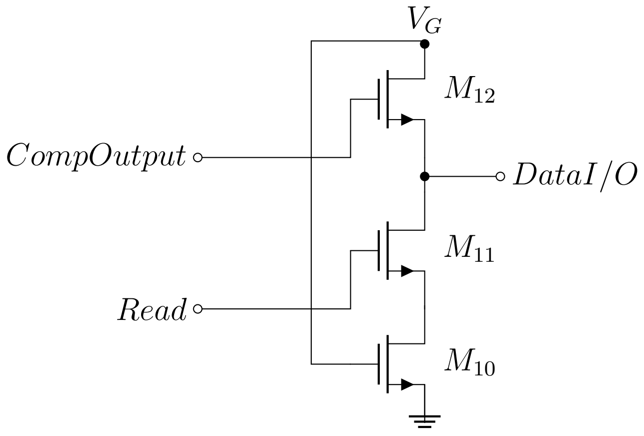


Figure 5: Single memory cell.

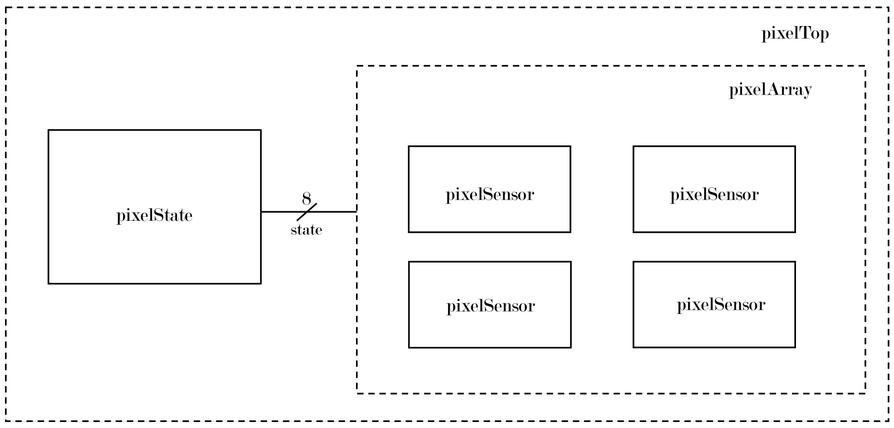


Figure 6: Block diagram for the digital part of the system.

3.2.1 State Machine

Figure 7 illustrates the finite state machine controlling the states of the system. The FSM is based on the FSM given in [4] and has six states: $IDLE$, $ERASE$, $EXPOSE$, $CONVERT$, $READ1$ and $READ2$. The two read states are used when reading from the two different rows in the 2x2 pixel array.
When the system is in $READ1$ a signal $read1$ is set to logic high while the other four signals are low. $ERASE$, $CONVERT$, $READ2$ and $EXPOSE$ are implemented the same way setting only their respective signals high. $IDLE$ resets all signals including the counter used for the timing of each state before transitioning to the next state.

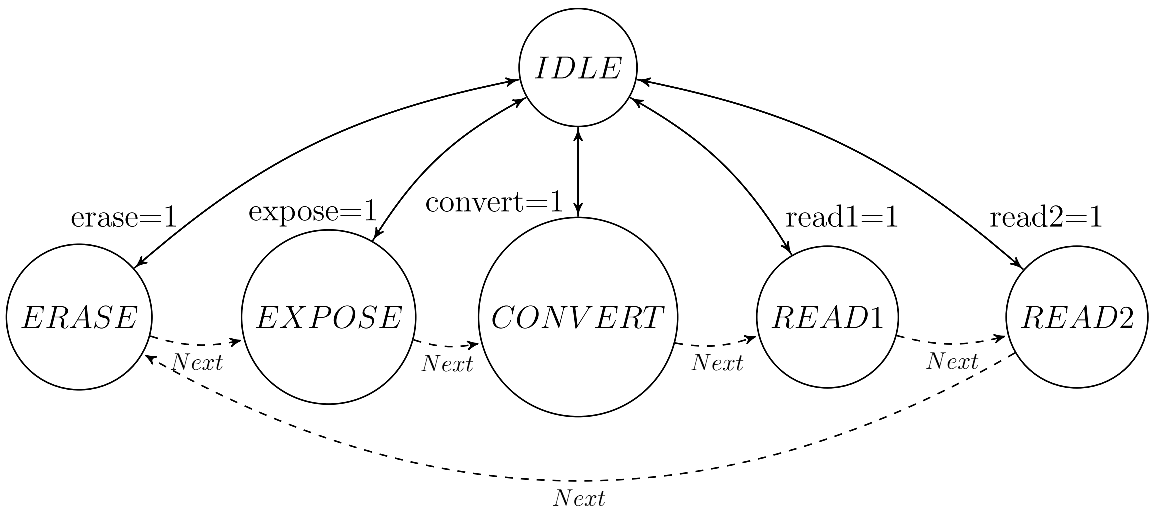


Figure 7: State diagram with all six states.

3.2.2 Readout

The readout of data happens row by row. The two rows in the 2x2 pixel array use separate read signals $read1$ and $read2$. When the read signal for a given row is high the data stored in the local memory of the pixel is loaded on to a bus. Pixel sensors that share read signals have separate data buses $pixelDataOut1$ and $pixelDataOut2$. The main blocks and signals used in the read states are illustrated in figure 8.

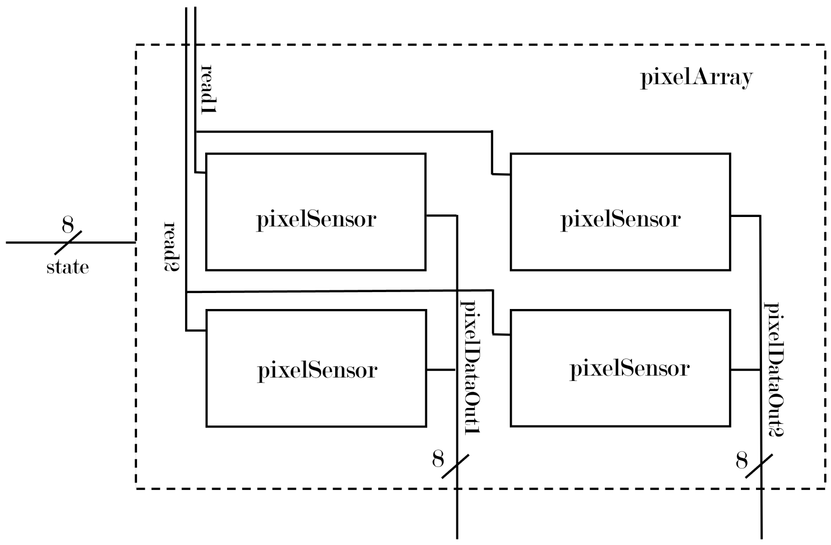


Figure 8: Block diagram illustrating the readout process.

4 Results

The analog pixel sensor circuit and the digital control circuit are simulated independently of each other using SPICE and Verilog.

4.1 Analog

The analog system was simulated with different values of *Expose* and R_{Photo} to verify and test the pixel circuit and the subsystems in different conditions. *Convert*, *Erase*, V_{Reset} and *Read* were fixed values for all simulations.

4.1.1 Sensor circuit verification

To validate and examine the behavior of the various components and signals, the sensor circuit was simulated. Figure 9 shows the simulation.

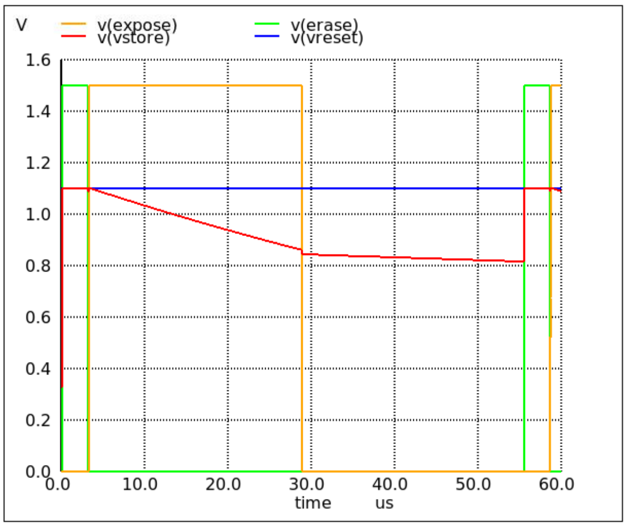
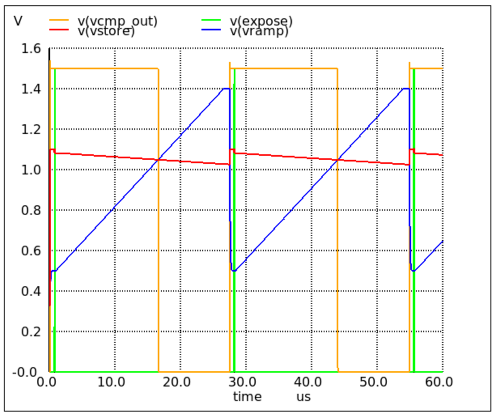


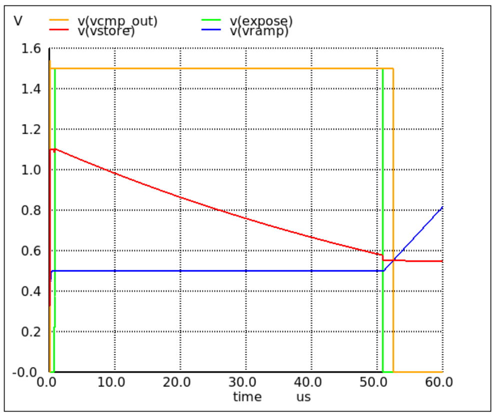
Figure 9: Sensor circuit simulation. The plot shows the following signals: V_{Store} (red), *Erase* (green), *Reset* (blue) and *Expose* (yellow).

4.1.2 Variation in expose time

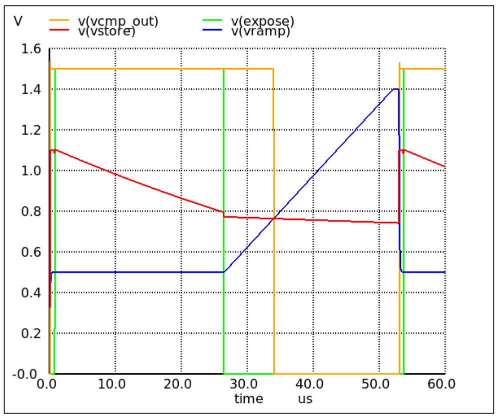
To simulate varied exposure time duration, a test bench was constructed. Figure 10 shows the simulations with *Expose* durations 25.6μs, 50.1μs and 1μs. R_{photo} was set to 1GΩ for the simulations.



(a) $Expose = 1\mu$s



(b) $Expose = 50.1\mu$s

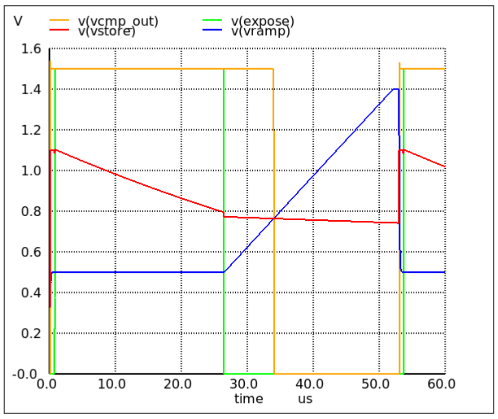


(c) $Expose = 25.6\mu$s.

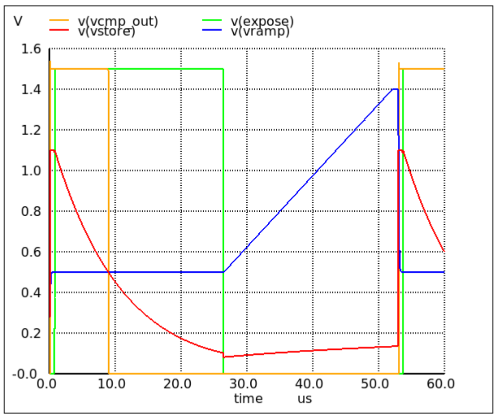
Figure 10: The different signals found in a pixel have been plotted with different $Expose$ pulse width. The plot shows the following signals: $V_{CompOut}$ (yellow), V_{Store} (red), V_{Ramp} (blue) and $Expose$ (green).

4.1.3 Variation in photoresistor values

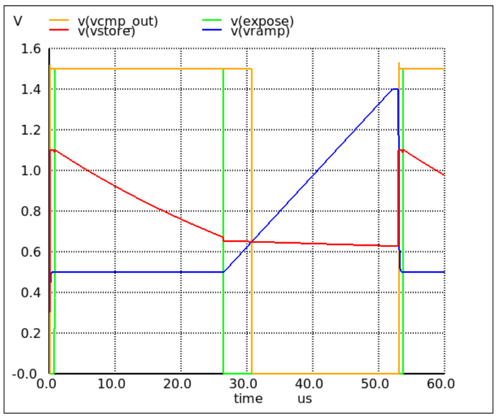
The system's behavior was simulated with different R_{Photo} values. Figure 11 shows the results with R_{Photo} values 1GΩ, 0.1GΩ, 0.5GΩ and 100GΩ. *Expose* was set to 25.6μ for the simulations.



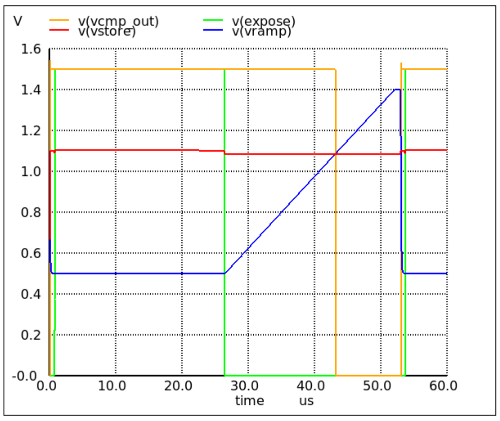
(a) R_{photo} = 1G Ω



(b) R_{photo} = 0.1G Ω



(c) R_{photo} = 0.5G Ω



(d) R_{photo} = 100G Ω

Figure 11: The different signals found in a pixel have been plotted with different R_{photo} values. The plot shows $V_{CompOut}$ (yellow), V_{Store} (red), V_{Ramp} (blue) and *Expose* (green).

4.1.4 Stored memory values

Different memory values were latched for the simulations with different values of R_{Photo} and *Expose*. Table 4 shows the stored value for a given parameter value. The start value of the ramp V_{Ref} has the value 0.5 V. The least significant bit has the following analog value:

$$LSB_{\text{analog value}} = \frac{V_{\text{Reset}} - V_{\text{ref}}}{256} \approx 0.0023V \tag{2}$$

Table 4: Stored memory values for the various values of R_{Photo} and *Expose*.

$\mathbf{R_{Photo}}$	***Expose***	**Stored binary value**	**Stored values in decimal (change in $\mathbf{V_{Store}}$)**	**Figure**
1GΩ	1μs	00010001_{binary}	0.0398 V	10a
1GΩ	50.1μs	10111111_{binary}	0.4477 V	10b
1GΩ	25.6μs	01110001_{binary}	0.2648V	11a
0.1GΩ	25.6μs	11111111_{binary}	0 V	11b
100GΩ	25.6μs	00000110_{binary}	0.0141 V	11d

4.2 Digital

To simulate the digital circuit in Verilog digital representations of the analog ramp, bias and V_{store} are made. These are not accurate representations of the signals in the analog circuit, but they can still be used to verify if the digital processes work as intended.

4.2.1 PixelArray

It is tested if reading from the pixels happens row by row in the *pixelArray* test bench. Figure 12 shows the digital value stored in each pixel in orange. The two read signals $read1$ and $read2$ are in green and the data buses $pixelDataOut1$ and $pixelDataOut2$ are blue.

4.2.2 PixelTop

The transition between states and the use of the FSM to control the pixel array is tested in the *pixelTop* test bench. Figure 13 shows selected signals from the results of the test bench. As the value of *states* changes the system goes from the $CONVERT$ to $READ1$, $READ2$, $ERASE$ and $EXPOSE$. During $ERASE$ the value *p_data*, represented by the orange graph, is reset.

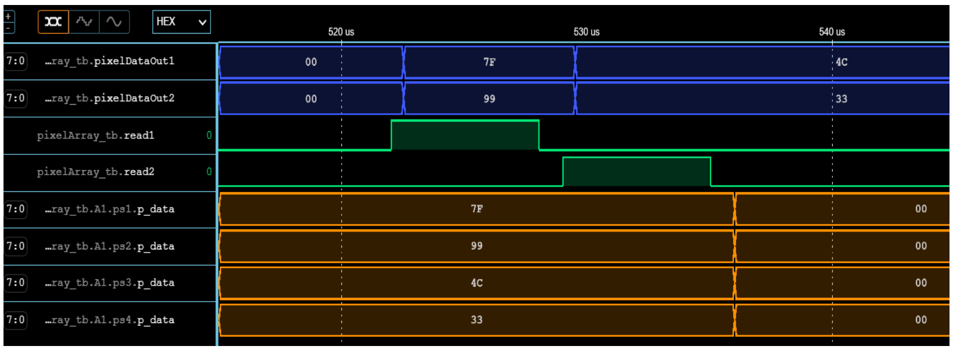


Figure 12: Results from the *pixelArray* test bench.

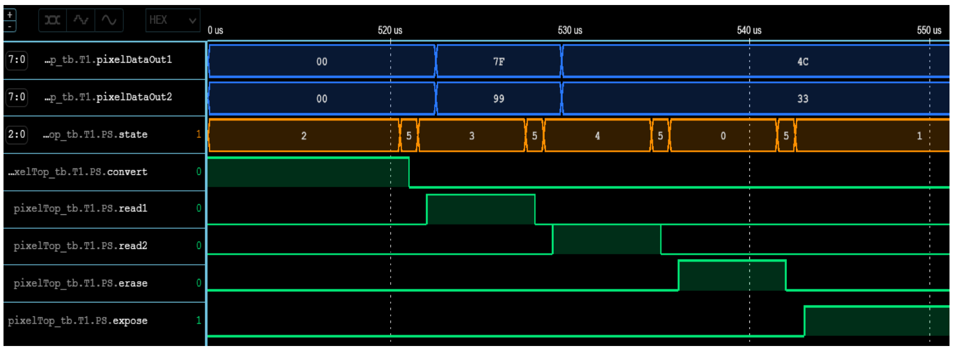


Figure 13: Results from the *pixelTop* test bench.

5 Discussion

5.1 Analog

5.1.1 Sensor circuit validation

Figure 10c illustrates the functionality of the sensor circuit in figure 2. It shows that during *Erase*, V_{Store}=V_{Reset}. This means that M_{Erase} operates as a switch. In addition, the figure shows that exposing leads to a discharging of C_S, and we can conclude that M_{Exp} operates as a switch.

5.1.2 Photoresistor variation

Figure 11b illustrates that the analog pixel sensor fails when R_{photo}=0.1GΩ. C_s will discharge faster because of lower resistance value and V_{store} will have a lower value then V_{Ramp} before the conversion process begins. Therefore a voltage value of 0 V (11111111_{binary}) will be stored in the memory. Contrary, a large value of R_{photo} leads to a slower discharge rate. Figure 11d illustrates this observation. For R_{Photo}=100GΩ a voltage value 0.0141 V (00000110_{binary}) was stored in the memory. Ideally 00000000_{binary} should have been stored in the memory.

Figure 11a, 11c and table 4 show that an approximately correct digital value is stored for R_{Photo} values 0.5GΩ and 1GΩ. The stored binary values are correct because their decimal representations are directly proportional to the amount of voltage loss in V_{store} during *Expose*.

From these observations, one can conclude the pixel sensor has approximately a limitation of $0.5\text{G}\Omega < R_{photo} < 100\text{G}\Omega$ for the given *Convert* pulse width. This means that the system's reliability is dependent on the lighting environment to some degree.

5.1.3 Expose variation

Figure 10 and table 4 illustrate how the duration of *Expose* can have an effect on the stored value in memory. Exposing longer leads to a longer discharge period for V_{Store}. Therefore a larger value will be stored in the memory. A shorter *Expose* leads to a lower value stored in the memory.

After exposure V_{Store} does not maintain a constant voltage value and a small change in voltage can be observed in the figures. Because of this the value stored in the memory might not be correct since the voltage has changed after $V_{Store} = V_{Ramp}$. A smaller voltage value will be stored in the memory. A small but dramatically voltage drop is observed when *Expose* switches from high to low. This is due to the phenomenon hot-carrier injection [9]. In addition, V_{Store} decreases almost linearly after exposure. This is due to a leakage current in M_{Exp}.

5.2 Digital

Figure 12 shows that *read*1 and *read*2 are not high at the same time and reading from the two rows happens separately. The data stored in each pixel is reset after the readout, and the readout process has the intended functionality.

Based on the results presented in figure 13 the interaction between the state machine and the pixel array also work as intended. The states are not overlapping and using the state machine to control the pixel array does not change the results compared to the ones in figure 12.

5.3 Readout

When reading out from the pixels on the second row the 8-bit data from these rows replaces the data from the first row. This implementation assumes that the data on the bus is stored in a register immediately. Alternatively the data bus can be expanded to a 16-bit bus containing data from both rows. This works for a small pixel array, but is not a good solution for a chip containing hundreds or thousands of pixels as it would require very large buses.

6 Future work

The next step in further development of the system would be to expand from a 2x2 pixel array to an array of size nxn. This could be done by replicating the basic building block that is the 2x2 pixel array.

The counter used in the convert state could be implemented using a gray counter. Because the counter value only increases by one at a time, using a gray counter would decrease the number of transitions as only one bit changes every time the counter value increases. An advantage of this would be that less power is required and there would be a reduction in noise, which would be important aspects when expanding the system to a larger number of pixels.

The image sensor described in this report is simulated in Verilog and SPICE using theoretical values. At some point in future development it would be necessary to test actual hardware as results may differ from the ones obtained from simulation tools.

7 Conclusion

The design and modeling of a 2x2 pixel array with CMOS technology was described. The system was implemented with an ADC per pixel and row-wise digital readout. Transistors with width 0.5μm and lengths in the range of 0.1-0.5μm was used in the design. The system was implemented and simulated using SPICE and Verilog. Simulations verified that the functionality of the analog and digital part of the system was as intended when the photoresistor in the pixel sensor circuit had a size in the range of 0.5-100GΩ.

References

[1] Stuart Kleinfelder, SukHwan Lim, Xinqiao Liu, and Abbas El Gamal, *A 10 000 Frames/s CMOS Digital Pixel Sensor*, 2001.

[2] Danilo Bronzi, Federica Villa, Simone Tisa, Alberto Tosi, Franco Zappa, Daniel Durini, Sascha Weyers, Werner Brockherde, *100 000 Frames/s 64 × 32 Single-Photon Detector Array for 2-D Imaging and 3-D Ranging*, 2014.

[3] Yunhong Kim, Heesung Chae, Kyung-Min Kim, Kyungtae Kim, Sukki Yoon, Kyoungmin Koh, Jesuk Lee, Yongin Park, *A 1/3-Inch 1.12μm-Pitch 13Mpixel CMOS Image Sensor with a Low-Power Readout Architecture*, 2020.

[4] Wolff Carsten, *dicex, GitHub repositories*, `https://github.com/wulffern/dicex` 2021.

[5] Neil H. E. Weste, David Money Harris, *CMOS VLSI Design A Circuits and Systems Perspective*, Fourth Edition, 2011.

[6] Higgins, Richard J *Electronics with digital and analog integrated circuits*, `https://archive.org/details/electronicswithd0000higg`, 1983

[7] Wulff, Carsten *TFE4152, Lecture 10 - SPICE*, NTNU 2021,

[8] Berkeley University *AIM-Spice Reference Manual*, January 2017,

[9] Tony Chan Carusone, David A. Johns, Kenneth W. Martin, *Analog Integrated Circuit Design*, 2011,

A Appendix

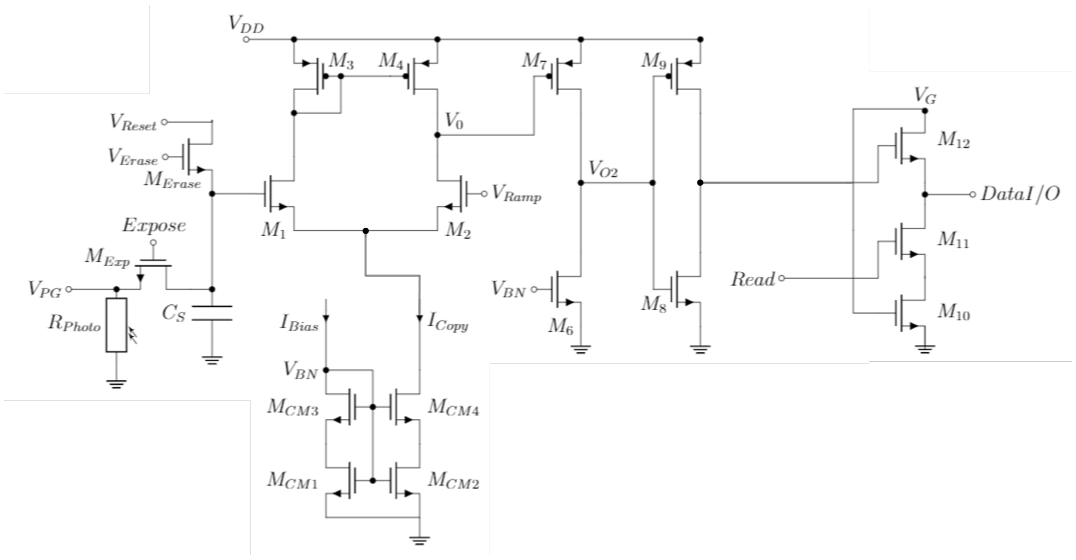


Figure A.1: Complete circuit schematics of a pixel .

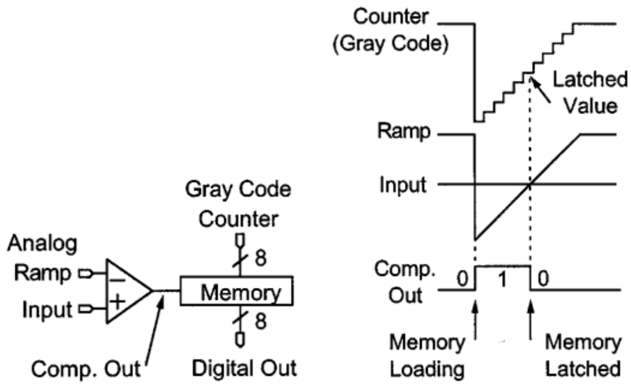
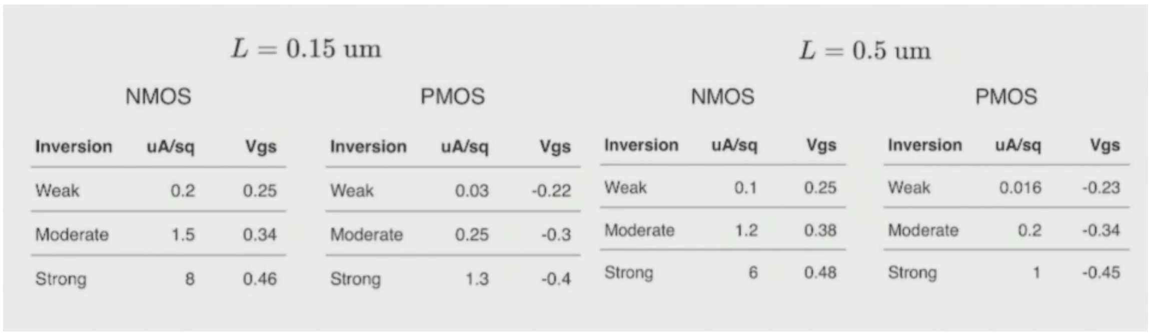


Figure A.2: ADC operation [1].



$L = 0.15$ um						$L = 0.5$ um					
NMOS			PMOS			NMOS			PMOS		
Inversion	uA/sq	Vgs	Inversion	uA/sq	Vgs	Inversion	uA/sq	Vgs	Inversion	uA/sq	Vgs
Weak	0.2	0.25	Weak	0.03	-0.22	Weak	0.1	0.25	Weak	0.016	-0.23
Moderate	1.5	0.34	Moderate	0.25	-0.3	Moderate	1.2	0.38	Moderate	0.2	-0.34
Strong	8	0.46	Strong	1.3	-0.4	Strong	6	0.48	Strong	1	-0.45

Figure A.3: Sizes and scale from lecture 10 [7].

B Code

The following code is the code needed to reproduce the work and implement the system in SPICE and Verilog. The complete code used to implement the system described in this report, including makefiles and test benches, can be found in the following GitHub repsitory: `https://github.com/simendo/IC`

B.1 Verilog

pixelSensor.v

```verilog
//====================================================================
//        Copyright (c) 2021 Carsten Wulff Software, Norway
// ===================================================================
// Created       : wulff at 2021-7-21
// ===================================================================
//  The MIT License (MIT)
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy
//  of this software and associated documentation files (the "Software"), to deal
//  in the Software without restriction, including without limitation the rights
//  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
//  copies of the Software, and to permit persons to whom the Software is
//  furnished to do so, subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
```

```
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
//  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
//  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
//  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
//  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
//  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
//  SOFTWARE.
//
//===========================================================

//-----------------------------------------------------------
// Model of pixel sensor, including
//  - Reset
//  - The sensor
//  - Comparator
//  - Memory latch
//  - Readout of latched value
//-----------------------------------------------------------
module PIXEL_SENSOR
  (
   input logic      VBN1,
   input logic      RAMP,
   input logic      RESET,
   input logic      ERASE,
   input logic      EXPOSE,
   input logic      READ,
   inout [7:0] DATA
   );

   real             v_erase = 1.2;
   real             lsb = v_erase/255; //step value of ramp
   parameter real   dv_pixel = 0.5;

   real             tmp; //voltage over capasitor
   logic            cmp; //compares the ramp and tmp
   real             adc; //digital representation of analog ramp

   logic [7:0]      p_data;

   //---------------------------------------------------------
```

```
// ERASE
//---------------------------------------------------------
// Reset the pixel value on pixRst
always @(ERASE) begin
   tmp = v_erase;
   p_data = 0;
   cmp  = 0;
   adc = 0;
end

//---------------------------------------------------------
// SENSOR
//---------------------------------------------------------
// Use bias to provide a clock for integration when exposing
always @(posedge VBN1) begin
   if(EXPOSE)
     tmp = tmp - dv_pixel*lsb; //discharge of capacitor
end

//---------------------------------------------------------
// Comparator
//---------------------------------------------------------
// Use ramp to provide a clock for ADC conversion, assume that ramp
// and DATA are synchronous
always @(posedge RAMP) begin
   adc = adc + lsb; //increases ramp by lsb based on the RAMP clock.
   if(adc > tmp) //ramp = vstore
     cmp <= 1;
end

//---------------------------------------------------------
// Memory latch
//---------------------------------------------------------
always_comb  begin
   if(!cmp) begin
      p_data = DATA;
   end

end
```

```

    // -------------------------------------------------------------
    // Readout
    // -------------------------------------------------------------
    // Assign data to bus when pixRead = 0
    assign DATA = READ ? ~p_data : 8'bZ; //p_data is inverted to give the correct output
endmodule // re_control
```

pixelArray.v

```
module pixelArray(
    input logic           VBN1,
    input logic           RAMP,
    input logic           RESET,
    input logic           ERASE,
    input logic           EXPOSE,

    input logic    READ1,
    input logic    READ2,

    inout  [7:0]      pixData1,
    inout  [7:0]      pixData2,
    inout  [7:0]      pixData3,
    inout  [7:0]      pixData4

);
     //dv_pixel is assigned different values for the differnet pixels to differentiate the results from each other.
     parameter real    dv_pixel1 = 0.5;
     parameter real    dv_pixel2 = 0.4;
     parameter real    dv_pixel3 = 0.7;
     parameter real    dv_pixel4 = 0.8;

     //instantiate the four pixels
     PIXEL_SENSOR #(.dv_pixel(dv_pixel1)) ps1(VBN1, RAMP, RESET, ERASE, EXPOSE,  READ1,  pixData1);
     PIXEL_SENSOR #(.dv_pixel(dv_pixel2)) ps2(VBN1, RAMP, RESET, ERASE, EXPOSE,  READ1,  pixData2);
     PIXEL_SENSOR #(.dv_pixel(dv_pixel3)) ps3(VBN1, RAMP, RESET, ERASE, EXPOSE,  READ2,  pixData3);
     PIXEL_SENSOR #(.dv_pixel(dv_pixel4)) ps4(VBN1, RAMP, RESET, ERASE, EXPOSE,  READ2,  pixData4);


endmodule
```

pixelState.v

```
module pixelState(


   input logic clk,
   input logic reset,
   output logic erase,
   output logic expose,
   output logic read1,
   output logic read2,
   output logic convert

);


   //------------------------------------------------------------
   // State Machine
   //------------------------------------------------------------
   parameter ERASE=0, EXPOSE=1, CONVERT=2, READ1=3, READ2=4, IDLE=5;

   logic [2:0]          state, next_state;   //States
   integer            counter;      //Delay counter in state machine

   //Duration of each state in clock cycles
   parameter integer c_erase = 5;
   parameter integer c_expose = 255;
   parameter integer c_convert = 255;
   parameter integer c_read1 = 5;
   parameter integer c_read2 = 5;

   // Control the output signals
   always_ff @(negedge clk ) begin
      case(state)
        ERASE: begin
           erase <= 1;
           read1 <= 0;
           read2 <= 0;
           expose <= 0;
           convert <= 0;
        end
        EXPOSE: begin
           erase <= 0;
           read1 <= 0;
           read2 <= 0;
           expose <= 1;
           convert <= 0;
        end
        CONVERT: begin
           erase <= 0;
           read1 <= 0;
           read2 <= 0;
           expose <= 0;
           convert = 1;
        end
        READ1: begin
           erase <= 0;
           read1 <= 1;
           read2 <= 0;
           expose <= 0;
           convert <= 0;
        end
        READ2: begin
           erase <= 0;
           read1 <= 0;
           read2 <= 1;
           expose <= 0;
           convert <= 0;
        end
        IDLE: begin
           erase <= 0;
           read1 <= 0;
           read2 <= 0;
           expose <= 0;
           convert <= 0;

        end
      endcase // case (state)
   end // always @ (state)


   // Control the state transitions.
   always_ff @(posedge clk or posedge reset) begin
      if(reset) begin
         state = IDLE;
         next_state = ERASE;
         counter  = 0;
         convert  = 0;
      end
      else begin
         case (state)
           ERASE: begin
              if(counter == c_erase) begin
                 next_state <= EXPOSE;
                 state <= IDLE;
              end
           end
           EXPOSE: begin
              if(counter == c_expose) begin
                 next_state <= CONVERT;
                 state <= IDLE;
              end
           end
           CONVERT: begin
              if(counter == c_convert) begin
                 next_state <= READ1;
                 state <= IDLE;
              end
           end
           READ1:
             if(counter == c_read1) begin
                state <= IDLE;
                next_state <= READ2;
             end
            READ2:
             if(counter == c_read2) begin
                state <= IDLE;
                next_state <= ERASE;
             end
           IDLE:
             state <= next_state;
         endcase
         if(state == IDLE)
           counter = 0;
         else
           counter = counter + 1;
      end // case (state)
   end // always @ (posedge clk or posedge reset)
endmodule // test
```

pixelTop.v

```


module pixelTop(
    input logic clk,
    input logic reset,
    inout [7:0]     pixData1,
    inout [7:0]     pixData2,
    inout [7:0]     pixData3,
    inout [7:0]     pixData4
);
    logic read1;
    logic read2;
    logic convert;
```

```
     logic expose;
     logic erase;
     logic anaBias1;
     logic anaRamp;
     logic anaReset;
     logic[7:0] data1;
     logic[7:0] data2;
     logic[7:0] data3;
     logic[7:0] data4;


     //Instanciate the pixelarray and pixelstate
     pixelArray PA(anaBias1,anaRamp,anaReset,erase,expose,
    read1,read2,pixData1,pixData2,pixData3,pixData4);
     pixelState PS(.clk(clk),.reset(reset),.erase(erase),.
    expose(expose),.read1(read1),.read2(read2),.convert(
    convert));

     assign anaRamp = convert ? clk : 0;

    // During expoure, provide a clock via anaBias1.
    // A digital representation without much resemblence to
    real world.
    assign anaBias1 = expose ? clk : 0;

    // If we're not reading the pixData, then we should drive
    the bus
    assign pixData1 = read1 ? 8'bZ: data1;
    assign pixData2 = read1 ? 8'bZ: data2;
    assign pixData3 = read2 ? 8'bZ: data3;
    assign pixData4 = read2 ? 8'bZ: data4;

    // When convert, then run a analog ramp (via anaRamp clock
    ) and digtal ramp via
    // data bus.
    always_ff @(posedge clk or posedge reset) begin
       if(reset) begin
          data1 =0;
          data2 =0;
          data3 =0;
          data4 =0;
       end
       if(convert) begin
          data1 += 1;
          data2 += 1;
          data3 += 1;
          data4 += 1;
       end
       else begin
```

```
         data1 = 0;
         data2 = 0;
         data3 = 0;
         data4 = 0;
      end
   end // always @ (posedge clk or reset)

   //
   ------------------------------------------------------------
   // Readout from databus
   //
   ------------------------------------------------------------
   logic [7:0] pixelDataOut1;
   logic [7:0] pixelDataOut2;
   always_ff @(posedge clk or posedge reset) begin
      if(reset) begin
         pixelDataOut1 = 0;
         pixelDataOut2 = 0;
      end
      else begin
         if(read1) begin
           pixelDataOut1 <= pixData1;
           pixelDataOut2 <= pixData2;
         end
         if(read2)begin
            pixelDataOut1 <= pixData3;
            pixelDataOut2 <= pixData4;
         end
      end
   end


    //wire = _net;


endmodule
```

B.2 SPICE

pixelSensor.cir

```

.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 VDD VSS COMP


XM1 READ VCMP_OUT DATA_7 DATA_6 DATA_5 DATA_4 DATA_3 DATA_2
   DATA_1 DATA_0 VDD VSS MEMORY

.ENDS

.SUBCKT MEMORY READ VCMP_OUT
+ DATA_7 DATA_6 DATA_5 DATA_4 DATA_3 DATA_2 DATA_1 DATA_0 VDD
    VSS

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

.ENDS

.SUBCKT MEMCELL CMP DATA READ VSS
M1 VG CMP DATA VSS nmos  w=0.2u  l=0.13u
M2 DATA READ DMEM VSS nmos  w=0.4u  l=0.13u
M3 DMEM VG VSS VSS nmos  w=1u  l=0.13u
C1 VG VSS 1p
.ENDS

.SUBCKT SERIESCURR DRAIN GATE SOURCE BULK w=0.5u l=0.5u
MS1 DRAIN GATE NODE1 BULK nmos W=w L=l
MS2 NODE1  GATE NODE2 BULK nmos W=w L=l
MS3 NODE2  GATE NODE3 BULK nmos W=w L=l
MS4 NODE3  GATE NODE4 BULK nmos W=w L=l
MS5 NODE4  GATE NODE5 BULK nmos W=w L=l
MS6 NODE5  GATE SOURCE BULK nmos W=w L=l
.ends

.SUBCKT SENSOR VRESET VSTORE ERASE EXPOSE VDD VSS

* Capacitor to model gate-source capacitance
C1 VSTORE VSS 100f
*MC65 VSS VSTORE VSS VSS nmos w=0.5u l=0.5u m 20
*Rleak VSTORE VSS 100T

* Switch to reset voltage on capacitor
*BR1 VRESET VSTORE I=V(ERASE)*V(VRESET,VSTORE)/1k
.model NMOS NMOS (LEVEL=14)
```

```


M99 VRESET ERASE VSTORE VSTORE nmos w=0.5u l=0.1u
* Switch to expose pixel
*BR2 VPG VSTORE I=V(EXPOSE)*V(VSTORE,VPG)/1k

M100 VPG EXPOSE VSTORE VSTORE nmos w=0.5u l=0.1u

*X23 VPG EXPOSE VSTORE VSTORE SERIESCURR
* Model photocurrent
Rphoto VPG VSS 1G
.ENDS



.SUBCKT SERIESCURR2 DRAIN GATE SOURCE BULK w=0.5u l=0.5u
MP1 DRAIN GATE NODE1 BULK pmos W=w L=l
MP2 NODE5  GATE SOURCE BULK pmos W=w L=l
.ends


.SUBCKT COMP VCMP_OUT VSTORE VRAMP VDD VSS

* Model comparator
*BC1 VCMP_OUT VSS V = ((atan(100000*(V(VSTORE) - V(VRAMP))))
   + 1.58)/3.14*1.5

*\\\diff pair
MP1 VP VP VDD VDD pmos w=0.5u l=0.5u
MP2 VO VP VDD VDD pmos w=0.5u l=0.5u
*XMP1 VP VP VDD VDD SERIESCURR2
*XMP2 VO VP VDD VDD SERIESCURR2

MN1 VO VRAMP VS VS nmos w=0.5u l=0.15u
MN2 VP VSTORE VS VS nmos w=0.5u l=0.15u
*bias current mirror
I3 0 VBN1 dc 1u

XM1 VBN1 VBN1 VSS VSS SERIESCURR
XM2 VS VBN1 VSS VSS SERIESCURR
*MB1 VBN1 VBN1 VSS VSS nmos w=0.5u l=0.15u
*MB2 VS VBN1 VSS VSS nmos w=0.5u l=0.15u

* Buffer
MP3 VO2 VO VDD VDD pmos w=0.5u l=0.15u
MN3 VO2 VBN1 VSS VSS nmos w=0.5u l=0.5u

*INVERTER
MIP1 VCMP_OUT VO2 VDD VDD pmos w=0.5u l=0.5u
```

```
MIN2 VCMP_OUT VO2 VSS VSS nmos w=0.5u l=0.15u





.ENDS
```