Digital camera

Title: Design of a digital camera
Group number: 17
Authors: Markus Søvik Gunnarsson and Stian G Hubred

Version: 1.0	Date: November 18, 2019

Abstract

In this report we design a digital camera, with an analog and a digital part. We simulate the design and verify that it works, but find that it has room for improvement.

Contents

1 Introduction

In this report we will take a closer look at digital cameras. A digital camera is a camera that captures photographs and stores it in the digital memory. The first cameras invented were purely analog. Those cameras used chemical processes to record the media. For more than a hundred years, this was the only kind of camera available. Most cameras produced today are digital and many cameras are now being incorporated into mobile devices.

Although they are called digital cameras they are not 100% digital. The reason for this is that the world is analog and the camera need to interact with our world to capture images. The light from the analog world is one of the input signals to the camera. This is an analog signal that varies with time and is converted to a digital signal through the use of an Analog to Digital Converter.

In this report it will be designed some readout electronics and a control circuit for a four pixel digital camera. The design choices for this project is mainly determining suitable geometries for the transistors used in the pixel circuit. Another part of this project is the design choices for the digital implementation for controlling the exposure and readout of new image data and erasing of old image data.[1]

There is as mentioned an analog part($Pixel\_electronics$) and a digital part($Re\_Control$) and they are shown in figure 1.

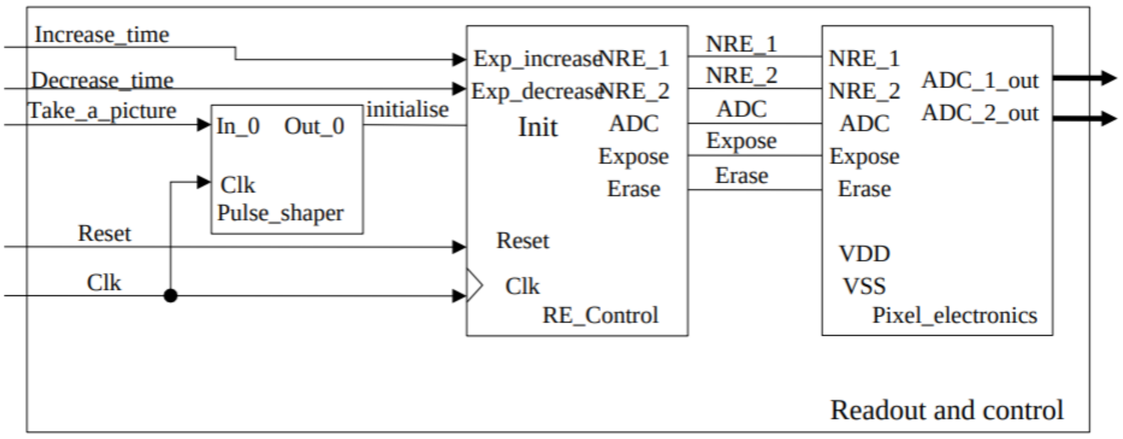


Figure 1: An overview of the whole readout and control-circuit. The picture is taken from [1].

.

Figure 1 shows a block diagram of the digital camera. The block diagram shows the four actions the photographer can use while operating the camera. The pulse-shaping module is not taken into consideration while designing this readout and control-circuit.

2 Analog

The analog part of the camera consists of a 2x2 pixel-array, made up by 4 identical pixel circuits, as illustrated in Figure 2. When a photo is taken, each pixel will generate a voltage proportional to the intensity of the light it is exposed to and the amount of time it is exposed. The resulting voltage in each pixel will then be connected to an Analog-Digital-Converter (ADC), which converts it to a digital value. This process will be controlled by the digital input-signals EXPOSE, ERASE, NRE_1, NRE_2 and ADC.

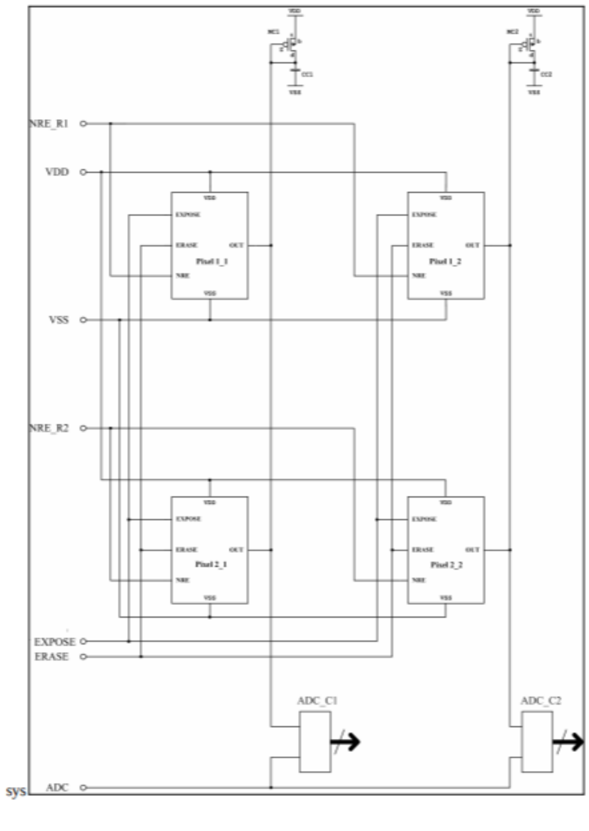


Figure 2: Illustration of the 2x2 analog pixel array.[1]

When the EXPOSE signal is high, the pixels will build up a voltage dependent on the light it is exposed to. When the ERASE signal is high, this voltage will be erased. When the NRE_1 signal is high, the pixels in row 1 of the pixel-array will be disconnected from the ADC. This function is needed because, as illustrated in Figure 2, pixels in the same column will share one ADC, and therefore no more than one pixel must be connected to the ADC at any time. The NRE_2 signal will have the same function for row 2. When the ADC signal is high, the

ADC will be active. In this design, we will not be concerned with designing the ADC.

2.1 Theory

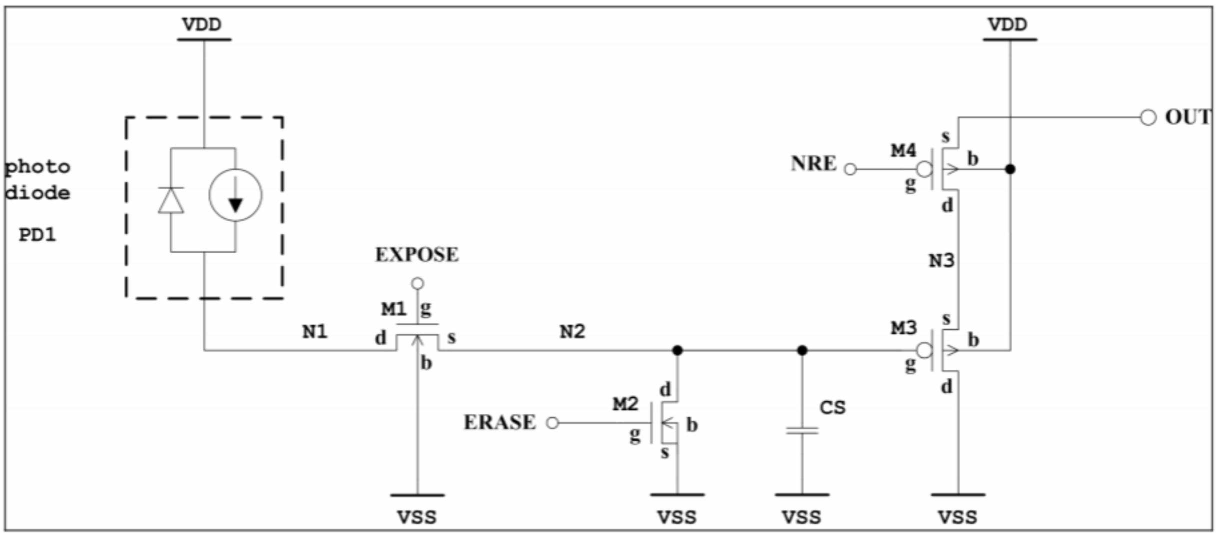


Figure 3: Illustration of the pixel circuit.[1]

The pixel circuit is displayed in Figure 3. When the photo diode PD1 is exposed to light, it will generate a current I_{pd} proportional to the intensity of the light. The NMOS-transistor M1 acts like a switch, and when the EXPOSE signal is high, it will connect PD1 to the capacitor CS. As the current I_{pd} flows into CS, a voltage V_s will build up over the capacitor. The NMOS-transistor M2 also works as a switch, and when the ERASE signal is high, CS will be short-circuited, thereby erasing the voltage V_s over it. Observe that EXPOSE and ERASE should not be high at the same time, as it will short-circuit the current I_{pd}, and thus that if they were both high, the ERASE-function would have priority. When neither EXPOSE nor ERASE is high, the voltage V_s will be preserved.

The PMOS-transistor M3 will act as a buffer between the capacitor CS and the ADC. The PMOS-transistor M4 will act as a switch, and when the NRE signal is low, it will connect the source of M3 to the ADC and a PMOS-transistor MC. When connected, MC will serve as an active load to M3, forming a common-source amplifier of which the output is connected to the ADC. This can be seen in Figure 2, where the capacitors CC represents the parasittic capacitance to the transistors MC.

Now that we have established the topology of the circuit, we will calculate the properties of the individual components, namely the geometries of the transistors and the capacitance of CS.

We know that transistors M1, M2 and M4 will act as switches, thus we want them to be able to switch quickly. This means we'll want a high transconductance g_m. The transconductanse of a transistor is a measure of how sensitive the drain current I_D is to changes in the gate-source voltage V_{GS}. The more sensitive it is, the quicker it will "react". The transconductance is defined as follows, for NMOS- and PMOS-transistors in the linear region respectively:

$$g_m = \frac{\partial I_D}{\partial V_{GS}} = \mu C_{ox} \frac{W}{L} V_{DS} \tag{1}$$

$$g_m = \frac{\partial I_D}{\partial V_{GS}} = \mu C_{ox} \frac{W}{L} V_{SD} \tag{2}$$

We see that for both NMOS- and PMOS-transistors g_m is proportional to $\frac{W}{L}$, thus we will want to maximise this ratio for the transistors M1, M2 and M4.

M3 will act as a voltage buffer, and we want it to have a gain A close to unity. If we derive the expression for A, using the small signal model, we will find that g_m appears in both the numerator and the denominator, thus A approaches unity when g_m approaches infinity. We will once again want to maximise the ratio $\frac{W}{L}$.

M_C will act as an active load. We want the load to have impedance, so that the variations in the voltage V_{out} are sizeable. The impedence is inversely proportional to the transconductance, which we thus want to minise. We want to minimise the ratio $\frac{W}{L}$.

The capacitance C_S will determine how quickly or slowly the capacitor charges up. Let T_E denote the amount of time the pixel is exposed. We can then derive a formula for the resulting voltage V_S using the voltage-current relationship for capacitors:

$$V_S = \frac{1}{C_S} \int_0^{T_E} I_{pd} dt$$

$$V_S = \frac{I_{pd} T_E}{C_S} \tag{3}$$

$$C_S = \frac{I_{pd} T_E}{V_S}$$

Observe from Figure 3 that the voltage V_S can never exceed $V_{Smax} = V_{DD} - V_T$. This means the voltage V_S will range from 0 to V_{Smax}. When generating the photograph we will want to utilise this entire range. At the same time, it is important that the photograph is not overexposed, meaning the voltage maxes out before the exposure is done. Let $I_{pd_{min}}$ denote the expected generated current by the photo-diode in poor light conditions, while $I_{pd_{max}}$ denotes the expected generated current in very bright conditions. The exposure time T_E can be varied between T_{Emin} and T_{Emax}. We will determine the capacitance C_S based on the following formulas:

$$C_S > \frac{I_{pd_{max}} T_{Emin}}{V_{DD} - V_T} \tag{4}$$

$$C_S \approx \frac{I_{pd_{min}} T_{Emax}}{V_{DD} - V_T} \tag{5}$$

By following (4) we make sure that the camera can produce non-overexposed photographs in bright conditions, and by following (5) we make sure we utilise the full voltage range.

2.2 Design

We will implement the design using 180nm transistor technology, and the following specifications:

- For all transistors:
  - 0.360 μm ≤ L ≤ 1.080 μm
  - 1.080 μm ≤ W ≤ 5.040 μm
- For capacitors:
  - $C_S \leq$ 3pF
  - The parasittic capacitance C_C will be modeled as 3pF
- For the current I_{pd}, we assume that:
  - $I_{pd_{min}}$ = 50 pA
  - $I_{pd_{max}}$ = 750 pA
- The exposure time T_E can be varied between:
  - T_{Emin} = 2 ms
  - T_{Emax} = 30 ms

We remember that the transistors M1, M2, M3 and M4 will all have the maximum width W and minimum width L. From the specifications we see L=0.360 μm and W=5.040 μm.

The transistors M_C will have the minimum width W, and the maximum length L: L=W=1.080 μm.

We determine the capacitance C_S by evaluating (4) and (5), and we observe that with the given specifications the right hand sides are equal:

$$\frac{I_{pd_{max}} T_{Emin}}{V_{DD} - V_T} = \frac{I_{pd_{min}} T_{Emax}}{V_{DD} - V_T} \approx 1.12pF$$

We round up rather than down because of the greather-than sign in (4).

2.3 Simulation

We will now simulate various parts of our design, using AIM-Spice. We start by simulating the expose-process, to make sure we have the ideal value of C_S. Netlist in Appendix A. We first simulate with maximum current I_{pd} and minimum exposure time T_E, to test for overexposure. The result is shown in Figure 4.

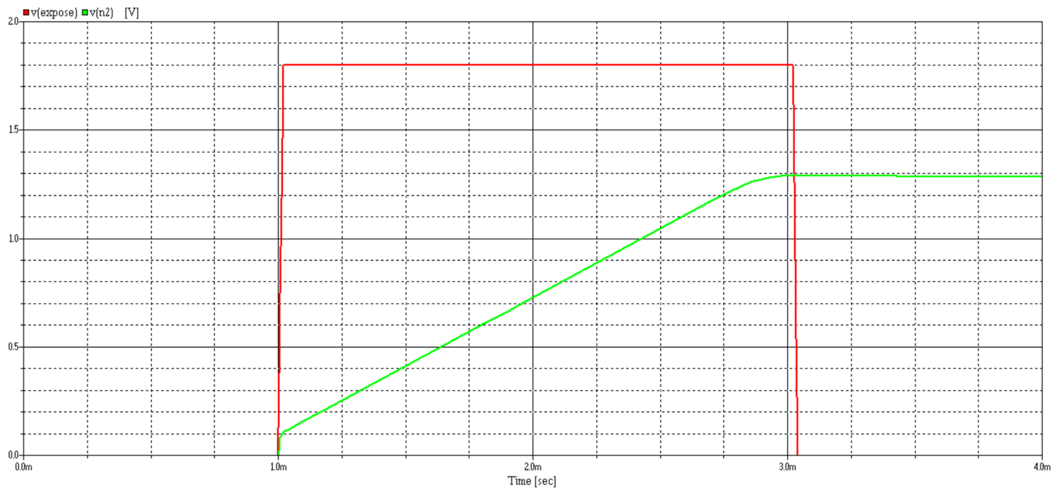


Figure 4: Simulation of the analog expose-process, with $I_{pd} = 750pA$, $T_E = 2ms$ and $C_S = 1.12pF$. v(exposure) is the EXPOSE-signal, v(n2) is the voltage V_S.

We see that the voltage V_S reaches a slight plateau towards the end of the exposure, as it closes in on 1.3V. We recalculate with 1.3V as the denominator in (4), and try again with $C_S = 1.15pF$. The result is shown in Figure 5.

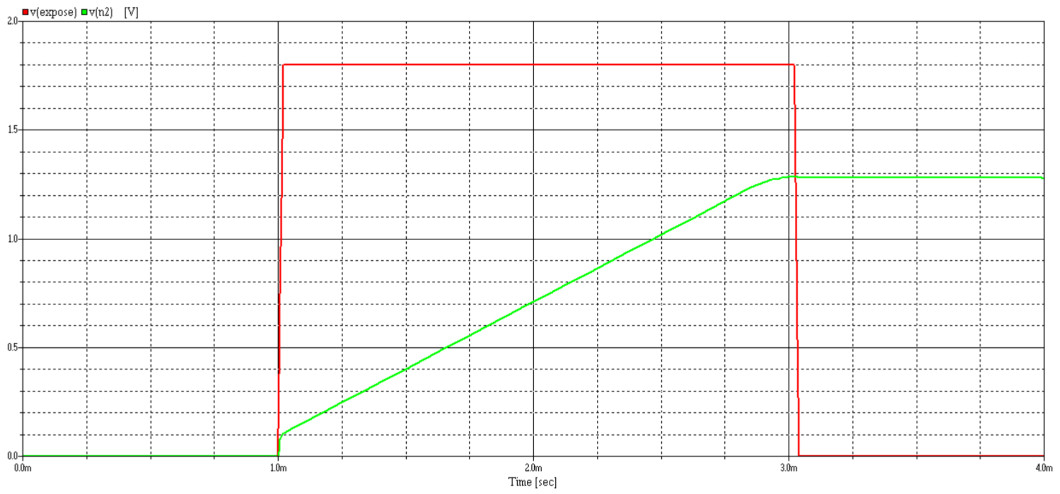


Figure 5: Simulation of the analog expose-process, with $I_{pd} = 750pA$, $T_E = 2ms$ and $C_S = 1.15pF$. v(exposure) is the EXPOSE-signal, v(n2) is the voltage V_S.

V_S now appears to be linear, though it was not far off in the first place. We now simulate with minimum current I_{pd} and maximum exposure time T_E. The result is shown in Figure 6.

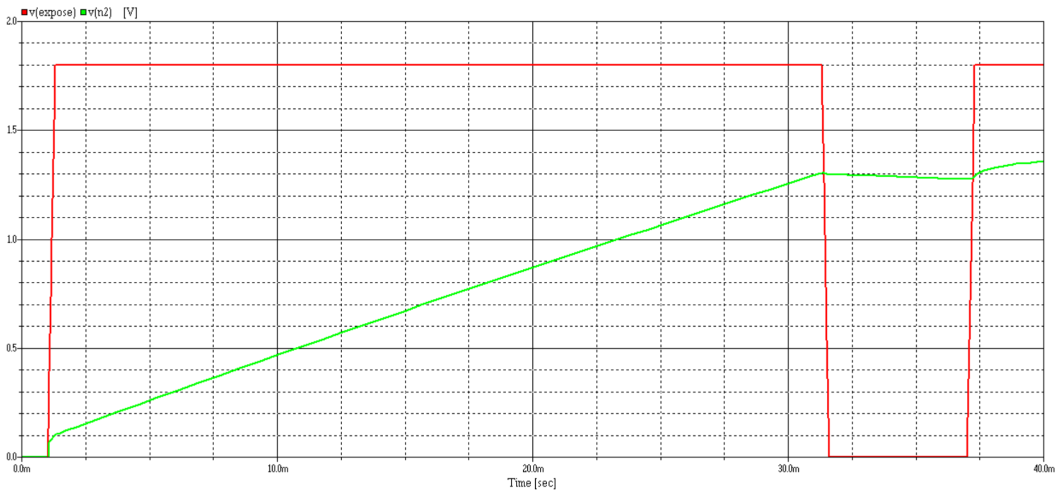


Figure 6: Simulation of the analog expose-process, with $I_{pd} = 50pA$, $T_E = 30ms$ and $C_S = 1.15pF$. v(exposure) is the EXPOSE-signal, v(n2) is the voltage V_S.

We see that V_S appears linear until the end, while also making use of its full range. This is ideal.

We now move on to the output part of the pixel circuit, and simulate the voltage over the ADC for varying values of V_S, with the M4-transistor open. Netlist in Appendix B. The result is shown in Figure 7.

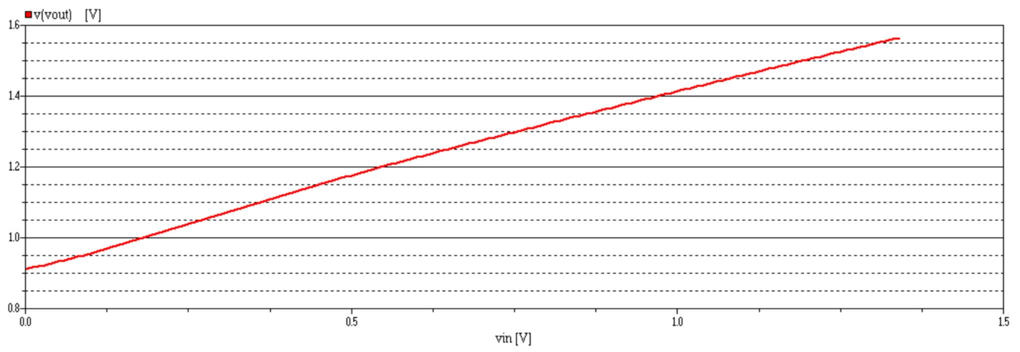


Figure 7: Simulation of the voltage over the ADC with varying values of the voltage V_C, with maximum $(\frac{W}{L})$-ratio for the transistor M3, and minimum ratio for the transistor M_C.

We see that V_{out} ranges from 0.9V to 1.6V when V_C ranges from 0V to 1.3V. This is not ideal, as a narrower voltage range will give a lower resolution in the converted signal from the ADC. Further simulations do however confirm the we have chosen the optimal $(\frac{W}{L})$-ratios for both M3 and M_C. This is a part of the circuit that can be improved upon.

We now simulate the entire functionality of one pixel circuit, with simulated digital input-signals. Netlist in Appendix C. Result shown in Figure 8.

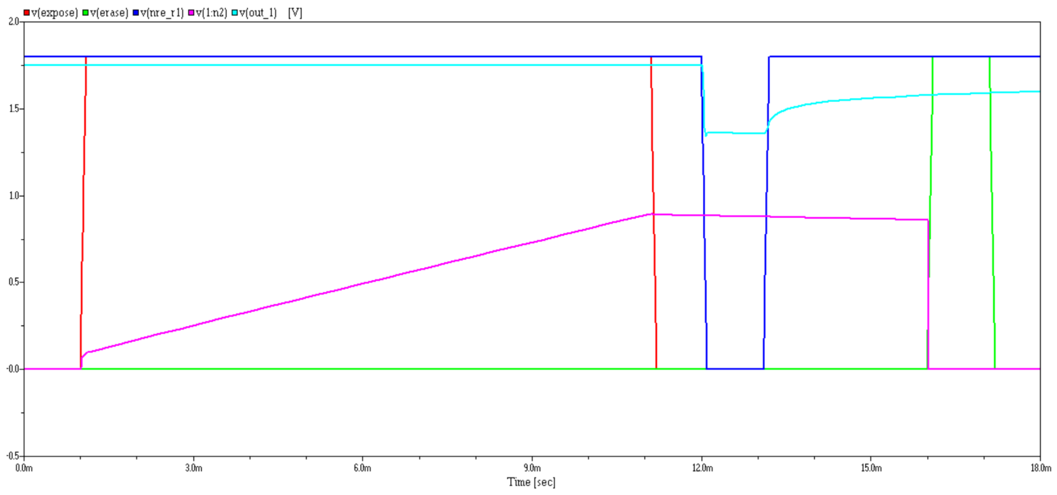


Figure 8: Simulation of one pixel. $I_{pd} = 100pA$, $T_E = 10ms$.

We see that the pixel works mostly as expected. The voltage V_S charges up as expected, and the ERASE-signal erases it as expected. We see that V_{out} connects to the pixel when the NRE-signal is low, however we also see the effect of the parasittic capacitance C_C, as V_{out} appears to need a substantial amount of time to recover.

We will now simulate the full pixel array. In order to test how V_{out} switches from one pixel to another, we will apply stronger current to the first pixel in the first column, and the second pixel in the second column, while applying a weaker current to the second pixel in the first column and the first pixel in the second column. Netlist in appendix D. The result is shown in Figure 9.

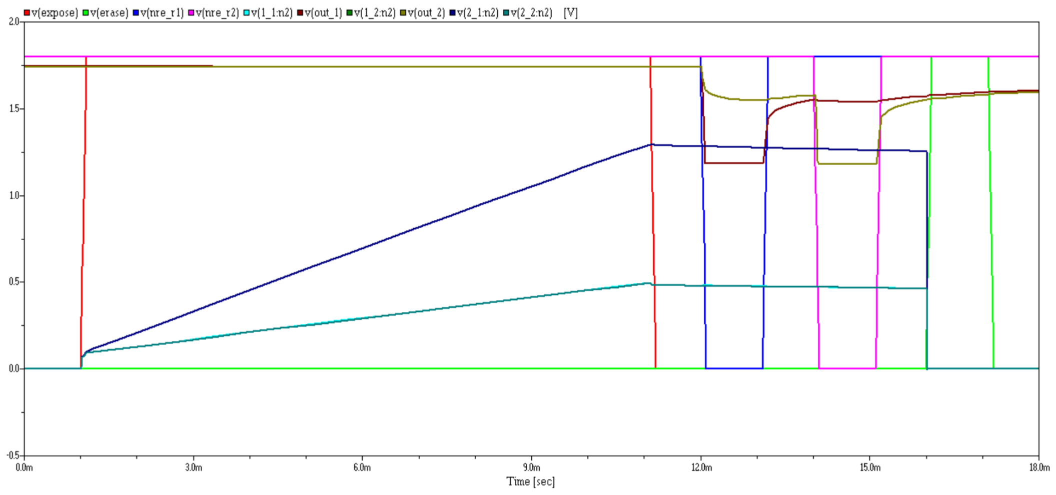


Figure 9: Simulation of the entire pixel array. Observe that the pixels have pairwise identical voltages. Two pixels were exposed to a current $I_{pd} = 50pA$, the other two were exposed to $I_{pd} = 150pA$, and the exposure time was $T_E = 10ms$.

The pixel array seems to be working mostly as expected. We see that the voltages builds up and are erased as expected. We see that the V_{out}'s are connected to the pixel at the right times. Again, we also see the effect of the parasittic capacitances, as the V_{out}-voltages need some time getting back up. This is another part of the circuit that could be improved upon. It does seem like we are getting fairly consistent readings, though, but the second high reading is lower than the first one. They are supposed to be equal.

3 Digital

3.1 Theory

As described in the introduction the readout control is purely digital. It sends digital signals to the transistors in the pixel electronic. The readout control needs to have 4 inputs as well as an input for a clock. The inputs is based on the photographer actions. Those actions is explained in the list below:

- Taking a picture - starting the exposure time and the readout control.
- Reset - resetting the whole process no matter what state the readout control is in.
- Increasing time - increasing the exposure time.
- Decreasing time - decreasing the exposure time.

The 5 outputs of the readout control should behave as described in the list below:

- NRE_1 - Signal will control which pixel is read. Can never be active the same time as NRE_2.
- NRE_2 - Signal will control which pixel is read. Can never be active the same time as NRE_1.
- ADC - Signal will control if the ADC should sample the analog value of the capacitor into digital value.
- Expose - Signal will control how long the capacitor will be charged.
- Erase - Signal will control if the value of the charged capacitor will be erased.

A finite-state machine (FSM) is a mathematical model of computation. A FSM is a machine that is abstract and has only a finite number of state. The FSM can transition from one state to another in response to inputs and/or different conditions. It has a list of states, an initial state, and conditions for each transition. It is possible to divide such state machines into two categories, Moore and Mealy machines. Moore machines output depends only on the state, while Mealy machines depends on both states and inputs. It is also possible to divide the FSM into deterministic and non-deterministic machines. In a deterministic machine, every state has exactly one transition for each possible input. In a non-deterministic machine, an input can lead to one, more than one, or no transition for a given state.[2]

Verilog is a hardware description language used to model electronic systems. Verilog can in a sense similar to software programming languages because it has ways to describe the propagation time and signal strengh. The syntax for Verilog were purposely made similar to C. Verilog uses two types of assignment operators, a blocking assignment (=) and a non-blocking (<=) assignment. The non-blocking assignment allows a state-machine to update without needing to declare and use temporary storage variables. A Verilog design often consists of a hierarchy of modules with inputs and outputs. Some of the Verilog language are synthesizable, which means that it can be physically realized by synthesis software.[3]

3.2 Design

The readout control is implemented as a top-level module with the hardware descriptive language Verilog, and is called Main. Main has two sub-modules, a counter called expcounter and a finite-state machine called readout. An overview block diagram is shown below in figure 10.

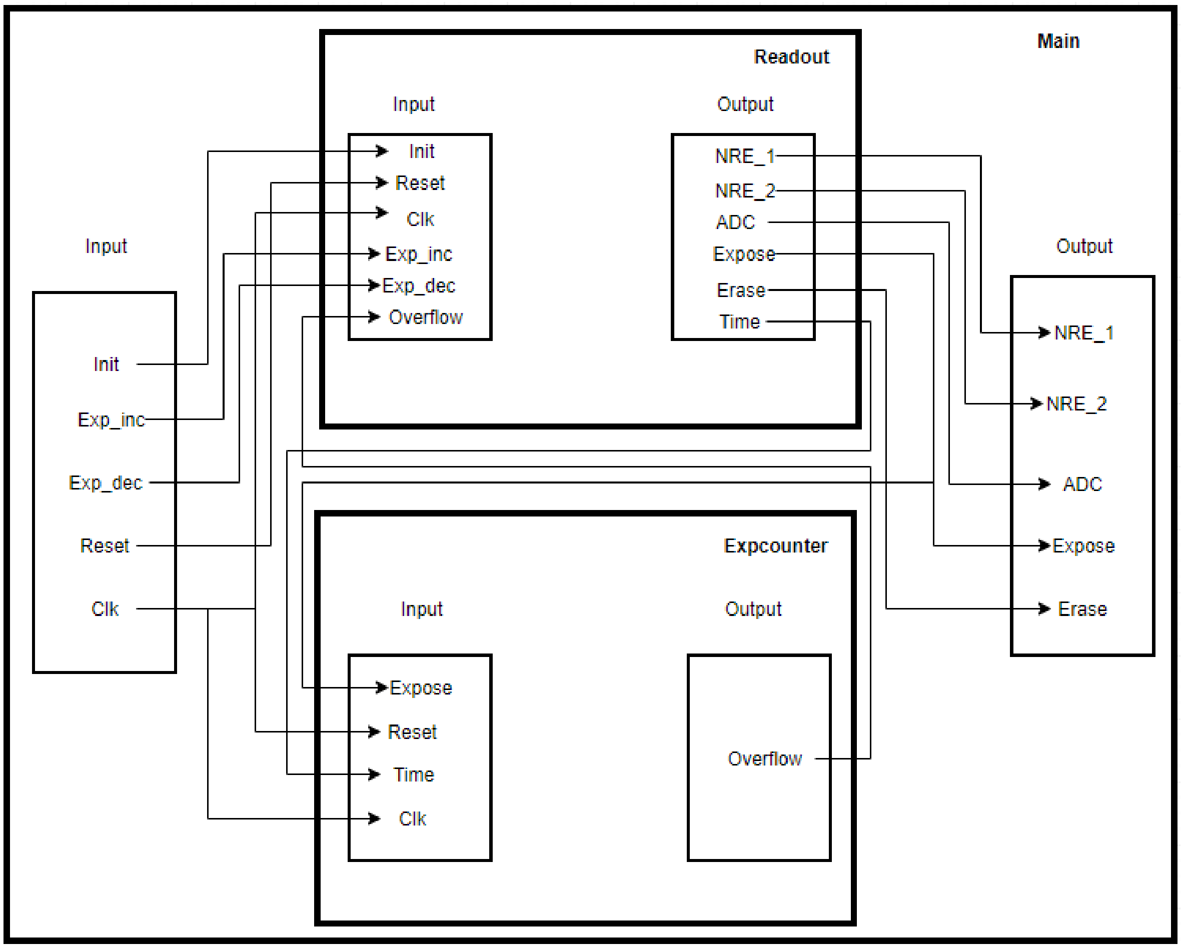
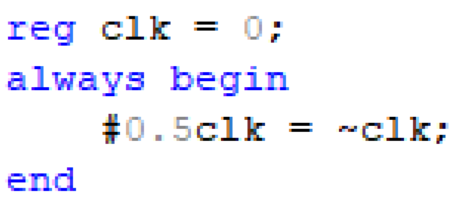


Figure 10: An overview of the top-level module Main and the two sub-modules Readout and Expcounter

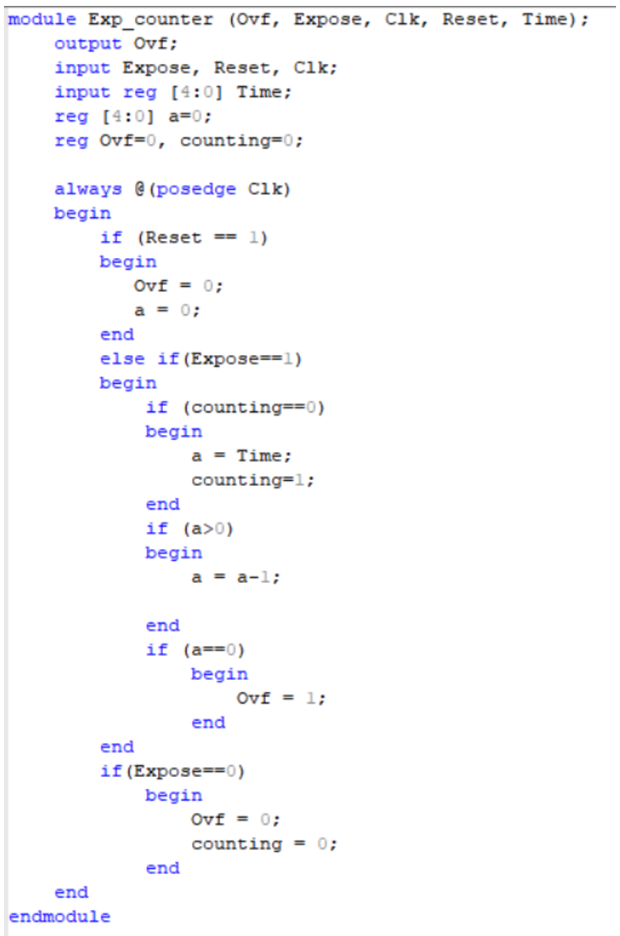
As shown in figure 10 all modules have their inputs and outputs. One of the inputs is the input Clk, which is the input for the built in clock. All the modules uses the same clock signal, which in this case is specified to have a frequency of 1000 Hz (1ms clock period). This is implemented by adjusting the timescale to 1ms/1ns and the following verilog code (figure 11).



```
reg clk = 0;
always begin
    #0.5clk = ~clk;
end
```

Figure 11: A small code snippet that shows the clock implementation

One of the modules that uses the Clk signal as implemented in figure 11 is Expcounter. Expcounter takes in an exposure time [Time], which is set by the user in Readout. When it gets the start signal [Expose], it starts counting down. Eventually Expcounter is done counting the correct amount of time and sends a signal to Readout [Overflow] to signal that it is finished counting. The code snippet shown in figure 12.



```
module Exp_counter (Ovf, Expose, Clk, Reset, Time);
    output Ovf;
    input Expose, Reset, Clk;
    input reg [4:0] Time;
    reg [4:0] a=0;
    reg Ovf=0, counting=0;

    always @(posedge Clk)
    begin
        if (Reset == 1)
        begin
           Ovf = 0;
           a = 0;
        end
        else if(Expose==1)
        begin
            if (counting==0)
            begin
                a = Time;
                counting=1;
            end
            if (a>0)
            begin
                a = a-1;

            end
            if (a==0)
                begin
                    Ovf = 1;
                end
        end
        if(Expose==0)
            begin
                Ovf = 0;
                counting = 0;
            end
    end
endmodule
```

Figure 12: A code snippet that shows the Expcounter implementation

From the code snippet in figure 12 it can be seen that the register "a" is only changes at positive clock edges (using the verilog syntax "posedege"), this way it will count using 1ms increment. It can also be seen from figure 12 that the input signal [Reset] have the highest priority. This implementation is a common feature throughout the design.

The design of the module Readout is a bit more complex than the counter Expcounter. As mentioned earlier Readout is a FSM. The 11 different states is shown in figure 13.

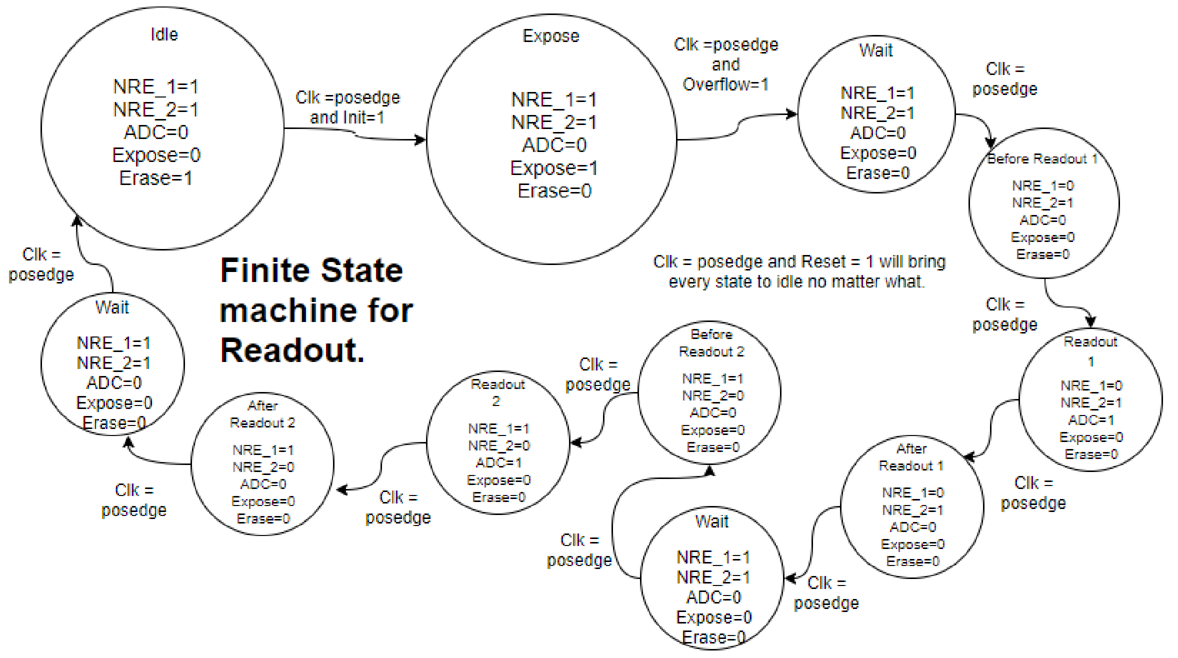


Figure 13: A state diagram for the FSM in Readout

Figure 13 displays the outputs [NRE_1], [NRE_2], [ADC], [Expose] and [Erase] for the 11 different states. It is important to notice that some states have similar outputs. Those states is made into different states to help the state machine knowing where in the readout process it currently is and what the next state should be like. The default state is Idle, so if the state machine get an illegal state it will put the state machine back to the Idle-state. This is also the case if Reset = 1. As mentioned earlier Reset has the highest priority of the input signals. This state machine will give the following timing chart (14). Note! (The exposure time here is 8ms).

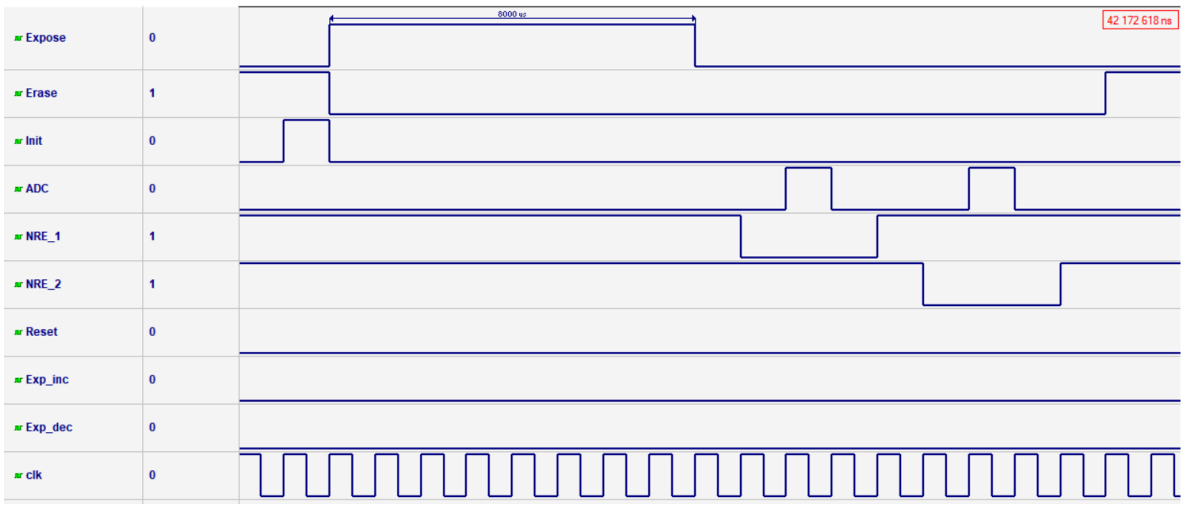
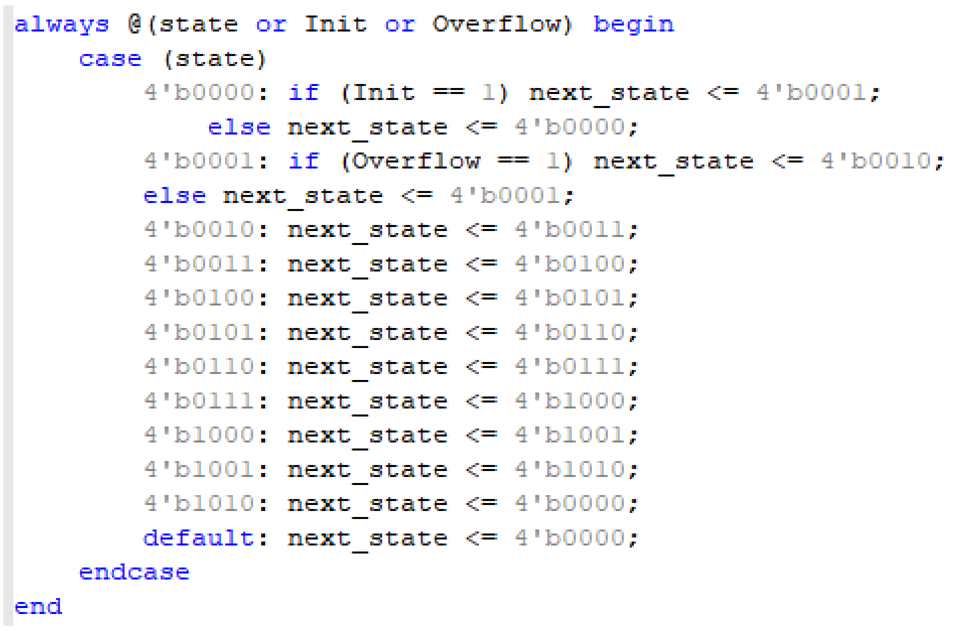


Figure 14: The timing chart for a normal cyclus for the statemachine. Note!(The exposure time here is 8ms)

The timing chart in figure 14 shows that when Init = 1 the state machine exits the idle state and start cycles trough states. All states except Idle, have a total time period of the exposure time + 9 ms (in this case: 8ms + 9ms = 17ms).

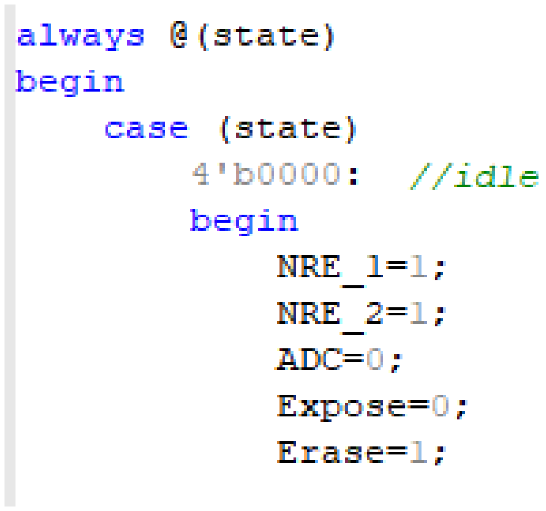
The verilog code that forfills the state diagram in figure 13 and the timing chart in figure 14 is a fragment of the verilog module Readout. A snippet of this code is displayed in figure 15



```
always @(state or Init or Overflow) begin
    case (state)
        4'b0000: if (Init == 1) next_state <= 4'b0001;
            else next_state <= 4'b0000;
        4'b0001: if (Overflow == 1) next_state <= 4'b0010;
        else next_state <= 4'b0001;
        4'b0010: next_state <= 4'b0011;
        4'b0011: next_state <= 4'b0100;
        4'b0100: next_state <= 4'b0101;
        4'b0101: next_state <= 4'b0110;
        4'b0110: next_state <= 4'b0111;
        4'b0111: next_state <= 4'b1000;
        4'b1000: next_state <= 4'b1001;
        4'b1001: next_state <= 4'b1010;
        4'b1010: next_state <= 4'b0000;
        default: next_state <= 4'b0000;
    endcase
end
```

Figure 15: Snippet of Readout, where the implementation of the state machine is.

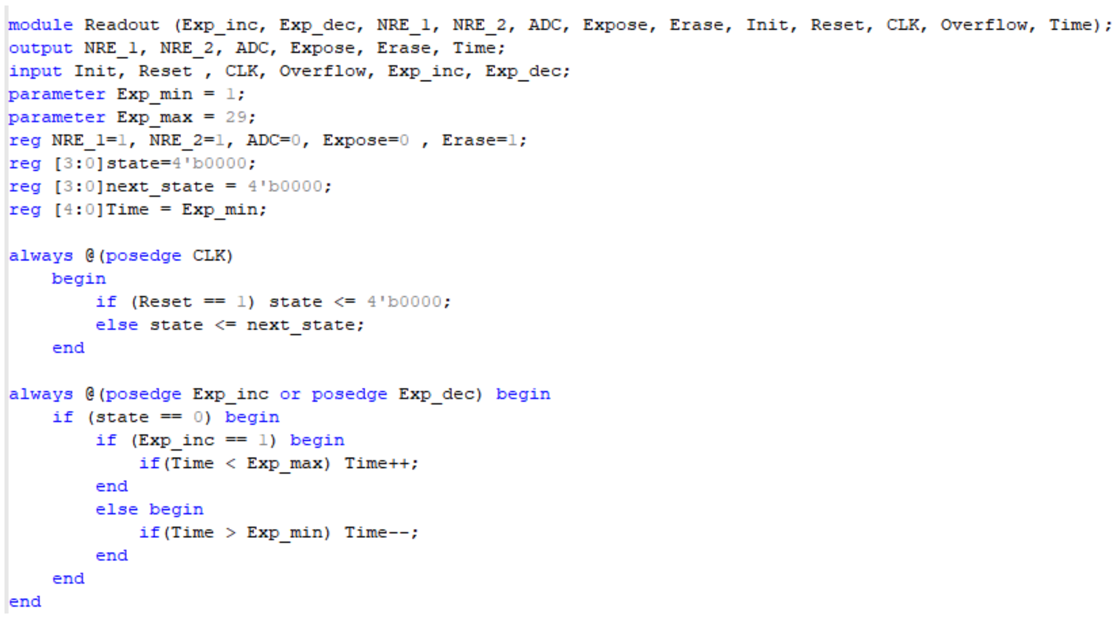
The way the states in figure 15 is configured is shown in the small snippet in figure 16.



```
always @(state)
begin
    case (state)
        4'b0000:  //idle
        begin
            NRE_1=1;
            NRE_2=1;
            ADC=0;
            Expose=0;
            Erase=1;
```

Figure 16: Snippet of Readout, where the implementation of each case is. Here is the idle state.

Even though figure 16 says state Idle, the rest of the states is implemented the same way. Another essential part of the Readout module is the incrementation/decrimentation of the exposure time. This is implemented as shown in the code snippet in figure 17.



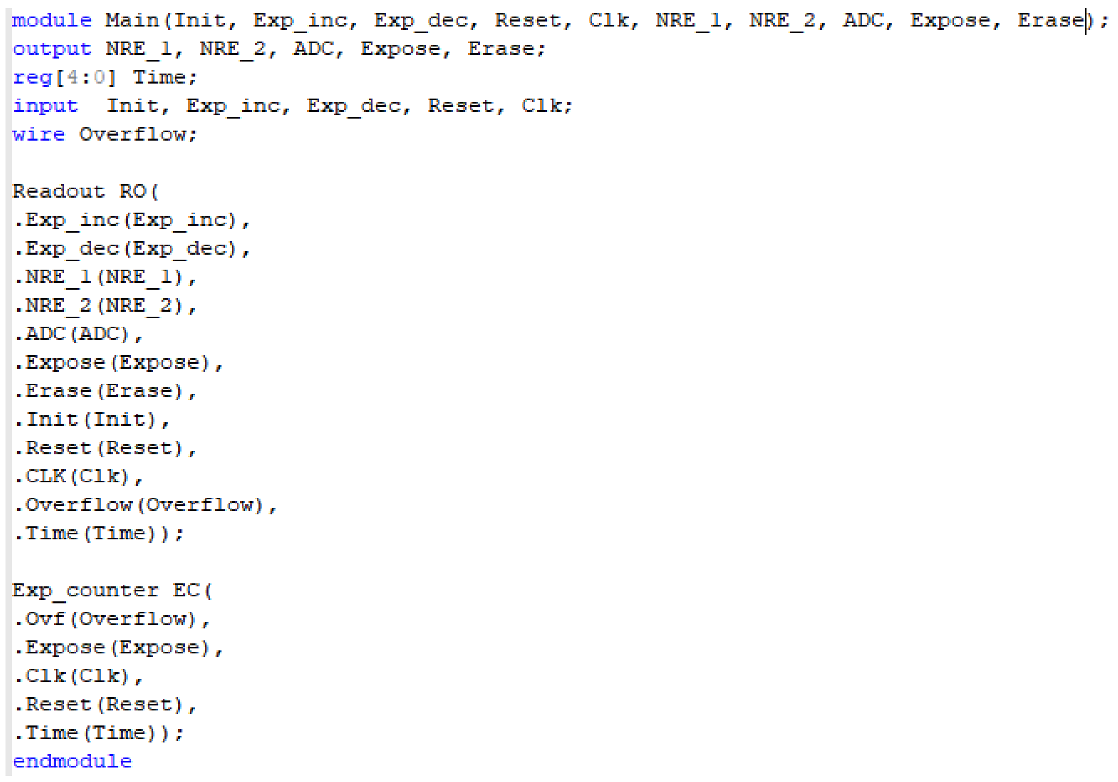
```
module Readout (Exp_inc, Exp_dec, NRE_1, NRE_2, ADC, Expose, Erase, Init, Reset, CLK, Overflow, Time);
output NRE_1, NRE_2, ADC, Expose, Erase, Time;
input Init, Reset , CLK, Overflow, Exp_inc, Exp_dec;
parameter Exp_min = 1;
parameter Exp_max = 29;
reg NRE_1=1, NRE_2=1, ADC=0, Expose=0 , Erase=1;
reg [3:0]state=4'b0000;
reg [3:0]next_state = 4'b0000;
reg [4:0]Time = Exp_min;

always @(posedge CLK)
    begin
        if (Reset == 1) state <= 4'b0000;
        else state <= next_state;
    end

always @(posedge Exp_inc or posedge Exp_dec) begin
    if (state == 0) begin
        if (Exp_inc == 1) begin
            if(Time < Exp_max) Time++;
        end
        else begin
            if(Time > Exp_min) Time--;
        end
    end
end
```

Figure 17: Snippet of Readout, where the implementation of incrementation/decrimentation of the exposure time is.

First of all, from figure 17, is is clear that once again Reset has the highest priority. With regards to priority it is also clear that Exp_inc has a higher priority than Exp_dec. This is stated as a specification for the project. With the verilog syntax parameter the minimum exposure time is set to 2ms by setting Exp_min to 1 and the maximum exposure time is set to 30 ms by setting Exp_max to 29.

One particular design choice that is made against the project specification is the implementation for Exp_inc and Exp_dec. It it asked by the specification that the value should update if those particular inputs were high for each clock period. That would have maxed the exposure time within 28 ms. If a photographer should use the camera he would have to have inhumane timing to chose his desired exposure time. The design chosen is to use the verilog syntax posedge to only update Exp_inc and Exp_dec during positive edges of the signal. This way the photographer has to release the button in order to increment/decriment the time further.

The two sub-level modules Expcounter and Readout is initiated in the top-level module Main as shown in figure 18.



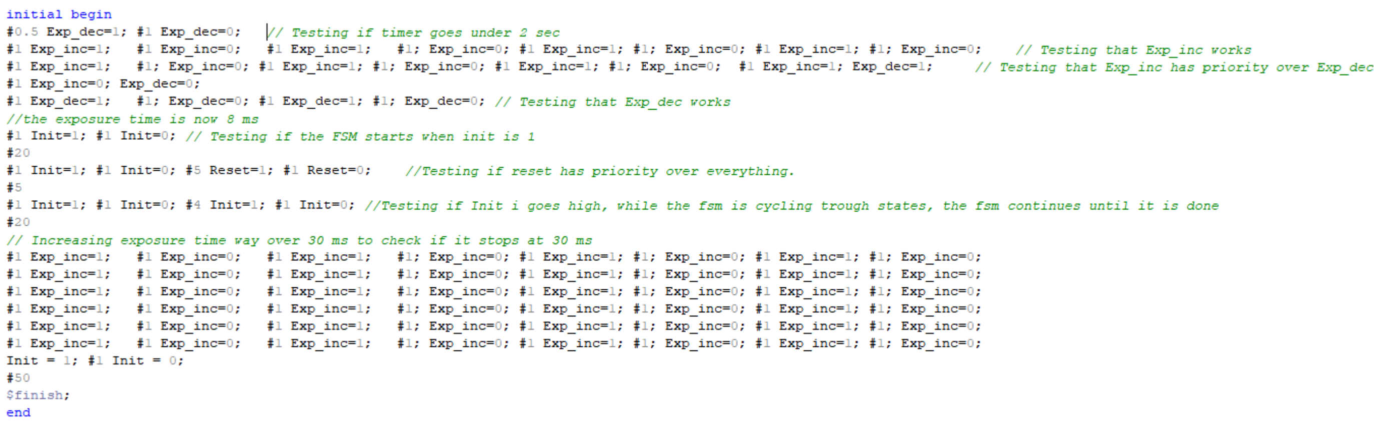
```
module Main(Init, Exp_inc, Exp_dec, Reset, Clk, NRE_1, NRE_2, ADC, Expose, Erase);
output NRE_1, NRE_2, ADC, Expose, Erase;
reg[4:0] Time;
input  Init, Exp_inc, Exp_dec, Reset, Clk;
wire Overflow;

Readout RO(
.Exp_inc(Exp_inc),
.Exp_dec(Exp_dec),
.NRE_1(NRE_1),
.NRE_2(NRE_2),
.ADC(ADC),
.Expose(Expose),
.Erase(Erase),
.Init(Init),
.Reset(Reset),
.CLK(Clk),
.Overflow(Overflow),
.Time(Time));

Exp_counter EC(
.Ovf(Overflow),
.Expose(Expose),
.Clk(Clk),
.Reset(Reset),
.Time(Time));
endmodule
```

Figure 18: Snippet of Readout, where the implementation of incrementation/decrimentation of the exposure time is.

Figure 18 shows how the top-level module is implemented using verilog.

3.3 Simulation

The top-level module main is tested by building a test bench and introducing it to different input. Such a test bench is displayed with the snippet of verilog code in figure 19.



```
initial begin
#0.5 Exp_dec=1; #1 Exp_dec=0;   // Testing if timer goes under 2 sec
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;    // Testing that Exp_inc works
#1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;  #1 Exp_inc=1; Exp_dec=1;     // Testing that Exp_inc has priority over Exp_dec
#1 Exp_inc=0; Exp_dec=0;
#1 Exp_dec=1;   #1; Exp_dec=0; #1 Exp_dec=1; #1; Exp_dec=0; // Testing that Exp_dec works
//the exposure time is now 8 ms
#1 Init=1; #1 Init=0; // Testing if the FSM starts when init is 1
#20
#1 Init=1; #1 Init=0; #5 Reset=1; #1 Reset=0;    //Testing if reset has priority over everything.
#5
#1 Init=1; #1 Init=0; #4 Init=1; #1 Init=0; //Testing if Init i goes high, while the fsm is cycling trough states, the fsm continues until it is done
#20
// Increasing exposure time way over 30 ms to check if it stops at 30 ms
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
#1 Exp_inc=1;   #1 Exp_inc=0;   #1 Exp_inc=1;   #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
Init = 1; #1 Init = 0;
#50
$finish;
end
```

Figure 19: Snippet of the test bench for main.

Figure 19 shows the implementation in verilog code for the different inputs for main. Specific scenarios that are being tested are:

- If exposure time goes under 2 ms
- If Exp_inc works
- If Exp_inc has priority over Exp_dec
- If Exp_dec works
- If the FSM starts when Init = 1
- If Reset has priority over everything else
- If Init goes high, while the FSM is cycling trough state, the FSM continues untill it is done
- If exposure time goes over 30 ms

All scenarios worked as expected.

4 Conclusion

We have designed a functional camera, by designing and combining an analog pixel array, and a digital finite state machine.

In the analog part of the design, we used theory to designed the circuit, and found optimal values for each component. We then simulated the circuit in parts, and as a whole, and made one adjustment - to the capacitor C_S. We verified that the circuit worked as expected, but found that the design had room for improvement in two specific areas:

- The output voltage V_{out} from the pixels had a narrow range, which may cause worse than expected resolution in the converted digital signal.
- The parasittic capacitance of the active load transistor M_C interfered noticeably with the V_{out}-signal

The digital readout and expose control worked as expected. Some experience with computer aided design tools such as Active HDl and Aim-Spice were improved.

References

[1] Bjørn B Larsen , 2019 *TFE4152 – Design of integrated circuits 2019 Project description - Digital camera*, NTNU

[2] Wikipedia , 17.11.2019 *Finite-state machine, https://en.wikipedia.org/wiki/Finite-state_machine*

[3] Wikipedia , 17.11.2019 *Verilog, https://en.wikipedia.org/wiki/Verilog*

5 Appendix A

Netlist for expose-simulation:

```
Expose Circuit
.param EXPOSURETIME = 2m ! Exposure time, range [2 ms, 30 ms]
.param TRF = {EXPOSURETIME/100} ! Risetime and falltime of EXPOSURE and ERASE
   ↪ signals
.param PW = {EXPOSURETIME} ! Pulsewidth of EXPOSURE and ERASE signals
.param FS = 1k ! Sampling clock frequency
.param CLK_PERIOD = {1/FS} ! Sampling clock period
.param PERIOD = {EXPOSURETIME + 6*CLK_PERIOD} ! Period for testbench sources
.param EXPOSE_DLY = {CLK_PERIOD} ! Delay for EXPOSE signal
.param VDD = 1.8
.param L = 0.36u
.param W = 5.04u
.param cs = 1.15p
.param Ipd_1 = 750p

VEXPOSE EXPOSE 0 dc 0 pulse(0 VDD EXPOSE_DLY TRF TRF EXPOSURETIME PERIOD)
```

```
VCD 1 0 dc VDD
x1 1 N1 PhotoDiode Ipd_1 = Ipd_1
mn1 N1 EXPOSE N2 0 NMOS W=W L=L
c1 N2 0 cs

.subckt PhotoDiode VDD N1_R1C1
I1_R1C1 VDD N1_R1C1 DC Ipd_1
d1 N1_R1C1 vdd dwell 1
.model dwell d cj0=1e-14 is=1e-12 m=0.5 bv=40
Cd1 N1_R1C1 VDD 30f
.ends

.include p18_cmos_models.inc
.include p18_model_card.inc
```

6 Appendix B

Netlist for simulating the output-part of one pixel:

```
* Buffer + Active load
.param VDD = 1.8
.param L3 = 0.36u
.param W3 = 5.04u
.param LC = 1.08u
.param WC = 1.08u
.param L4 = 0.36u
.param W4 = 5.04u
.param cc = 3p

VDD 1 0 dc VDD
VIN V2 0 dc
VNRE NRE 0 dc 0
* DRAIN - GATE - SOURCE - BULK
mp3 0 V2 N3 1 PMOS W=W3 L=L3
mp4 N3 NRE VOUT 1 PMOS W=W4 L=L4
mpc1 VOUT VOUT 1 1 PMOS W=WC L=LC
cc1 VOUT 0 cc

.include p18_cmos_models.inc
.include p18_model_card.inc
```

7 Appendix C

Netlist for simulating one pixel:

```
*One Pixel

* Ipd_1 = Photodiode current, range [50 pA, 750 pA]
.param VDD = 1.8 ! Supply voltage
.param EXPOSURETIME = 10m ! Exposure time, range [2 ms, 30 ms]

.param TRF = {EXPOSURETIME/100} ! Risetime and falltime of EXPOSURE and ERASE
   ↪ signals
.param PW = {EXPOSURETIME} ! Pulsewidth of EXPOSURE and ERASE signals
.param PERIOD = {EXPOSURETIME*10} ! Period for testbench sources
.param FS = 1k; ! Sampling clock frequency
.param CLK_PERIOD = {1/FS} ! Sampling clock period
.param EXPOSE_DLY = {CLK_PERIOD} ! Delay for EXPOSE signal
.param NRE_R1_DLY = {2*CLK_PERIOD + EXPOSURETIME} ! Delay for NRE_R1 signal
.param ERASE_DLY = {6*CLK_PERIOD + EXPOSURETIME} ! Delay for ERASE signal

.param LS = 0.36u
.param WS = 5.04u
.param LB = 0.36u
.param WB = 5.04u
.param LC = 1.08u
.param WC = 1.08u
.param cs = 1.15p

VDD 1 0 dc VDD
VEXPOSE EXPOSE 0 dc 0 pulse(0 VDD EXPOSE_DLY TRF TRF EXPOSURETIME PERIOD)
VERASE ERASE 0 dc 0 pulse(0 VDD ERASE_DLY TRF TRF CLK_PERIOD PERIOD)
VNRE_R1 NRE_R1 0 dc 0 pulse(VDD 0 NRE_R1_DLY TRF TRF CLK_PERIOD PERIOD)
*VNRE_R2 NRE_R2 0 dc 0 pulse(VDD 0 NRE_R2_DLY TRF TRF CLK_PERIOD PERIOD)

x1 1 EXPOSE ERASE NRE_R1 OUT_1 PIXEL Ipd_1=100p

xc1 1 OUT_1 ACTIVE_LOAD

.subckt ACTIVE_LOAD vdd OUT
mpc OUT OUT vdd vdd PMOS W=WC L=LC
cc OUT 0 3p
.ends

.subckt PIXEL vdd EXPOSE ERASE NRE OUT
x1 vdd N1 PhotoDiode Ipd_1=Ipd_1
```

```
* DRAIN - GATE - SOURCE - BULK - type - W=w - L=w
mn1 N1 EXPOSE N2 0 NMOS W=WS L=LS
mn2 N2 ERASE 0 0 NMOS W=WS L=LS
c1 N2 0 cs
mp3 0 N2 N3 vdd PMOS W=WB L=LB
mp4 N3 NRE OUT vdd PMOS W=WS L=LS
.ends

.subckt PhotoDiode VDD N1_R1C1
I1_R1C1 VDD N1_R1C1 DC Ipd_1
d1 N1_R1C1 vdd dwell 1
.model dwell d cj0=1e-14 is=1e-12 m=0.5 bv=40
Cd1 N1_R1C1 VDD 30f
.ends

.include P18_cmos_models.inc
.include P18_model_card.inc
```

8 Appendix D

Netlist for simulation of the entire pixel array:

```
* Full analog

* Ipd_1 = Photodiode current, range [50 pA, 750 pA]
.param VDD = 1.8 ! Supply voltage
.param EXPOSURETIME = 10m ! Exposure time, range [2 ms, 30 ms]

.param TRF = {EXPOSURETIME/100} ! Risetime and falltime of EXPOSURE and ERASE
   ↪ signals
.param PW = {EXPOSURETIME} ! Pulsewidth of EXPOSURE and ERASE signals
.param PERIOD = {EXPOSURETIME*10} ! Period for testbench sources
.param FS = 1k; ! Sampling clock frequency
.param CLK_PERIOD = {1/FS} ! Sampling clock period
.param EXPOSE_DLY = {CLK_PERIOD} ! Delay for EXPOSE signal
.param NRE_R1_DLY = {2*CLK_PERIOD + EXPOSURETIME} ! Delay for NRE_R1 signal
.param NRE_R2_DLY = {4*CLK_PERIOD + EXPOSURETIME} ! Delay for NRE_R2 signal
.param ERASE_DLY = {6*CLK_PERIOD + EXPOSURETIME} ! Delay for ERASE signal

.param LS = 0.36u
.param WS = 5.04u
.param LB = 0.36u
.param WB = 5.04u
```

```
.param LC = 0.36u
.param WC = 1.08u
.param cs = 1.15p

VDD 1 0 dc VDD
VEXPOSE EXPOSE 0 dc 0 pulse(0 VDD EXPOSE_DLY TRF TRF EXPOSURETIME PERIOD)
VERASE ERASE 0 dc 0 pulse(0 VDD ERASE_DLY TRF TRF CLK_PERIOD PERIOD)
VNRE_R1 NRE_R1 0 dc 0 pulse(VDD 0 NRE_R1_DLY TRF TRF CLK_PERIOD PERIOD)
VNRE_R2 NRE_R2 0 dc 0 pulse(VDD 0 NRE_R2_DLY TRF TRF CLK_PERIOD PERIOD)

x1_1 1 EXPOSE ERASE NRE_R1 OUT_1 PIXEL Ipd_1=50p
x1_2 1 EXPOSE ERASE NRE_R1 OUT_2 PIXEL Ipd_1=150p
x2_1 1 EXPOSE ERASE NRE_R2 OUT_1 PIXEL Ipd_1=150p
x2_2 1 EXPOSE ERASE NRE_R2 OUT_2 PIXEL Ipd_1=50p

xc1 1 OUT_1 ACTIVE_LOAD
xc2 1 OUT_2 ACTIVE_LOAD

.subckt ACTIVE_LOAD vdd OUT
mpc OUT OUT vdd vdd PMOS W=WC L=LC
cc OUT 0 3p
.ends

.subckt PIXEL vdd EXPOSE ERASE NRE OUT
x1 vdd N1 PhotoDiode Ipd_1=Ipd_1
* DRAIN - GATE - SOURCE - BULK - type - W=w - L=w
mn1 N1 EXPOSE N2 0 NMOS W=WS L=LS
mn2 N2 ERASE 0 0 NMOS W=WS L=LS
c1 N2 0 cs
mp3 0 N2 N3 vdd PMOS W=WB L=LB
mp4 N3 NRE OUT vdd PMOS W=WS L=LS
.ends

.subckt PhotoDiode VDD N1_R1C1
I1_R1C1 VDD N1_R1C1 DC Ipd_1
d1 N1_R1C1 vdd dwell 1
.model dwell d cj0=1e-14 is=1e-12 m=0.5 bv=40
Cd1 N1_R1C1 VDD 30f
.ends

.include P18_cmos_models.inc
.include P18_model_card.inc
```

9 Appendix E

Netlist for the verilog module expcounter

```
‘timescale 1 ms / 1 ns

//{{ Section below this comment is automatically maintained
// and may be overwritten
//{module {Readout}}


module Exp_counter (Ovf, Expose, Clk, Reset, Time);
        output Ovf;
        input Expose, Reset, Clk;
        input reg [4:0] Time;
        reg [4:0] a=0;
        reg Ovf=0, counting=0;

        always @(posedge Clk)
        begin
                if (Reset == 1)
                begin
                   Ovf = 0;
                   a = 0;
                end
                else if(Expose==1)
                begin
                        if (counting==0)
                        begin
                                a = Time;
                                counting=1;
                        end
                        if (a>0)
                        begin
                                a = a-1;

                        end
                        if (a==0)
                                begin
                                        Ovf = 1;
                                end
                end
                if(Expose==0)
                        begin
```

```
                                Ovf = 0;
                                counting = 0;
                        end
        end
endmodule
```

10 Appendix F

Netlist for the verilog module readout

```
‘timescale 1 ms / 1 ns

module Readout (Exp_inc, Exp_dec, NRE_1, NRE_2, ADC, Expose, Erase, Init,
    ↪ Reset, CLK, Overflow, Time);
output NRE_1, NRE_2, ADC, Expose, Erase, Time;
input Init, Reset , CLK, Overflow, Exp_inc, Exp_dec;
parameter Exp_min = 1;
parameter Exp_max = 29;
reg NRE_1=1, NRE_2=1, ADC=0, Expose=0 , Erase=1;
reg [3:0]state=4’b0000;
reg [3:0]next_state = 4’b0000;
reg [4:0]Time = Exp_min;

always @(posedge CLK)
        begin
                if (Reset == 1) state <= 4’b0000;
                else state <= next_state;
        end

always @(posedge Exp_inc or posedge Exp_dec) begin
        if (state == 0) begin
                if (Exp_inc == 1) begin
                        if(Time < Exp_max) Time++;
                end
                else begin
                        if(Time > Exp_min) Time--;
                end
        end
end
always @(state)
begin
        case (state)
                4’b0000: //idle
```

```
begin
        NRE_1=1;
        NRE_2=1;
        ADC=0;
        Expose=0;
        Erase=1;

end
4'b0001: //Expose
begin
        NRE_1=1;
        NRE_2=1;
        ADC=0;
        Expose=1;
        Erase=0;


end
4'b0010: //Wait
begin
        NRE_1=1;
        NRE_2=1;
        ADC=0;
        Expose=0;
        Erase=0;

end
4'b0011: // Before Readout 1
begin
        NRE_1=0;
        NRE_2=1;
        ADC=0;
        Expose=0;
        Erase=0;
        next_state= 4'b0100;
end
4'b0100: // Readout 1
begin
        NRE_1=0;
        NRE_2=1;
        ADC=1;
        Expose=0;
        Erase=0;

end
4'b0101: // After Readout 1
begin
```

```
        NRE_1=0;
        NRE_2=1;
        ADC=0;
        Expose=0;
        Erase=0;

end
4'b0110: // Wait
begin
        NRE_1=1;
        NRE_2=1;
        ADC=0;
        Expose=0;
        Erase=0;

end
4'b0111: // Before Readout 2
begin
        NRE_1=1;
        NRE_2=0;
        ADC=0;
        Expose=0;
        Erase=0;
        next_state= 4'b1000;
end
4'b1000: // Readout 2
begin
        NRE_1=1;
        NRE_2=0;
        ADC=1;
        Expose=0;
        Erase=0;
        next_state= 4'b1001;
end
4'b1001: // After Readout 2
begin
        NRE_1=1;
        NRE_2=0;
        ADC=0;
        Expose=0;
        Erase=0;

end
4'b1010: // Wait
begin
        NRE_1=1;
        NRE_2=1;
```

```
                        ADC=0;
                        Expose=0;
                        Erase=0;

                end
                default:
                begin
                        NRE_1=1;
                        NRE_2=1;
                        ADC=0;
                        Expose=0;
                        Erase=1;
                end
        endcase
end
always @(state or Init or Overflow) begin
        case (state)
                4’b0000: if (Init == 1) next_state <= 4’b0001;
                        else next_state <= 4’b0000;
                4’b0001: if (Overflow == 1) next_state <= 4’b0010;
                else next_state <= 4’b0001;
                4’b0010: next_state <= 4’b0011;
                4’b0011: next_state <= 4’b0100;
                4’b0100: next_state <= 4’b0101;
                4’b0101: next_state <= 4’b0110;
                4’b0110: next_state <= 4’b0111;
                4’b0111: next_state <= 4’b1000;
                4’b1000: next_state <= 4’b1001;
                4’b1001: next_state <= 4’b1010;
                4’b1010: next_state <= 4’b0000;
                default: next_state <= 4’b0000;
        endcase
end
endmodule
```

11 Appendix G

Netlist for the verilog module main

```
‘timescale 1 ms / 1 ns

module Main(Init, Exp_inc, Exp_dec, Reset, Clk, NRE_1, NRE_2, ADC, Expose,
    ↪ Erase);
output NRE_1, NRE_2, ADC, Expose, Erase;
reg[4:0] Time;
```

```
input Init, Exp_inc, Exp_dec, Reset, Clk;
wire Overflow;

Readout RO(
.Exp_inc(Exp_inc),
.Exp_dec(Exp_dec),
.NRE_1(NRE_1),
.NRE_2(NRE_2),
.ADC(ADC),
.Expose(Expose),
.Erase(Erase),
.Init(Init),
.Reset(Reset),
.CLK(Clk),
.Overflow(Overflow),
.Time(Time));

Exp_counter EC(
.Ovf(Overflow),
.Expose(Expose),
.Clk(Clk),
.Reset(Reset),
.Time(Time));
endmodule
```

12 Appendix H

Netlist for the testbench in verilog for the module main

```
‘timescale 1 ms / 1 ns

module testbench_Main ();

       logic Init = 0, Exp_inc = 0, Exp_dec = 0, Reset = 0, NRE_1 , NRE_2,
           ↪ ADC, Expose, Erase;
       reg clk = 0;
       always begin
              #0.5clk = ~clk;
       end
       Main m1(Init, Exp_inc, Exp_dec, Reset, clk, NRE_1, NRE_2, ADC, Expose
           ↪ , Erase);

       initial begin
```

```
        #0.5 Exp_dec=1; #1 Exp_dec=0; // Testing if timer goes under 2 sec
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; // Testing that
            ↪  Exp_inc works
        #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; Exp_dec=1; // Testing that
            ↪ Exp_inc has priority over Exp_dec
        #1 Exp_inc=0; Exp_dec=0;
        #1 Exp_dec=1; #1; Exp_dec=0; #1 Exp_dec=1; #1; Exp_dec=0; // Testing
            ↪ that Exp_dec works
        //the exposure time is now 8 ms
        #1 Init=1; #1 Init=0; // Testing if the FSM starts when init is 1
        #20
        #1 Init=1; #1 Init=0; #5 Reset=1; #1 Reset=0; //Testing if reset has
            ↪ priority over everything.
        #5
        #1 Init=1; #1 Init=0; #4 Init=1; #1 Init=0; //Testing if Init i goes
            ↪ high, while the fsm is cycling trough states, the fsm continues
            ↪  until it is done
        #20
        // Increasing exposure time way over 30 ms to check if it stops at 30
            ↪  ms
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        #1 Exp_inc=1; #1 Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0; #1 Exp_inc
            ↪ =1; #1; Exp_inc=0; #1 Exp_inc=1; #1; Exp_inc=0;
        Init = 1; #1 Init = 0;
        #50
        $finish;
        end
endmodule
```