Design of Integrated Circuits (TFE4152): Digital Pixel Sensor Project

Charbel Badr, *ID: 557389* and Harish Nagaruru, *ID: 557392*

***Abstract*—In this paper, we propose a method of stacking digital pixel sensors in a 2x2 array. It is dealt with from the analog and digital perspectives. The results are encouraging and show that the overall system is coherent, robust and efficient.**

I. Introduction

The design of image sensors has long been under research. Before the prevalence of the digital domain, the image sensors were mainly of analog form. The sensors constitute the bedrock of technologies such as cameras and their efficiency is something vital for improved enhancements. This being said, we will look at the work done in [1], where they implemented a 355x288 digital pixel sensor and implement a simplified 2x2 array of digital pixel sensors. More specifically, we will focus on implementing a comparator using optimized dimensions transistors and control the states of the pixel array through the implementation of a state machine. We will simulate the analog part of the system in NgSpice and the digital part using iVerilog.

Previous works such as that of the authors in [1] focused on implementing an array of pixels by designing multiple pixel circuits in parallel. The pixel circuits constituted of a photogate, a comparator and an 8 bit memory cell. They used a photogate instead of photodiode due to the latter's high leakage. The photogate circuit used thick oxide transistors rather than thin oxide ones to avoid leakage currents. The comparator used a differential gain followed by a single-ended gain and an inverter. Lastly, the memory part were implemented using a 3T dynamic memory cells which can accomodate a hold time of 10ms and have a high speed readout. The proposed method achieved a speed of 10,000 frames per second with minimal power consumption of 50mW. Lately, the authors in [2] have implemented a video graphic array by the use of 640x480 time of flight CMOS sensors to achieve a depth map with the fixed pattern phase noise being compensated.

The rest of this paper is organized as follows: Section II offers a theoretical background of the technologies discussed in this paper. Section III provides the technical design or implementation of the proposed system. Section IV offers the experimental results of the system. Section V discusses the analysis of the results found. Finally, Section VI and Section VII provide a conclusion to the paper along with the future scope of the system.

II. Theory

In this section, we will explain in technical terms the main building blocks referred to in [1] as that is reflected in our proposed design. The three main blocks as discussed previously are the photogate, the comparator/analog-to-digital converter (ADC) and the memory block. We will also mention the importance of gray counters.

A. Photogate Circuit

The photogate circuit can be shown in Fig. 1 and will be explained further.

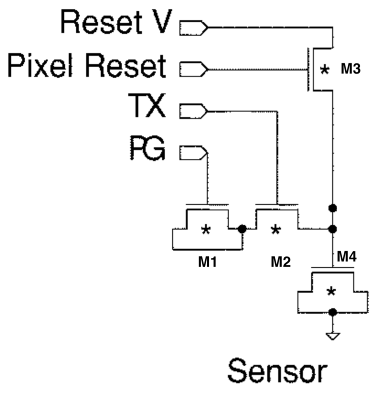


Fig. 1. Photogate Circuit

From the way these transistor are configured, M1 is the photogate, M2 and M3 are technically switches and M4 is modeled as a capacitor. That means the second stage after this circuit either carries the value of the photogate exposure or is reseted.

B. Comparator / ADC

The comparator circuit can be shown in Fig. 2 and will be explained further.

From the way this circuit looks, M5 acts as a current source in which it is branched into the differential pair (M3,M4). Also it can be seen that M1 and M2 act as a current mirror. Thus based on the gate source voltage of M3 and M4, we will determine which branch has a higher current value. This will affect the voltage drive of the node in between M2 and M4 based on the comparison of the ramp and input signal value. This node is then fed into the M6,M7 circuit which results in an output proportional to the comparison value in which it is fed to an inverter (M8,M9).

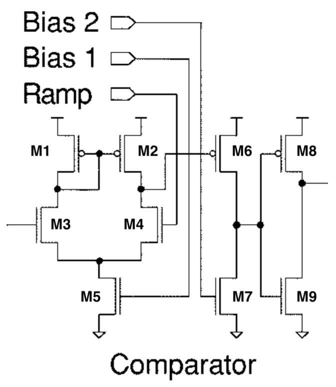


Fig. 2. Comparator Circuit

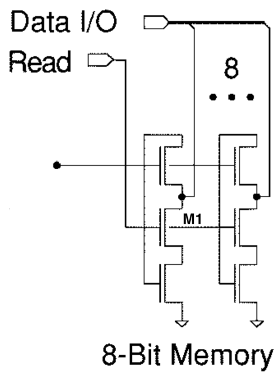


Fig. 3. Memory Circuit

C. Memory Circuit

The memory circuit can be shown in Fig. 3 and will be explained further.

In this circuit, transistor M1 acts as a switch when we want to read out the value. As long as the comparator output is not zero (conversion process is not done), we keep on updating the memory by Data IO until we stop when comparator is high and we store the value.

D. Gray Counters

The importance of gray code lies in the fact that the up-counting process changes 1 bit at a time and thus the switching is minimized and reliability is increased [3]. This significantly lowers the possibility of glitches in the system.

III. Implmentation

In this section, we will describe our work done in the analog and digital domains. For the analog part, we only implemented the comparator in terms of transistors. The photogate circuit and the memory was supplied by DICEX [4]. In the digital part, we organized the state machine which controls the behaviour of a 2x2 pixel array.

A. Comparator Circuit

We designed this circuit by getting inspired by the work of [1] and building up our circuit in NgSpice. The circuit can be found in Fig. 4 where we combine the two bias signals into one.

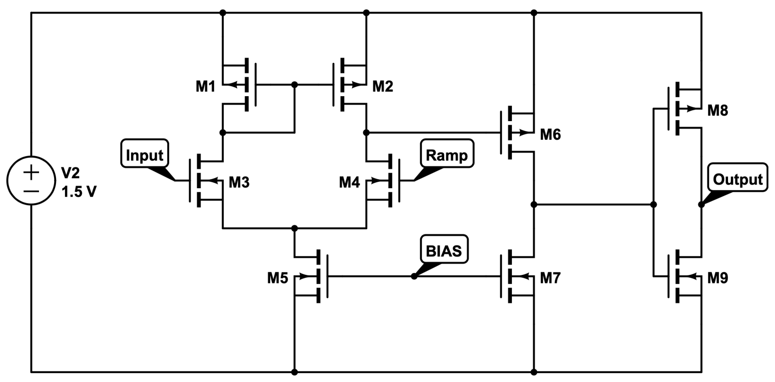


Fig. 4. Comparative Circuit

We chose the following values in Table I for our transistors width and lengths.

TABLE I
Widths and Lengths of the transistors

Transistor	Width [μm]	Length [μm]
M1	0.5	0.5
M2	0.5	0.5
M3	0.5	0.15
M4	0.5	0.15
M6	0.5	0.15
M7	0.5	0.15
M8	0.5	0.15
M9	0.5	0.15

We chose the bias current to be equal to 1μA. We also chose as a baseline two length values for our transistors which are 0.15μm and 0.5μm. We determine the width based on the following formula:

$$\mu A = \frac{\mu A}{sq} \times W/L \tag{1}$$

$$W = \frac{\mu A}{\frac{\mu A}{sq}} \times L \tag{2}$$

Referencing the lecture on Spice, we determine that our current mirrors need to be in strong inversion and our differential pair should be in weak inversion. The current in the two branches above M5 is almost 0.5μA. For the p-mos current mirror, the μA/sq= 1 for L=0.5μm. Thus W is 0.5μm.For the n-mos differential pair, the μA/sq= 0.2 for L=0.15μm. Thus W is 0.5μm. For the bias current mirror, we also need it in strong inversion. Thus, we used a NCHCM model.

In a nutshell, the comparator receives a ramp function and an input analog function in which they are continuously compared up to the point where the ramp value intersects the input signal. After that a trigger is initiated that the ramp becomes higher than the input value. This is a form of analog to digital conversion.

B. Digital Part

For the digital part, our aim is to modularize the testbench that we initially received in DICEX [4]. The testbench should only aim to test a module without further manipulations related to the control of the system. Thus, for this purpose, we design a system based on independent modules (state machine and pixel array) and we use a testbench for each one created to verify its working mechanism.

The pixelArray contains a 2x2 pixel sensor in which pixel sensors 1 and 2 supply their data on the higher nibble of the output when needed and the pixel sensors 3 and 4 supply their data on the lower nibble when needed. The control of the system array is found in pixelState which supplies the control signals based on the clock and reset. Figure 5 below better explains the overall digital part.

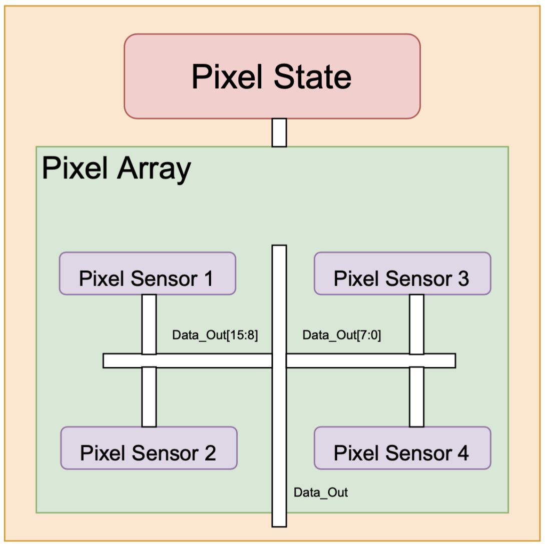


Fig. 5. Digital Schema

Our design choice was to read pixel sensor 1 on Data_Out [15:8] and pixel sensor 3 on Data_Out [7:0] when a READ1 signal is supplied. Moreover, we chose to read pixel sensor 2 on Data_Out [15:8] and pixel sensor 4 on Data_Out [7:0] when a READ2 signal is supplied.

This being said, we only added 2 states to the already present state machine which are READ1 and READ2. This can be better shown in Figure 6.

The state machine starts with ERASE state and proceeds with the states as shown in the arrows. Each state loops in itself for a certain amount of clock rising edges defined by the metric ”c_[name]”. The IDLE state is not added in the figure for reduced complexity. After the completion of each state, they jump into the IDLE state for a short period of time before continuing their path as shown in the state machine.

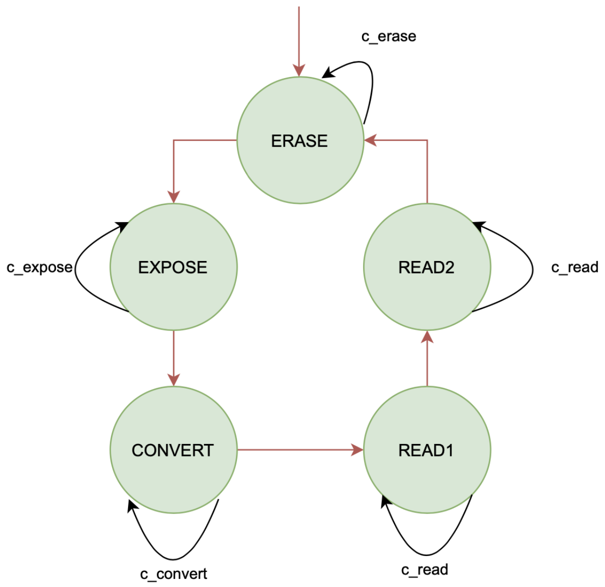


Fig. 6. State Machine

IV. Results

In this section, we will demonstrate the results of our systems in the analog and digital domain.

A. Analog Domain

To start with, we verify that the control signals are working in the right order and quite the right time. In this context, ERASE (pink color) and READ (green color) have a small duration while the EXPOSE (blue color) and CONVERT (orange color) have a relatively longer duration. This can be seen in Figure 7 below.

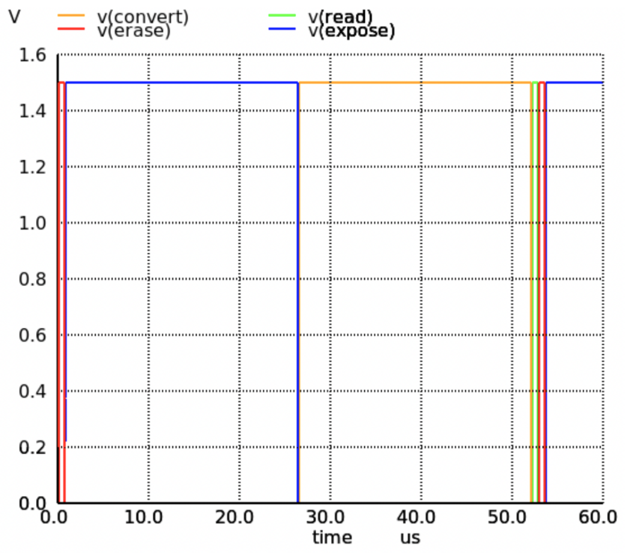


Fig. 7. Control Signals

For the purpose of verifying that the comparator works properly, we use the transient analysis to test its functionality. Figure 8 below shows two graphs that are similar with the only difference that the graph to the right has more pixel exposure time. This results in a lower voltage (vstore - blue color). This should have been higher voltage but can be explained as vstore is an inverted signal. Focusing only on the left graph, we can see that as the ramp function (red color) reaches the level of the input (blue color), the vcmp_out (yellow color) goes low. This proves that the comparator works properly.

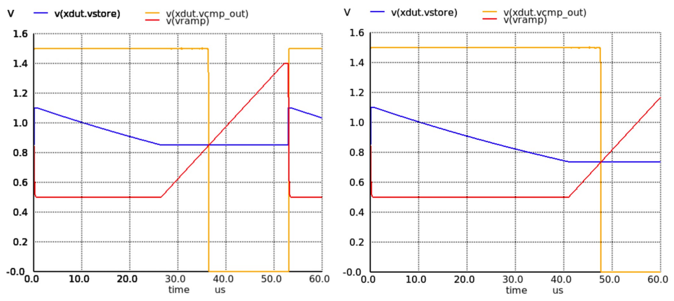
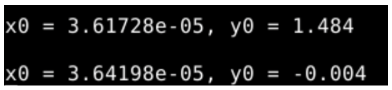


Fig. 8. Comparator Behaviour

For the purpose of checking the delay of the comparator, we probe the signal and subtract the x values which represents time from the initial fall of the vcmp_out to when it actually becomes 0V. Figure 9 shows the values received from terminal. The delay value is then almost 247 ns.



```
x0 = 3.61728e-05, y0 = 1.484

x0 = 3.64198e-05, y0 = -0.004
```

Fig. 9. Terminal Values

B. Digital Part

In the digital part, we created testbenches to verify that several parts of our system works. The first of which is the transition of states based on the time inputted in the program. This is verified through Figure 10 which shows that the states are shifted as shown in the state machine.

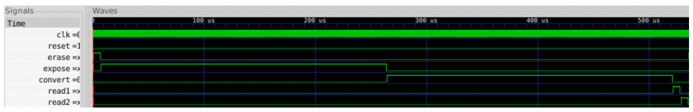


Fig. 10. State Machine Timeline

In the process of conversion, we used the gray code instead of normal upcounting. Thus, we also look at that portion in the waveform to verify it is working properly. This is shown in Figure 11.

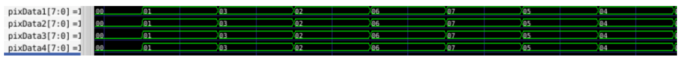


Fig. 11. Gray Code Counter

As it can be seen, the normal sequence of gray code up counting is verified.

The last thing that can be verified from the testbench is the reading of values from the pixel sensors to the 16-bit output in the right manner. Figure 12 shows the 4 pixel sensors' output along with the final 16-bit output.

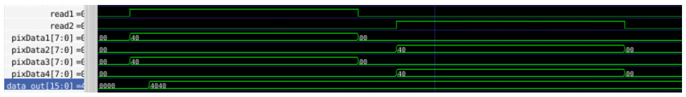


Fig. 12. Reading Out Pixel Values

It can be seen from Fig. 12 that when READ1 is on, then the pixel sensor 1 and pixel sensor 3 are being read, each of which having a value of 0x40. The data_out is simply the concatenation of the two values where pixel sensor 1 is the higher nibble and pixel 3 sensor is the lower nibble. The same applies to pixel sensor 2 and 4 when READ2 signal is on. A final thing to mention here is that our verilog modules are synthesizable

V. Discussion

The results shown in the previous section verifies that the implementation works. The comparator in the analog domain works as required to transform the analog value into a digital one. The time factor here is important because excessive delays could not capture the exact value of the input. Thankfully, our comparator has a delay in the order of magnitude of nanoseconds. As for the digital part, the conversion would have incurred minor glitches, hadn't we used the gray code counter. The gray code counter increases the credibility of the system. Moreover, the reading of two pixel sensors simultaneously into a 16-bit output made the system faster than waiting for the reading of every pixel sensor sequentially. Overall speaking, the system could have encountered minor issues from certain subsystems, but our design helped in making the it avoid such errors.

VI. Future Work

As for the future works, the system can be enhanced using the following methods:

- Implementation of hardware for the proposed system instead of keeping it limited to simulations.
- Addition of a higher number of pixel sensor elements in the pixel sensor array
- Using different kinds of analog to digital convertors such as successive approximation.

VII. Conclusion

In this paper, we introduced an enhanced method of using and reading a 2x2 pixel array by dealing with it from two different aspects, namely from analog and digital perspectives. Our implementation focused on the simulation from SPICE and Verilog. The results proved that our system works in the right manner and achieve their purpose.

VIII. Appendix

This repository can be found using the following link: https://github.com/charbelcbadr/TFE4152-Final-Project.git

The code of the modules can be found here:

A. Analog Part-Spice

```
.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 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 SENSOR VRESET VSTORE ERASE EXPOSE
     VDD VSS

* Capacitor to model gate-source
    capacitance
C1 VSTORE VSS 100f
Rleak VSTORE VSS 100T

* Switch to reset voltage on capacitor
BR1 VRESET VSTORE I=V(ERASE)*V(VRESET,
    VSTORE)/1k

* Switch to expose pixel
BR2 VPG VSTORE I=V(EXPOSE)*V(VSTORE,VPG)
    /1k

* Model photocurrent
Rphoto VPG VSS 1G
.ENDS

.SUBCKT COMP VCMP_OUT VSTORE VRAMP VBN1
    VDD VSS

* Model comparator
M1 VG1 VG1 VDD VDD pmos w=0.5u l=0.5u
M2 VD2 VG1 VDD VDD pmos w=0.5u l=0.5u
M3 VG1 VSTORE VS3 VS3 nmos w=0.5u l=0.15u
M4 VD2 VRAMP VS3 VS3 nmos w=0.5u l=0.15u
XM5 VS3 VBN1 VSS VSS NCHCM2
M6 VD6 VD2 VDD VDD pmos w=0.5u l=0.15u
M7 VD6 VBN1 VSS VSS nmos w=0.5u l=0.15u
M8 VCMP_OUT VD6 VDD VDD pmos w=0.5u l
    =0.15u
M9 VCMP_OUT VD6 VSS VSS nmos w=0.5u l
    =0.15u


.ENDS
```

B. Digital Part - Verilog

```
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;
   parameter real   dv_pixel = 0.5;

   real             tmp;
   logic            cmp;
   real             adc;

   logic [7:0]      p_data;

   //------------------------
   // RESET
   //-------------------------
   // 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;
   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;
      if(adc > tmp)
        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;

endmodule // re_control

module PixelArray (
   input logic       CLK,
   inout logic       VBN1,
   inout logic       RAMP,
   input logic       VRESET,
   input logic       RESET,
   input logic       ERASE,
   input logic       EXPOSE,
   input logic       CONVERT,
   input logic       READ1,
   input logic       READ2,
   output logic [15:0]     DATA_OUT

);
  tri[7:0]          pixData1; //  We need
     this to be a wire, because we're
     tristating it
  tri[7:0]          pixData2; //  We need
     this to be a wire, because we're
     tristating it
  tri[7:0]          pixData3; //  We need
     this to be a wire, because we're
     tristating it
  tri[7:0]          pixData4; //  We need
     this to be a wire, because we're
     tristating it

PIXEL_SENSOR ps1 (VBN1,RAMP,RESET,ERASE,
   EXPOSE,READ1,pixData1);
PIXEL_SENSOR ps2 (VBN1,RAMP,RESET,ERASE,
   EXPOSE,READ2,pixData2);
PIXEL_SENSOR ps3 (VBN1,RAMP,RESET,ERASE,
   EXPOSE,READ1,pixData3);
PIXEL_SENSOR ps4 (VBN1,RAMP,RESET,ERASE,
   EXPOSE,READ2,pixData4);



   integer          counter;
      //Delay counter in state machine
   logic[7:0]       data;


   // If we are to convert, then provide
      a clock via anaRamp
   // This does not model the real world
      behavior, as anaRamp would be a
      voltage from the ADC
   // however, we cheat
   assign RAMP = CONVERT ? CLK : 0;

   // During expoure, provide a clock via
       anaBias1.
   // Again, no resemblence to real world
      , but we cheat.
   assign VBN1 = EXPOSE ? CLK : 0;

   // If we're not reading the pixData,
      then we should drive the bus
   assign pixData1 = READ1 ? 8'bZ: out;
   assign pixData2 = READ2 ? 8'bZ: out;
   assign pixData3 = READ1 ? 8'bZ: out;
   assign pixData4 = READ2 ? 8'bZ: out;

   logic [7:0] out;
   // When convert, then run a analog
      ramp (via anaRamp clock) and digtal
       ramp via
   // data bus. Assert convert_stop to
      return control to main state
      machine.
   always_ff @(posedge CLK or posedge
      RESET) begin
      if(RESET) begin
```

```
         data =0;
      end
      if(CONVERT) begin
         data = data + 1;
      end
      else begin
         data = 0;
      end
      out = { data[7],data[7:1]^data
          [6:0]};
   end // always @ (posedge clk or reset)

   //--------------------------
   // Readout from databus
   //--------------------------

   always_ff @(posedge CLK or posedge
      RESET) begin
      if(RESET) begin
         DATA_OUT = 0;
      end
      else begin
         if(READ1)
         begin
           DATA_OUT[15:8] <= pixData1;
           DATA_OUT[7:0] <= pixData3;
          end
         else if(READ2)
         begin
           DATA_OUT[15:8] <= pixData2;
           DATA_OUT[7:0] <= pixData4;
         end
      end
   end


endmodule

 module pixelState(
 input logic clk,
 input logic reset,
 output logic erase,
 output logic expose,
 output logic convert,
 output logic read1,
 output logic read2
 );

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


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

```
 //State duration in clock cycles
 parameter integer c_erase = 5;
 parameter integer c_expose = 255;
 parameter integer c_convert = 255;
 parameter integer c_read = 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: begin
               if(counter == c_read) begin
                 state <= IDLE;
                 next_state <= READ2;
               end
             end

             READ2: begin
               if(counter == c_read) begin
                 state <= IDLE;
                 next_state <= ERASE;
               end
             end

             IDLE:
               state <= next_state;
           endcase // case (state)
           if(state == IDLE)
             counter = 0;
           else
             counter = counter + 1;
        end
     end // always @ (posedge clk or
        posedge reset)
endmodule
```

```
module pixelTop(
   input logic      CLK,
   inout logic      VBN1,
   inout logic      RAMP,
   input logic      VRESET,
   input logic      RESET,
   inout logic      ERASE,
   inout logic      EXPOSE,
   output logic     CONVERT,
   inout logic      READ1,
   inout logic      READ2,
   output logic [15:0] DATA_OUT);

PixelArray pa (CLK,VBN1,RAMP,VRESET,RESET
     ,ERASE,EXPOSE,CONVERT,READ1,READ2,
  DATA_OUT);

pixelState statemachine(CLK,RESET,ERASE,
    EXPOSE,CONVERT,READ1,READ2);

endmodule
```

References

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

[2] M. Keel et al, "A VGA Indirect Time-of-Flight CMOS Image Sensor With 4-Tap 7- μ m Global-Shutter Pixel and Fixed-Pattern Phase Noise Self-Compensation," IEEE Journal of Solid-State Circuits, vol. 55, (4), pp. 889-897, 2020.

[3] "What is gray code? and advantages of gray code," TutsMaster, 02-Feb-2020. [Online]. Available: https://tutsmaster.org/what-is-gray-code-and-advantages-of-gray-code-2/.

[4] C. Wulffern, "Wulffern/dicex: Exercises for design of Integrated Circuits," GitHub. [Online]. Available: https://github.com/wulffern/dicex.