2019-04-09

Real-Time Wide Dynamic Range Capture and Display System Based on Mantissa-Exponent Representation

Shahnovich, Ulian

UNIVERSITY OF CALGARY

Real-Time Wide Dynamic Range Capture and Display System Based on Mantissa-Exponent Representation

by

Ulian Shahnovich

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

APRIL, 2019

Abstract

In this work, we present a wide dynamic range (WDR) vision system that is able to capture bright and low light objects in a single frame without being over or underexposed. The data is captured and streamed in real time (RT) mode from the sensor to a monitor at 60 fps or a PC.

The presented system includes:

1) WDR complementary metal-oxide-silicon (CMOS) image sensor design which is able to update the integration time of each individual pixel depending on the ambient light and provide output in a mantissa-exponent format (floating point) [1], [2].

2) Interface hardware designs including PCBs and HDL drivers required to activate the sensor and all the peripheral components.

3) A tone mapping algorithm based on a mantissa/exponent representation along with its hardware implementation that allows us to render WDR data on a conventional low dynamic range (LDR) devices[3].

Preface

Throughout human existence, we can observe a constant process of creation of art which has one of its expressions as drawings of familiar or imaginary objects. As the civilization moved forward, the drawings transferred from caves to stones, from stones to trees or leather, and later to newly invented materials such as rags and papers. Where the final creation stands behind a digital way of data capturing and presentation such as TVs, smartphones, printers, or virtual reality systems. Along with the presenting devices development process, art creation tools also improved and significantly changed. In the beginning, humans used brushes, sticks or stones, with different types of pigmentations that could be found in nature for colors to create a painting and any type of art. The rise of the sciences; especially chemistry, brought us to the invention of photography about two hundred years ago, where humanity no longer had the need to spend time or to use complicated tools to depict their visual perception of objects. And finally, the latest achievement is standing behind capturing of scenes on a digital camera and its direct presentation on any types of image rendering devices [Figure I].

Figure I: Example of human art development from the oldest on the left to the latest on the right; Lascaux Cave Paintings about 17,000 years old; Papyrus, paint 1050 B.C. Book of the Dead for the Singer of Amun; 18th-century style artist Kasia Wozniak – R; Digital Photo.

For the last 25-30 years, capturing and rendering digital devices were significantly improved in main parameters such as resolution, frame rate, color depth, and physical dimensions. The developed improvements have allowed a realistic representation of the environment in most cases. Digital image sensors can feature more than 250 Million pixels, or with less resolution, can feature high frame rates with more than 25k frames per second [4]. The dimensions of the sensors are also very impressive, with a size of less than a quarter of 1 mm^2, allows them to be deployed in many applications where size is a critical consideration [5].

Current rendering devices can also feature high pixel per inch (PPI) density, with resolutions of more than 8k, and frame rates of 240fps or above, which all together allows visualizing image data in a very realistic manner. Regardless of all the presented features, dynamic range (depth) of the scene (or the image data), is a limiting factor in allowing cameras to "see" perfectly, and to rendering devices to visualize image data in a completely realistic manner.

The dynamic range represents the ratio between the darkest and the brightest intensities in the scene and is usually measured in dB where the value is given by (). In capturing devices such as digital CMOS image sensors, the dynamic range is mainly dependent on the photodiode capacity and the noise floor of the signal. Precise dynamic range considerations for CMOS image sensors will be discussed in the next section.

(Equation I) $$\boldsymbol{dynamic\ range = 20 \times log_{10}(\frac{brightest\ intensity}{darkest\ intesity})}$$

To present the image data (that was either captured or created), we will need to use a possible rendering device such as a monitor, projector, printer, etc. In most of the cases current solutions will work well; however, if the depth of the data exceeds the rendering capabilities of

these devices (Due to high or low light levels in the presented scene), they will either truncate the data or in the worst case they will not render the signal at all. Current state-of-the-art rendering devices such as TVs or monitors have different abilities in presenting high contrast data which is usually dependent on the technology they are based on. Manufacturers claim the dynamic contrast is 1:1,000,000 and the static contrast can reach 1:100,000 with a peak luminance of 600cd/m2 on state-of-the-art OLED displays; however, the measured contrast ratio in a dark room remains at around 1:350 if compared to a real-life illuminance scale [6]. This rendering limitation means, if we want to present data with a higher depth on existing devices, we will need to compress the depth dimension of this data, this compression is called tone mapping, which will be discussed in the next section.

In this thesis, we present two possible solutions for:

1) Capturing scenes with wide dynamic range (WDR) of brightness values
2) WDR data rendering for low dynamic range devices.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Orly Yadid-Pecht for her invaluable guidance, support. and encouragement throughout my research work. Secondly, I thank all the I2Sense lab students, post-doctoral fellows, my friends and department staff for their help and support. Many thanks to Dr. Alain Horé, Dr. Jie Yang, Dr. Kartikeya Murari, Nikhil Vastarey, Heather Henley, Cristobal Martinez, Matan Assaf, and Devin Michael Atkin. Thirdly, I would like to thank my committee members for taking their time to serve in my committee. Last but not least, I would like to thank the CMC Microsystems for access to the design tools, workshops, and fabrication services through MOSIS.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations, and Nomenclature

Symbol	Definition
WDR	Wide Dynamic Range
CMOS	Complementary Metal Oxide Silicon
LDR	Low Dynamic Range
ASIC	Application Specific Integrated Circuit
CMC	Canadian Microelectronics Corporation
EV	Exposure Value
CMOS	Complementary Metal Oxide Silicon
OLED	Organic Light Emitting Diode
CCD	Charge Coupled Device
FPN	Fixed Pattern Noise
SNR	Signal-to-Noise Ratio
CIS	CMOS Image Sensor
DR	Dynamic Range
TMO	Tone Mapping Operator
FPGA	Field Programmable Gate Array
M-E	Mantissa-Exponent
TL	Time Latch
S&H	Sample & Hold
LDO	Low Dropout Regulator
PCB	Printed Circuit Board
PGA	Pin Grid Array
VS	Visual Studio
DVI	Digital Visual Interface
HPF	High Pass Filter
SSIM	Structural Similarity Index
PSNR	Peak Signal to Noise Ratio
TMQI	Tone Mapped Image Quality Index
PHY	Physical Layer
ADC	Analog to Digital Converter
S&H	Sample and Hold

Chapter 1 : **INTRODUCTION**

This thesis will discuss two different problems, and thus the introduction is divided into two main parts. Part one will discuss the WDR capture, and part two will discuss the WDR data display.

1.1. WDR Capture

Before CMOS image sensors became popular, and before the Charged Coupled Device (CCD) was invented, there were MOS sensors. With varying levels of success of capturing light NMOS, PMOS, or BJT image technologies were implemented; however, these sensors suffered from high fixed pattern noise (FPN) which limited their applications. When CCD technology was firstly introduced for light sensing applications during the late 1960s [7], [8] with improved sensitivity and fewer process variations, many companies started to design image sensors based on a CCD concept, this was the beginning of digital photography. Image sensors based on the CCD concept continued to provide improved image quality, resolution, and frame rate until the active pixel sensor was developed in 1985 by Tsutomu Nakamura [9]. and more broadly defined by Eric Fossum in 1993 [10].

Even though image sensors based on CMOS technology were less sensitive, they had a few advantages over CCD technology which made them so broadly successful. The advantage of image sensors based on CMOS technology was the possibility of integrating well-defined components on the same chip such as ADC, memory, and amplifiers, with the sensing matrix (pixels). This made the design process faster, less complicated, and reduced some noise related to external readout of analog signals. More than that, CCD sensors require different voltage levels compared to other on-board components, requiring additional peripheral design, and they were significantly more power “hungry” which made them unsuitable for low power applications

[11]. These disadvantages made CCD technology a very small niche, and today they are found only in applications where very high sensitivity is required above any other consideration, e.g., astronomy or microscopes[12].

For the last 25 years, CMOS image sensors have improved significantly in features such as sensitivity, resolution, frame rate, and dynamic range. Although current features are sufficient for most applications, the Dynamic Range ability is limited and requires either automatic or manual calibration of Exposure Value (EV) and gain to select the required brightness level during the capturing process as presented in Figure 1-1 - Same scene captured with different EV values..

Figure 1-1 - Same scene captured with different EV values.

To understand the challenge of extending the dynamic range let's consider the following; an image sensor is an array of photodiodes which convert incident light into a photocurrent i_{ph}. Since the change in photocurrent is linearly related to light intensity, a photocurrent is a good measure for the intensity of the captured light. The resulting photocurrent is usually very low and difficult to measure, so it is accumulated into a charge which is later linearly translated into a voltage, and finally, the voltage is digitized and read out.

Eliminating the dark current additive noise, which can be very destructive, noise can be expressed as a sum of four independent components[13]:

1) FPN gain and offset due to process variation and device mismatches.

2) Readout circuit noise (including quantization noise) with zero mean and average power of $\sigma^2_{\text{Readout}}$.

3) Reset (kTC) noise.

4) Integrated shot noise, which has zero mean and average power $i_{ph} * t_{int} \Big/ q_e^2$ where q is the electron charge. The output charge equation can be expressed as in Equation 1-2. Where $Q(t_{int}) \leq Q_{max}$ and the saturation charge is referred to the well capacity.

Equation 1-1 - $$\boldsymbol{Q(t_{int}) = \frac{1}{q}(i_{ph}t_{int} + Q_{FPN} + Q_{Readout} + Q_{kTC} + Q_{Shot})}$$

Assuming the correlated-double-sampling (CDS), is performed, we can eliminate Q_{RESET} and the offset part Q_{FPN}. And if we assume that Fixed Pattern Noise (FPN) is negligible compared to shot noise, Signal to Noise Ratio (SNR) is given by Equation 1-2

Equation 1-2 - $$\boldsymbol{SNR(i_{ph}) = \frac{(i_{ph}t_{int})^2}{qi_{ph}t_{int}+q^2\sigma^2} \quad for \quad i_{ph} \leq \frac{qQ\ max}{t_{int}}}$$

Note that SNR increases with i_{ph}, first at $20dB \Big/ decade$ when readout noise variance dominates, and then at $10dB \Big/ decade$ when shot noise variance dominates. SNR also increases with t_{int} . Thus, it is always preferred to keep the integration time as long as possible.

Image sensor Dynamic Range is defined as a ratio of the largest unsaturated value to the smallest detectable value, typically defined as a standard deviation of the noise under dark current conditions. Assuming the above sensor model, $i_{max} = qQ_{sat} \Big/ t_{int}$ and $i_{min} = q\sigma_{read} \Big/ t_{int}$ the dynamic range is given by Equation 1-3

Equation 1-3 $$\mathbf{DR} = \frac{\mathbf{i_{max}}}{\mathbf{i_{min}}} = \frac{\mathbf{Q_{max}}}{\boldsymbol{\sigma_{read}}}$$

Extending the dynamic range at the high end requires increasing i_{max} in one of the following ways: Varying integration time: The integration time is adapted to pixel photocurrent

providing long integration times for pixels with small photocurrents and short integration times for pixels with high photocurrents. Examples of such techniques are - well-capacity adjusting [14]–[16], multiple-reset [2], [17], [18], logarithmic sensors variations [19] or other HDR schemes which are not related to the parameter modifications of the DR equation such as integration of multiple captures [20], [21].

Extending DR at the low end requires reducing i_{min}, which can be achieved by, reducing $\sigma_{Readout}$ [22], by increasing t_{int}, or when using multiple-resets a combination of image blur prevention and weighted averaging of the samples[20].

In this thesis, we present an architecture to extend the DR into the high end which is based on time and space trade-offs [1]. That implies an implementation of a multidimensional pipeline control logic integrated with the pixel array that allows achieving more than 120dB of linear dynamic range with up to 8 conditional resets and provides an output in mantissa-exponent format. Compared to a previously designed CMOS Image Sensor (CIS), which is based on the same concept but with only 3 integration times (2 conditional resets) [23], the presented design has an increased DR, requires less memory and has more quantization levels to represent the light, which increases the contrast and overall naturalness of the scene. In the presented sensor, the internal pipeline controls the integration time of every pixel separately and can update their integration time to the following partial values $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\ \frac{1}{32}, \frac{1}{64}, \frac{1}{128}$, and $\frac{1}{256}$ of the global frame time based on the amount of light they receive. This implies that the sensor can conduct up to 8 conditional resets to each pixel based on the ambient light. Knowing the number of resets

integrated with the sampled analog value of each pixel allows us to represent the captured WDR data directly in a mantissa-exponent structure $value = mantissa * 2^{exponent}$ where mantissa is the final sampled analog value of the pixel and the exponent is the number of resets the pixel occurred during the frame time.

The presented technique allows us to embed nine integration times into in a single shot acquisition time with corresponding incremental of the Dynamic Range by $20 * log_{10}(2^8) = 48dB$. The number of resets is a function of the sensing matrix dimensions, and for a bigger sensor (4096 rows), the number of resets will increase to 12 with a corresponding increment of the dynamic range of another $24dB$.

1.2. WDR Display

Compared to low dynamic range sensors, WDR sensors allows us to capture more levels of the existing light intensities per pixel; this lets us see more details, have higher contrast, and obtain high range data without losing information regardless of their illumination levels in the scene. Captured WDR data may have more than 1000 times larger range than the presenting capabilities of conventional visual devices. These LDR have limited range, or brightness values they can render per single pixel and depending on the technology (Plasma/ LED/ OLED/ LCD, etc.), the maximum range is usually varying from 1:256 to up to 1:1024 of discrete values, i.e., 8-10 bit of brightness levels whereas the WDR data has 20-bits or more. The limitation of existing display devices led to the development of the image processing application area called tone mapping. Tone mapping or tone mapping operator (TMO) is a technique used in image processing and computer graphics to map one set of brightness intensities to another to approximate the appearance of high-dynamic-range images in a medium that has a more limited dynamic range[24]. Tone mapping is a compression performed on HDR data to meet the

LDR visualization standards of 8-10 bits by preserving the overall image naturalness and structure.

Many TMOs have been developed by the academic or industrial groups for the last two decades. Even though all of them has the same purpose, they all feature different capabilities and work in different ways. Some of them are highly efficient and can be implemented on hardware platforms such as a GPU or FPGA for real-time applications [25]–[29], and some of them are less efficient [30] but give better visual results. Usually, the faster algorithms are based only on a global operator (single equation applied to all the pixels in the picture) [31]–[33], whereas more complex tone mapping have both, global and local operators (local operator is more complex operator which is based on a spatial intensities and thus requires more memory and computational effort to generate value) [34]–[37]. The latest achievements in TMOs are based on neural network computing techniques[38]–[40].

To compress the data generated by the newly developed image sensor a special TMO is required. The output from the sensor has a mantissa-exponent data structure where the mantissa represents the detail level, and the exponent represent the scaling factor of the mantissa's values. To perform a tone mapping process on this data, we could transform the data into a standard binary representation as is usually performed[41][42]; however, this was avoided due to the additional logic implementation required making it an expensive operation in an ASIC. Thus, we present here a new tone mapping algorithm that can process the image data in the specified format. The designed TMO takes advantage of the input data structure for compression proposes; this ensures lower computational effort, small footprint and possible integration with the image sensor on the same chip. Integration of the TMO with the image sensor will reduce, the interface with the image sensor, the driver complexity, and the complexity of the entire

system in general. These features make the sensor suitable for medical, low power, and low dimension WDR imaging applications. We note that for WDR inputs that do not satisfy our mantissa/exponent representation, a conversion module may be used for formatting the pixel intensities into the appropriate representation [Figure 5-9].

In the following sections, we will present the sensor design, the interface between the sensor and the FPGA, the hardware driver to control the system and to stream WDR output, and the tone mapping algorithm with its hardware implementation.

Chapter 2 **: M-E SENSOR**

The image sensor presented in this section was designed using the Cadence Virtuoso software combined with an AMS 350-nm CMOS opto-process technology. Selected tools and technology were provided by Canadian Microelectronics Corporation (CMC) Microsystems and were the most suitable for light capturing application and for prototyping compared to other provided technologies. We note that for improved image quality and lower SNR features a special opto-process is required which should include pinned photodiode device, designed for CIS analog components (charge amplifiers, capacitors) column ADCs, CFA, smaller gate size, etc. A presented CIS has a completely new design which allows access in parallel to up to 9 pixels at the same time for resetting proposes to generate WDR output. Modules for this CIS (FIFO buffers, Multiplexers) were designed for possible integration of the presented pipeline with previously designed solutions by our group [43]–[45].

The following chapter is organized as follows: at the beginning, we present a precise explanation of the designed blocks along with their operation and necessity to the design, and

after we finish to present each of the modules, we explain how all these modules work together and generate WDR output.

The presented CIS is designed to perform up to 8 conditional resets for each pixel during the integration time to avoid their saturation at the end of the frame time. To implement this logic and to establish parallel access and conditional resets to 8 pixels a few necessary components

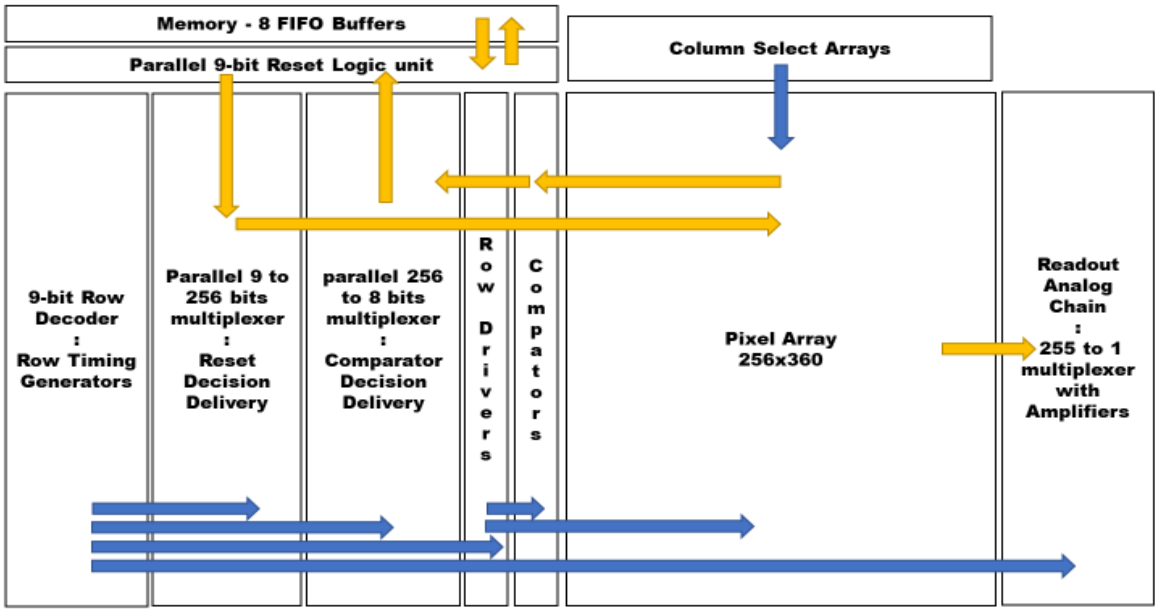


Figure 2-1 – Sensor components. Blue arrows symbolize the control lines. Yellow arrows symbolize the data flow

were designed and integrated together: 9-bit Row Decoder, Single bit ADC (latched comparator), parallel 256 to 8 bits multiplexer, parallel 9-bit Reset Logic unit, parallel 9 to 256 bits multiplexer, Row Driver, multiple access low noise pixel and readout chain. The complete design is presented in the block diagram in Figure 2-1. In the following subsections, we will describe each of the components and its importance for the presented sensor. And at the end, we will describe the control/data-flow and WDR output generation process.

2.1. 9-bit Row Decoder

In a standard row decoder one signal can be selected (activated) each time, and the address of the selected signal is provided as a binary input to the decoder. This allows random access to each row in the image sensor. In our design, there is parallel access to 9 rows simultaneously, and regular decoder couldn't provide the necessary functionality. There was an optional solution to use 9 row decoders in parallel; however, integrating 9 parallel decoders will require 9 addressing busses, will consume a lot of power and a lot of chip space (row decoder is a relatively big component). Since in our design, there is no random access to the rows, a different solution for row decoding, based on an array of flip-flops only was designed. A presented solution also allowed to reduce control logic complexity to only two signals (only reset and clock signals required), make the design smaller and reduce the total power consumption of the sensor. Here we explain the operation and design consideration of this module.

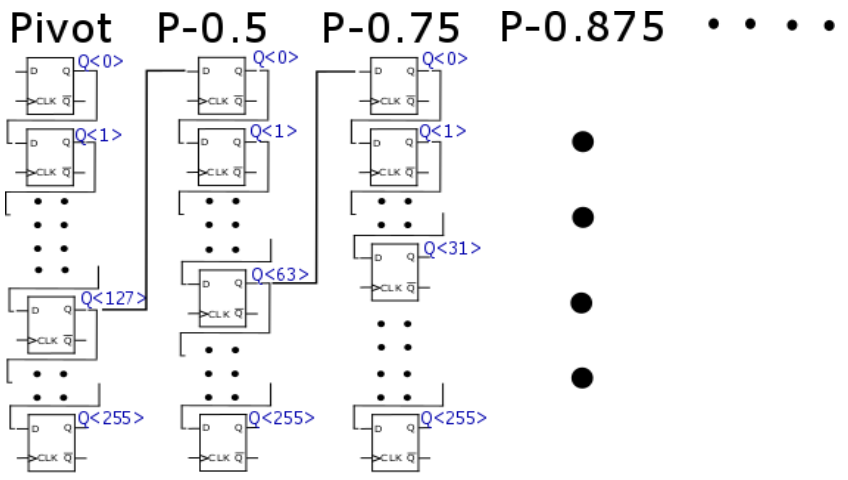


Figure 2-2 - Row decoder internal design

The number of generated by this module signals and the physical distances between them is strongly dependent on the number of rows in the pixel matrix. In the designed sensor the number of rows is 256 and thus, following time and space logic, the intervals between the accessed pixels and the intervals between the control lines are the following: 128, 64, 32, 16, 8, 4, 2, 1, and another 1 (in terms of rows).

The distances between the nine activated control signals are always fixed, and there is no option of random access (i.e., randomly switching from row to row).

The nine active control lines are moving in a circular direction and activating a different set of 9 rows in the image sensor, where each next selection of 9 rows is triggered by the clock signal.

The decoder (Figure 2-2) was implemented using flip-flops instead of using a common for row decoders combinational logic design flow for a few reasons: There is no random access, it has a much smaller footprint, only two control lines (reset and clock), and it consumes much less power.

The module works in the following way:

1) The "running" logical 1 signal (called Pivot) represents the start of the frame integration time.
2) This signal is "moving" in a circular direction from the first to the last (255^{th}) flip-flop and then repeat its cycle.
3) Every time this signal crosses its halfway (flip-flop #127), it sends a logical "1" to the first flip-flop in the next circular buffer (called Pivot - 0.5).

This connection creates an effect of running in a circular direction two signals with a phase of exact half frame time from each other. We note: The logic behind the buffer names is in the delay between them and the Pivot signal.

The next circular buffer is called “Pivot - 0.75” (Pivot -1/2-1/4), and the delay from Pivot will be exactly 0.75 frame time. This buffer receives logical “1” from Pivot – 0.5 buffer from its 64th flip-flop. We continue to connect all the following circular buffers in the same way and thus create all the explained above 9 signals/delays.

2.2. Latched Comparator

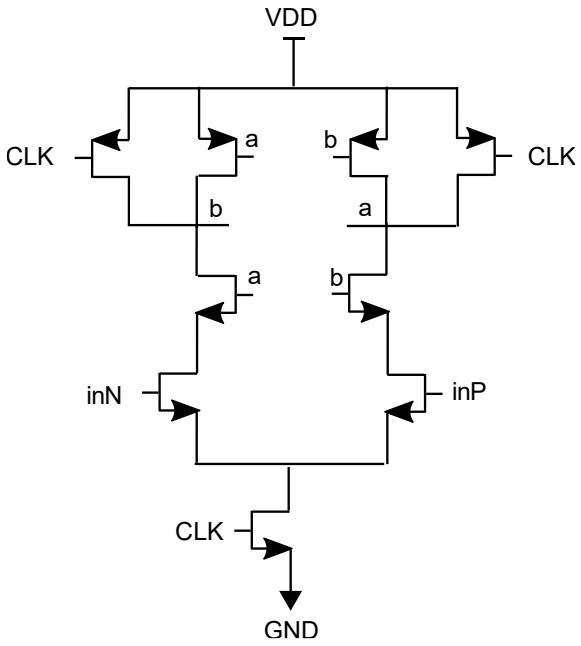


Figure 2-3 - Latched comparator, transistor level design

The purpose of this unit (Figure 2-3) is to compare two analog values (inP and inN) and return the result in digital form (terminal a and b, are outputs), in other words, this module works as a single bit Analog to Digital Converter (ADC). This module is also synchronized so we can use its output without inserting another latch. This is important for pipeline integrity and will be explained later in the sensor operation section.

The reset logic, memory, and control circuits are digital circuitries whereas the pixel matrix, the amplifiers, and the rest of the circuitries are based on analog concepts; therefore, this

type of comparator plays a significant role as a separator in the presented mixed analog & digital design and a few peripherals which were added to the original comparator to improve its functionality. Here we will discuss the principles of these improvements.

Analog inputs were buffered with PMOS based common-drain (source follower) amplifiers (pre-amplifiers) for the following reasons. To avoid injections of switching noise back into the pixel matrix. The latch and the input clock signal are injecting a significant amount of charge back to the analog parts which are then reflected to the sensing matrix (using an amplifier before this module will suppress this noise). To match the analog values of the *read line* (from pixel array) to the required by the comparator input values (the inputs of the latched comparators must be above one volt, whereas the voltage over the *read line* is varying between 0 and 1V). The PMOS based source follower increases the input analog value that fits the analog swing of the comparator.

Digital outputs were buffered with inverters to equalize load capacitance on the output terminals. Different load capacitance connected to the output terminals will have a direct impact timing of the latch circuit and affect the cooperation abilities of the comparator.

2.3. Parallel 256 to 8 bits Multiplexer

In every clock cycle, up to eight pixels are being checked in parallel for possible conditional reset. To meet the constraint of parallel access, an 8-bit multiplexer has been designed (Figure 2-4). This module delivers 8 of 256 possible results from the latched comparators, to the corresponding reset logic circuitries. The multiplexer inputs are presented on the right side and called cmpIn0-cmpIn255 (output from the comparators) whereas the outputs of the multiplexer (left side) are called res1-res8 (connected to the reset logic inputs). The select signals of the multiplexer $(TL1 - TL8)$ and $(TL1n - TL8n)$ are generated by the explained

above 9-Bit Row-Decoder module and define the exact encoded combination to deliver 8 signals from the inputs to the outputs of the multiplexer. The multiplexer was designed using transmission gates instead of digital logic cells (Figure 2-5) mainly for integrating of current design with active column sensing, SAR and AR solutions presented by our research team earlier which requires of delivering an analog signal to the reset logic circuits.

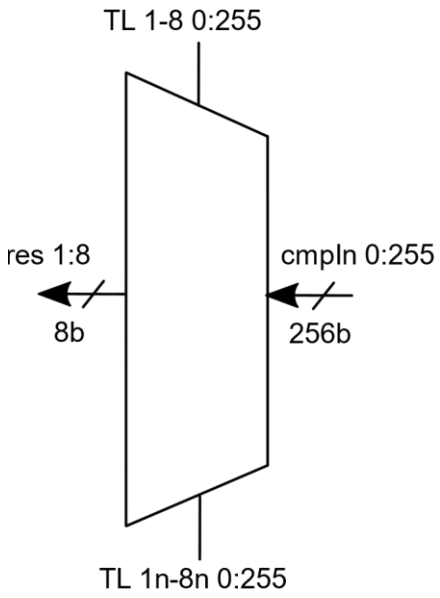


Figure 2-4 - 256 to 8 Parallel Multiplexer

A similar to a presented multiplexing module, another module was designed to deliver eight results from the reset logic module back to the selected eight pixels and called demultiplexer (Figure 2-6). This module has 10 inputs (resConf0-9): eight of them deliver the conditional reset data, one to deliver the global reset signal, and the last to deliver logical “0” to the 245 non-selected MOS gates (common to TTL logic design considerations). This multiplexer has 256 outputs (resetToPixel0-resetToPixel255) that activate the corresponded reset “lines” in

the pixel array. The select lines of the multiplexer $(TL0 - TL9)$ and $(TL0 - TL9)'$ are controlled by the same 9-Bit Row-Decoder module and define the exact encoded combination to enable delivery of the reset signals back to the pixel array.

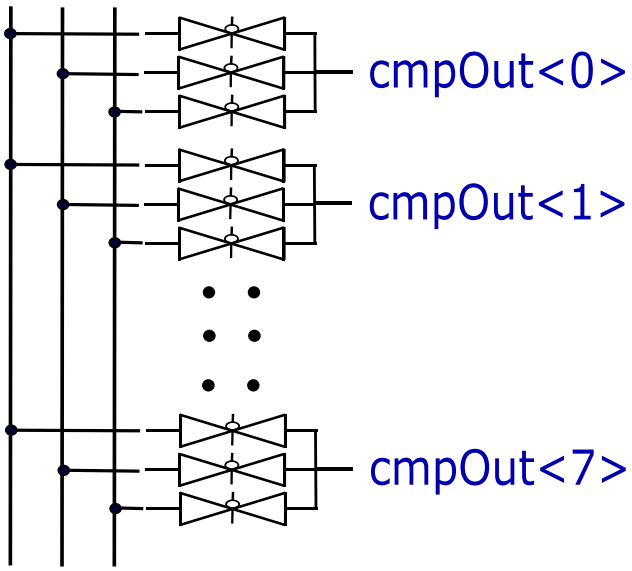


Figure 2-5 - Single cell of the 256:8 multiplexer

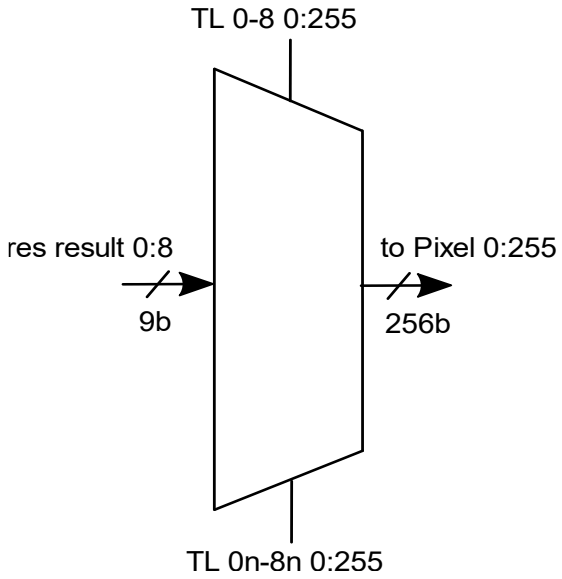


Figure 2-6 - 10:256 multiplexing module

2.4. 9-bit Reset Logic Module

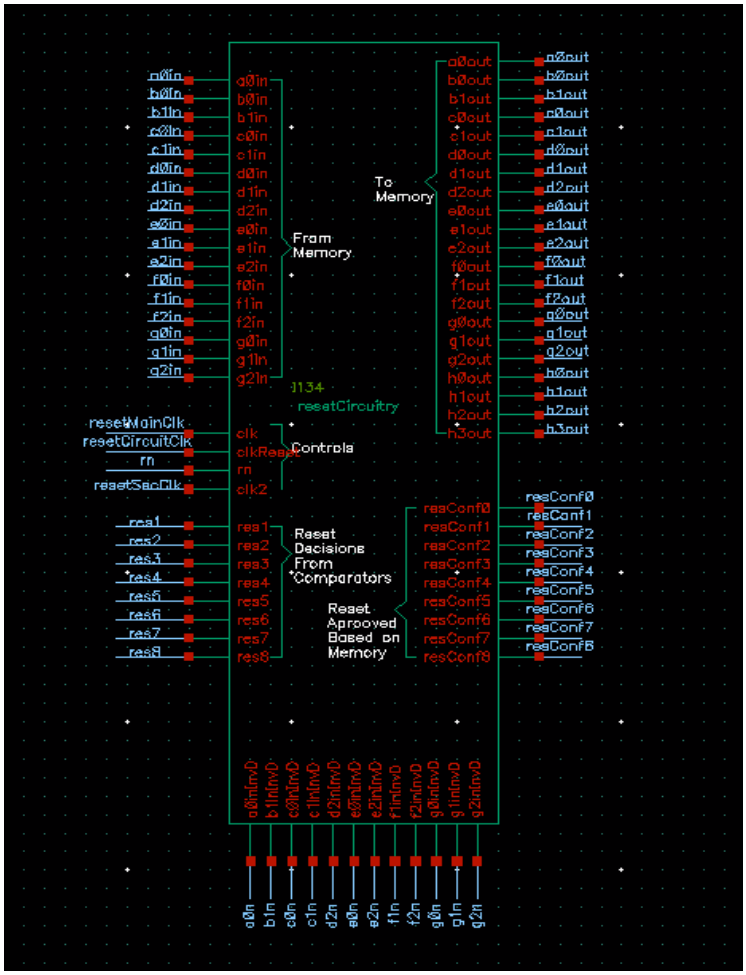


Figure 2-7 - Reset Logic Module

This module (Figure 2-7) generates all the necessary reset signals to the pixel array. Every clock cycle, this module receives information from the comparators (res1-res8) and the memory (a0in - g2in) about eight of the nine selected pixels and makes the decision to generate a conditional reset or not. Along with generating conditional reset signals (resConf1-resConf8), this module also generates a hard-reset signal (resConf0), hard reset signal indicates a frame starting point for one of the nine selected pixels in the presented rolling shutter sensor.

Reset logic module consists of nine different combinational logic circuitries for each of the resets (8 conditional and 1 hard-reset). To generate a hard-reset signal (start for the frame time), no conditions are required, and thus this circuit receives only one input bit which is the shared between all the reset circuits synchronization signal *clkReset* (

Table 2-1 case # 0). The first conditional reset circuit (

Table 2-1 case # 1), receives two bits, one is the result of the latch comparator and the second is the synchronization signal (*clkReset*). Depending on the result of the comparator, the reset logic will either generate a conditional reset or not. Regardless of the generated signal, we store the result into a FIFO buffer (a0out) until the next time this pixel is being checked (FIFO buffers and their structure is the next explained component).

The second conditional reset circuit (

Table 2-1 case# 2) oversees resetting the pixels on the second time and will generate a conditional reset signal only if the pixel crossed the threshold value and was reset previously. Therefore, this circuit has three inputs, one bit from the comparator (about current status), one bit from the FIFO (information about previous conditional reset) and the third bit is the sync. signal. Here again, we store the decision into the next FIFO buffer until the next cond. (reset, access phase (#3)). To store the decision of the second conditional reset logic, we will need two bits, and thus the interface with the memory has two bits (b1out and b0out)

Next conditional reset circuitries #3-#8 features the same logic. The difference between them is a different bit selection from the corresponding FIFO buffers. The bit selection varies from 1 to 3 bits depending on the reset circuit/case. e.g., at reset case #6. we need to check if the

pixel received 5 conditional resets. If this is the case (5th FIFO stores binary value of '101'), we can check only two bits (the first and the third one) if they store logical '1'.

The reset module is a set of combinational logic circuitries and to synchronize them to generate a reset at the same time, we use a shared between all the conditional reset circuitries a synchronization signal, and we call it *clkReset*.

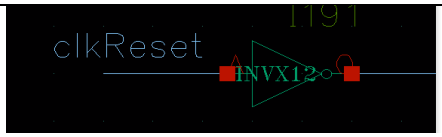
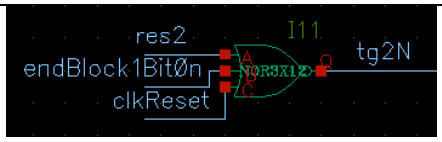
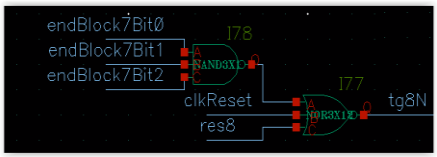
<table>
<tr><th>Reset circuit #</th><th>Schematic View</th><th>Truth Table</th></tr>
<tr><td>Hard reset
Reset #0</td><td> 
</td><td>
<table>
<tr><th>clkReset</th><th>out</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Conditional
reset #1</td><td> 
</td><td>
<table>
<tr><th>Res1</th><th>clkReset</th><th>Tg1N</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Conditional
reset #2</td><td> 
</td><td>
<table>
<tr><th>Res2</th><th>EB0_1</th><th>clkReset</th><th>Tg2N</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Conditional
reset #8</td><td> 
</td><td>
<table>
<tr><th>Res8</th><th>EB7_0</th><th>EB7_1</th><th>EB7_0</th><th>clkReset</th><th>Tg8N</th></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

Table 2-1 – Internal reset circuits for reset #0, #1, #2 and #8

2.5. Row Driver

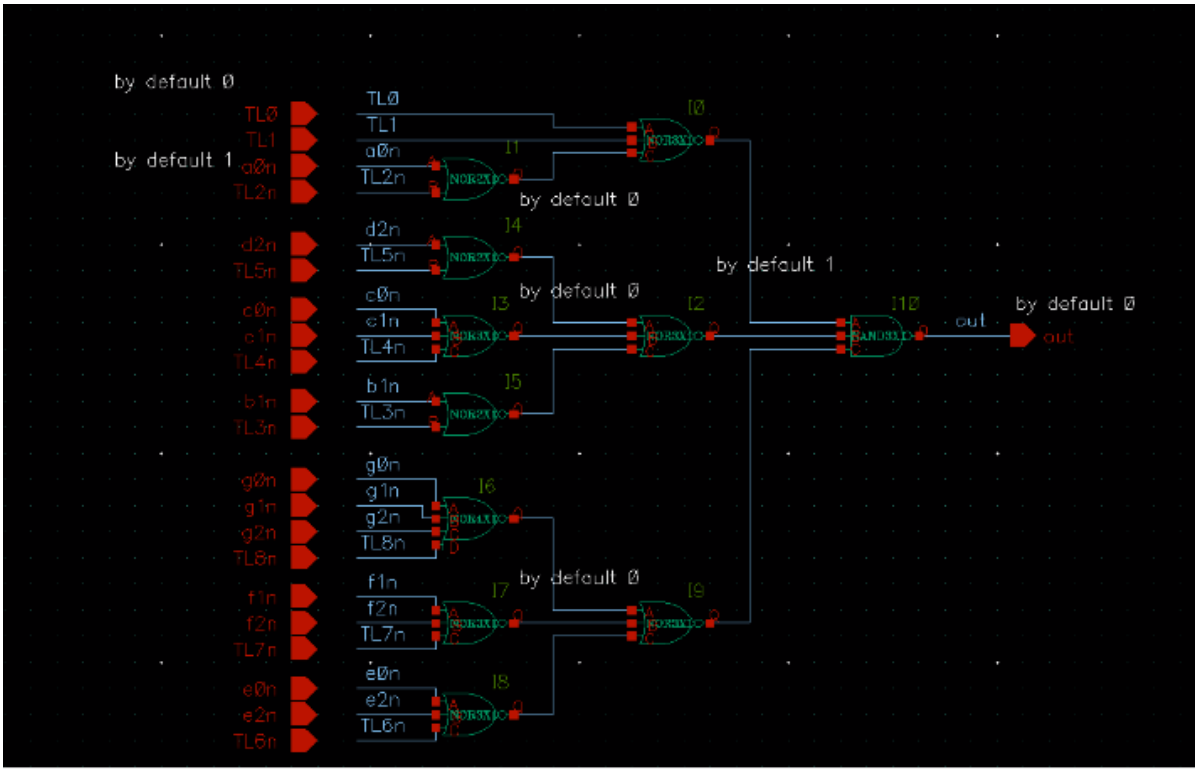


Figure 2-8 - Row Controller Module

The row driver module (Figure 2-8) allows reading the analog value of the selected pixel in the selected row by activating the Column Select line and the required for reading purposes in row level circuits (current sink, source follower amplifier, and comparator). Combined signals from the Row Decoder (TL0n-TL8n) and from the FIFO buffers will allow access to a pixel in this row. Signals from the Row Decoder are activated only if there might be a need in a conditional reset (based on time and space tradeoffs) and the signals from the FIFO shift registers will be activated only if the requested for access pixel had a conditional reset on its previous access phase (this data is stored in FIFO buffers). In total, we have 256 Row Controllers in our sensor, one for each row and the output of each one of them is latched to avoid glitch effects (Figure 2-9).

The purpose of this module is to reduce the overall power consumption of the sensor and to reduce the amount of noise injected to all the pixels in the accessed row.

The power consumption is being reduced since this module is turning on only up to 9 readout circuities from 256 possible options, and only doing so if there is a need of a conditional reset (each readout circuitry includes: a comparator, source follower amplifier, and current-sink).

The noise is reduced since the “read line” will be activated only if the pixel had a conditional reset on its previous access phase; otherwise, the access line will stay untouched. Thus, if there is no need for a conditional reset, the number of reading accesses can be reduced from nine to one. Reducing the number of accesses will reduce the noise injected to all the pixels in the accessed row when the ‘read’ line is activated or deactivated.

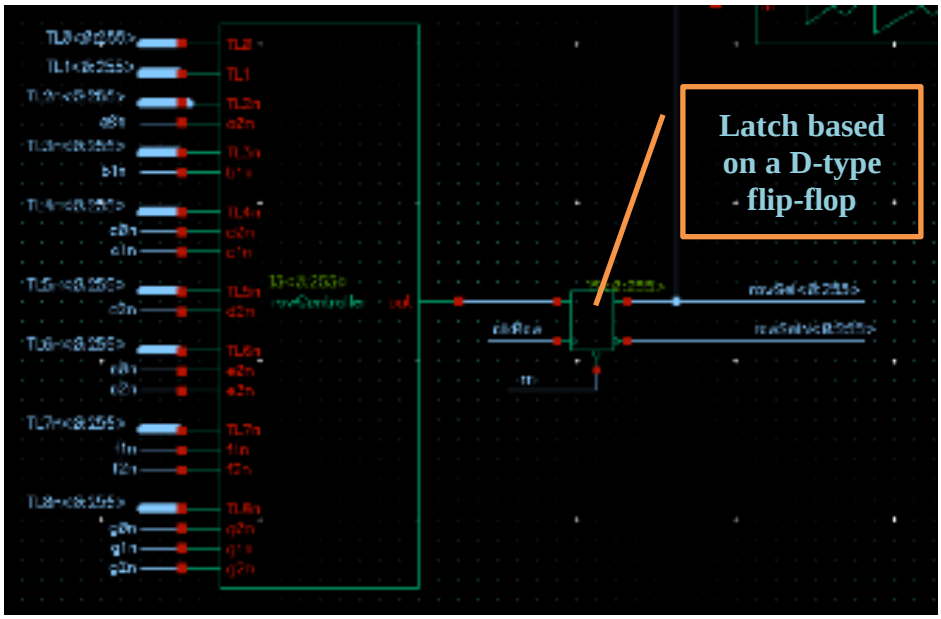


Figure 2-9 - 256 Row controllers as a sequential module (chip level view)

2.6. Pixel design

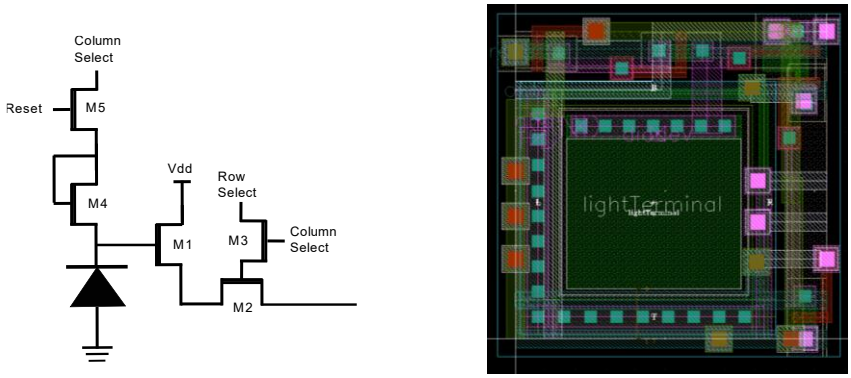


Figure 2-10 - Pixel Architecture (Schematics on the left and layout on the right)

A standard 3T pixel configuration was chosen as a source to design a pixel that allows a single pixel selection in the array as a combination of a row and column decoders. A 3T pixel with a regular photodiode was used due to lacking a pinned photodiode in an AMS35 Opto-process. Opto-process (C35B4O1): 3.3/5V provided by CMC is similar to the basic AMS35 process, but with a high-efficiency photodiode and anti-reflective coating for imaging and optoelectronic detection applications. But, to take advantage of the CMOS process for low noise imaging applications, technology with pinned photodiode must be selected.

To allow access to a single pixel from the pixel array a 3T pixel was modified as follows:

The reset transistor's configuration was changed to perform as an AND gate. This configuration allows us to reset a single photodiode in the array when both signals from the column and reset logic arrives (are logically high).

A diode-connected transistor was added between the photodiode and reset transistor to decouple switching noise. This noise is reflected on the photodiode when one of the terminals on the reset transistor is being activated, that happens only when pixels in the same row or column

are being accessed. The diode-connected transistor is activated only when there is reset access to the transistor otherwise this transistor is off, and its drain terminal is reflected as capacitance load that absorbs the charge injected from the reset transistor terminals.

Read access transistor as AND logic configuration was added to allow read access from a single pixel when a signal from the row and column decoder appear.

In total, the pixel features 5 transistors, with a size of 10.4um×10.4um and 34% fill factor (Figure 2-10).

We note – the pixel is very sensitive to access and reset noise since it features a regular instead of a pinned photodiode, using a pinned photodiode will enable us to eliminate a major amount of noise and will significantly increase the dynamic range of the sensor into the low end.

2.7. Readout Chain

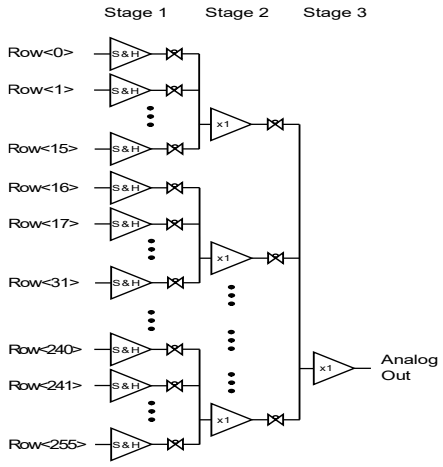


Figure 2-11 - Readout chain with 3 multiplexers and amplifiers

The readout sequence from the sensor is the following: We select a row and scan all the pixels in this row. Once the row scan is complete, we select the next row and repeat the pixel by the pixel scanning process. We repeat this logic for all the rows in the sensor until we finish

scanning the entire sensing matrix. We repeat the frame scanning process in a continuous data streaming mode, and thus we ensure fixed frame time and fixed frame per second (fps) rate.

An Analog multiplexer module was designed to establish a fast and continuous scanning process from all the sources (rows). The multiplexer is based on transmission gates and amplifiers and allows us to read data from 256 different sources (rows) through a single analog output (Figure 2-11).

The multiplexer features three amplification stages. The first stage is based on an op-amp amplifier in a sample and hold (S&H) configuration (

Figure 2-12). The output terminal of each of the S&H amplifiers is multiplexed through a 16:1 analog multiplexer to the second amplification stage. In total 16 multiplexers are required to deliver the outputs from 256 S&H amplifiers to the second amplification stage. Outputs of the second amplification stage are multiplexed through a 16:1 analog multiplexer to a third and the final unity gain amplifier which pushes the analog signal outside the sensor to the ADC.
In total, we have 256 op-amp amplifiers for the first phase, 16 amplifiers for the second and one for the final amplification phase.

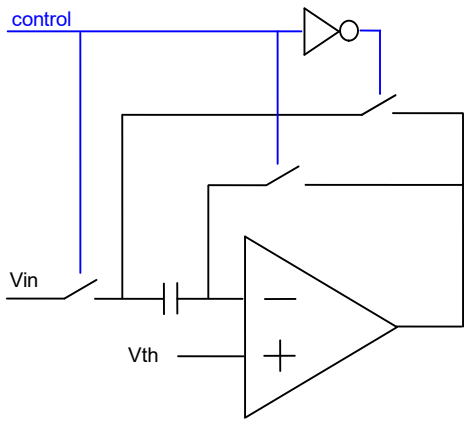


Figure 2-12 - First stage amplifier in S&H configuration

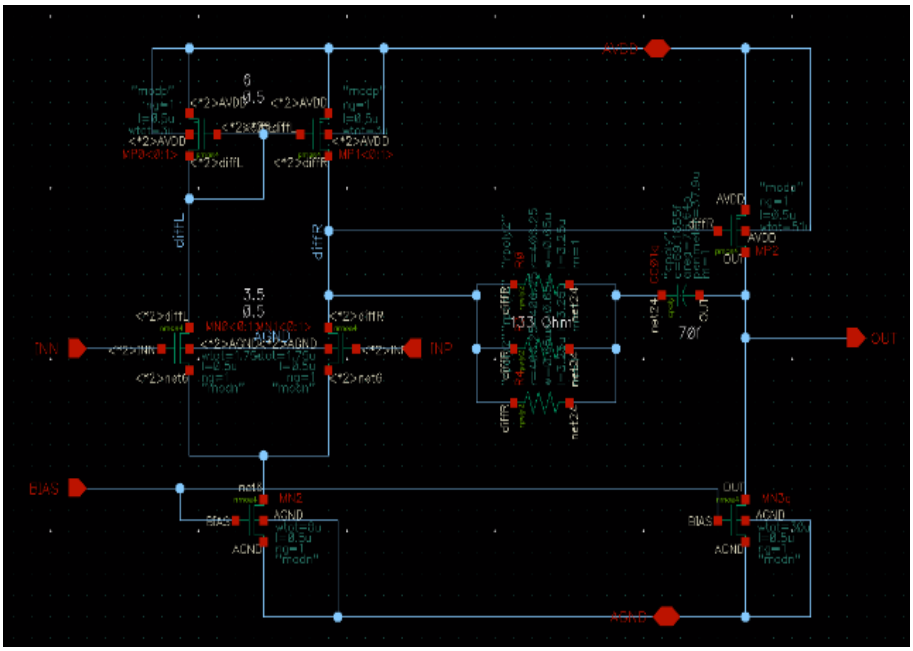


Figure 2-13 - general architecture of the 2nd and 3rd amplification stages

We use 3 stages of amplification to decrease the delay of the analog signal on its way out from the sensor. The delay is caused by the different current driving abilities of the internal/transistor level amplifiers to the required current driver to charge the capacitance of the chip level pad.

In the described readout chain only the first amplifiers have S&H peripheral networks whereas the rest amplification stages (second and third) are connected in a negative feedback configuration with a bandwidth required to deliver the signal in one clock time.

We note:

1) In case of faster design constraints or more complex design, all three amplification stages can be turned into S&H configuration.

2) To ensure a higher frame rate and lower read noise, an internal ADC must be used instead of streaming the analog value outside the sensor for sampling.

3) The reading sequence is synchronized with the Pivot signal, and the pixels are being read and then reset during the same access time window.

2.8. Sensor Operation

To understand the pipeline and the general data flow in the sensor let's consider the sequence of the events which are related to a single pixel and later extend and explain how the same control logic applies to the entire pixel matrix and successfully generates WDR data in mantissa-exponent format.

Let us consider a pixel located on the top left corner of the sensor and has the following coordinates <0,0> (where the first digit represents the rows and the second the columns). After a global pipeline reset, the pivot control signal (from the row decoder), and the column decoder signal are set to select the row and column number 0 (<0,0>). I.e., the values from row decoder and column decoder select a single pixel from the matrix. The pivot control signal enables the connection between the row #0 and hard reset circuitry (reset #0) via the 9-bit demultiplexer, and the column decoder defines a certain pixel in the selected row that will be connected to the reset logic circuitry. Once access is established, the analog value of the pixel is sampled via the readout chain for further ADC steps (the readout process will be explained later). Within the same access window, the pixel is reset by a signal that arrives from the reset #0 circuitry (hard reset), this moment indicates a beginning of the new frame time for this pixel. The column decoder moves to the next column and selects the next pixel for readout and reset purposes. This process is repeated for all the pixels in this row (row # 0), once the first row is completed, the row decoder (Pivot) selects the next row (row #1), and column decoder repeats its cycle from column #0 to the last column. A combination of the row and column decoders enables us to scan

and reset the entire matrix in a continuous mode. I.e., the described operation is the exact operation of a sensor in a rolling shutter mode with a single exposure setup.

To enable the WDR capture, the pipeline must be extended. In parallel with the previously described process, the pipeline performs access to up to another eight pixels for possible updating of the integration times and performing conditional resets (reset # 1-8). Here we explain the signal sequence to generate the first conditional reset.

As explained in section *9-bit Row Decoder*, the Pivot signal and Pivot-1/2 (P-1/2) signal has a delay of an exact half frame time. The delayed control signal (P-1/2) is sent to row driver, 8-bits, and 9-bits multiplexers, these modules function to: activate the read circuitry in the selected row, deliver the comparator result to the conditional reset logic circuitry # 1, and deliver the reset logic result back to the pixel accordingly. Analog value of the selected pixel is being compared with vTh (half of the possible voltage swing over the read line). If this analog value crossed vTh means the pixel will be overexposed by the end of the frame time and must be reset to prevent saturation and data loss (perform a cond. reset # 1). If the analog value from the pixel did not cross the vTh, means the pixel will not be overexposed, and there is no need to restart the light accumulation process (Figure 2-14). Regardless of the decision of the comparator, the pipeline stores the decision in the memory for further access.

The selection of the Row Decoder will stay until the Column Decoder finishes scanning all the pixels in the row. Once the Column Decoder finished its cycle, Row decoder will select the next set of rows.

We continue to check the pixels at different stages of their integration times up to another 7 times for possible conditional reset to prevent their overexposure state by the end of their frame time (Figure 2-15).

Parallel access to up to 9 pixels is done in the following way: As discussed in section (9-bit Row Decoder), the generated control signals have delays of $^1/_2$, $^1/_4$, $^1/_8$, … $^1/_{256}$ of the frame time from the Pivot signals. All the generated delayed signals (after Pivot) have the same purpose, they all activate the required read circuitries (via row driver), and they control the parallel 8 and 9-bit multiplexers that delivers data from comparators to the reset logic circuitries and back to pixels. I.e., in the same way as the P-1/2 signal allows the pipeline to sample pixels at the middle of their integration time and connects them to the correspondent reset circuit #1, P-1/2-1/4 control signal allows us to sample pixels that reached ¾ of their integration time and establish connection with the corresponding reset circuit (reset #2). The rest of the generated control signals has a similar function, e.g., P-1/2-1/4-1/8 signal allows us to sample pixels that reached 7/8 of the integration and so on where the latest delayed signal with a name: P-1/2-1/4-1/8-1/16-1/32-1/64-1/128-1/256 is connecting pixels at 255/256 of their integration time to the last reset logic circuit and back (reset #8). In general, we can select up to 9 different pixels at the same time for possible updating of their integration times.

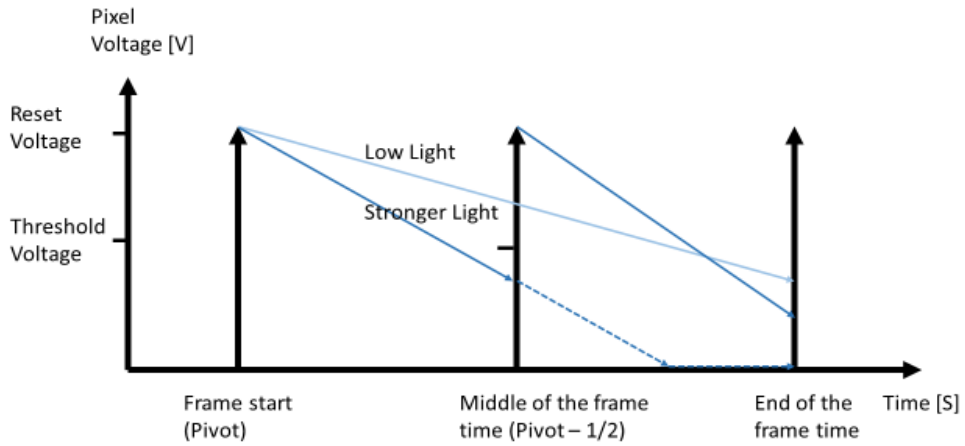


Figure 2-14 – Light blue line represents low light – no conditional reset. Dark blue line represents higher light intensity, conditional reset will be applied.

Inherent parts of the sensor are the memory buffers. The memory is required to store the number of conditional resets for each pixel. The total number of resets is being accumulated in the memory buffers and represents the final exponent value in the discussed mantissa-exponent data structure. Besides the reset accumulator purpose, the memory plays a significant role during the conditional reset #2-8 decision stages. To generate the second and above conditional resets (#2-#8), two conditions must be met. The photodiode in the pixel must be more than 50% discharged (crossed vTh value) which we know from the latched comparator module. The pixel received a conditional reset signal on its previous access phase which we know from the memory buffers. We are using the data from these two sources (memory + comparators) in order to prevent unnecessary conditional resets. E.g., If reset #4 is the last conditional reset the pixel

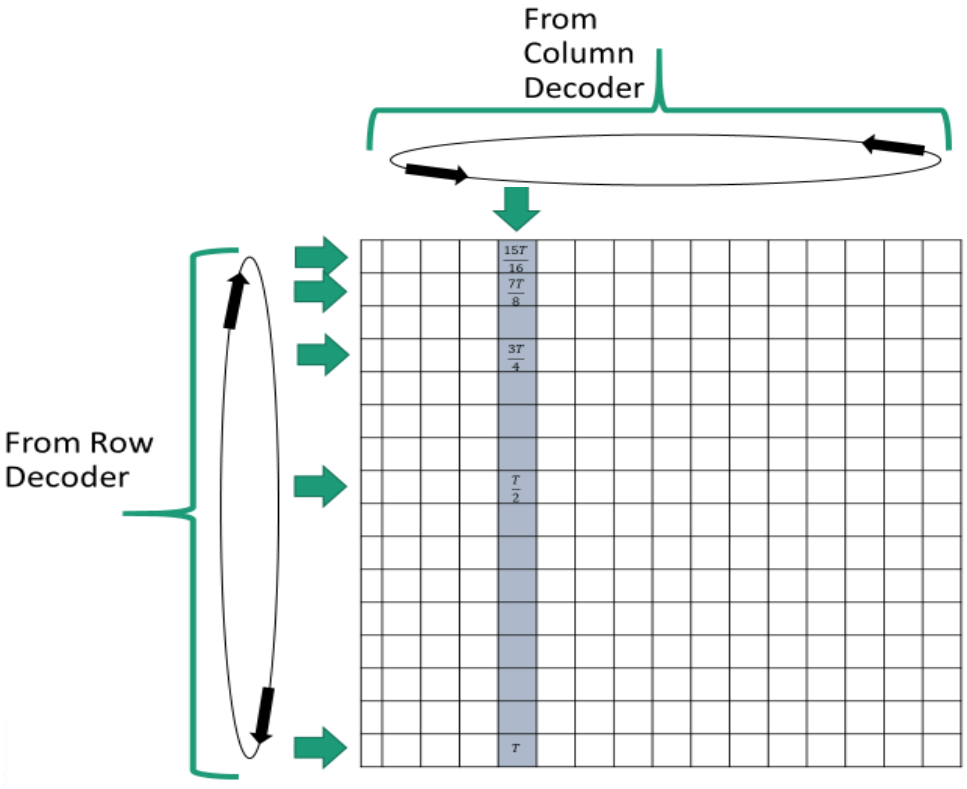


Figure 2-15 : Control signals and the selected pixels for conditional reset. Presented on a 16x16 example matrix

received (Figure 2-16), and there was no reset signal on the next reset phase #5 due to the voltage over the photodiode didn't cross the vTh value. There might be a case when the talked about analog value will cross the vTh at the ascending conditional resets #6-8 access phases. If the reset logic circuits rely only on comparators' output to generate reset signals, they will generate reset signals when the pixel will not be overexposed by the end of the fame time, and no conditional resets should be applied and completely ruin the sensor data output values.

More than that, the row driver module is using the memory data as well to decide to allow read access to the pixel or not. So not only unnecessary reset signals will not be generated, but even the access to the pixel will not be granted if there is no need in resets, so the pixel will be left untouched until the end of the frame time without being reset and accessed.

Finally, here we explained the exact purpose of the reset data (from the memory buffers) being delivered to the conditional reset circuitries #2-8 and to row driver modules.

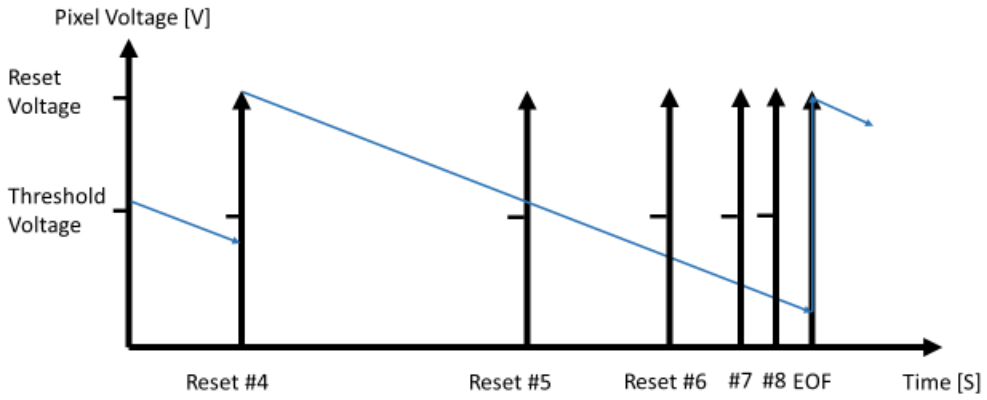


Figure 2-16 - 4th reset is the last one the pixel received, at reset #5 the pixel didn't cross the vTh value and thus at reset #6-#8 stages the pixel must stay untouched

It is important to discuss the dimensions of the memory buffers and their type. We are taking advantage of storing the exponent values only and the rolling shutter mode to design a sensor which takes much less memory compared to storing a quantized value (after ADC) in the

same rolling shutter mode or global shutter sensors. In global shutter sensors, to generate the exponent part for this representation, we will need a memory buffer with similar dimensions to the sensing matrix whereas in rolling shutter mode we are taking advantage of the fixed intervals between the accessed pixels (based on the theory of time-space trade-offs [46]) and the fact that we need only a few bits (1-4) to store the exponent values only for comparisons and for final data representation instead of the storing the values after ADC (12 or more bits per pixel)[23] significantly reduce the amount of embedded memory. E.g., the total number of pixels the pipeline will scan between the conditional reset #1 and #2 will always be fixed and equal to the ¼ of the sensing matrix. I.e., to store the generated data at reset #1 stage and until the data is required on reset #2, we can use FIFO shift register with fixed dimensions of the ¼ frame (Figure 2-17). Besides the length, first FIFO will have only one-bit in width to store the decision of the conditional reset logic. The second and the third FIFOs will have 2-bit in widths (that will allow them to store binary values of "10" and "11") with correspondent lengths of 1/8 and 1/16 parts of the frame and so on. The final FIFO needs to carry up to 4 bits (to store binary value of 8 "1000") , and its length will be 1/256 of the frame sensor size. A complete FIFO structure is presented in Figure 2-17.

In this work, the sensor size is 256×360 (0.09MP), and the required amount of memory is 56.7Kbit. For a 1024 x 1280 (1.3MP) sensor, the required memory will be ~1.06Mbit. Regardless of the rolling/global shutter types of operations, the presented mantissa-exponent data representation significantly reduces the embedded memory compared to other multiple-reset WDR sensors/systems as discussed in the Introduction section. This allows for the miniaturization and integration into systems where size and costs are key factors, E.g., in biomedical or security applications.

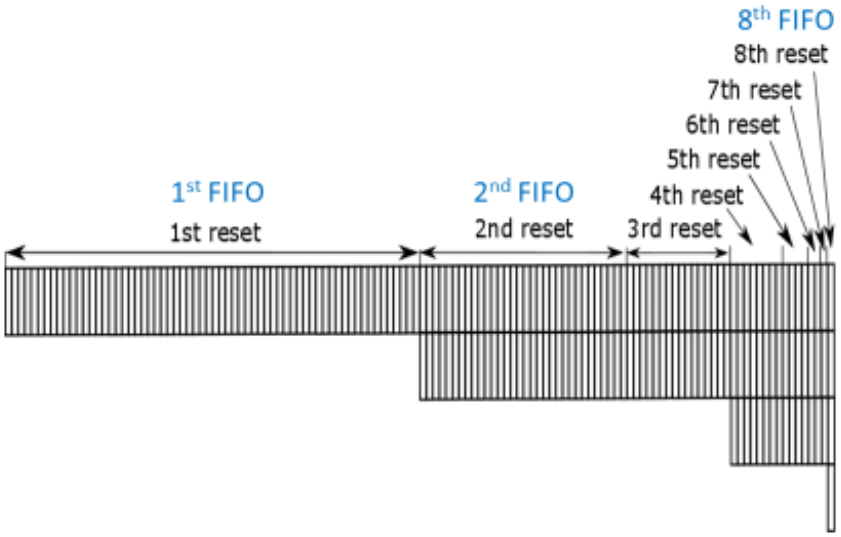


Figure 2-17 - Integrated altogether FIFO buffers with their sizing visualization according to the reset data they carry

In the following part, we will present the designed hardware testing system to communicate with the sensor and stream all the necessary digital signals to establish a real-time WDR video stream to a PC or monitor via a DVI interface.

Chapter 3 **: M-E SENSOR TESTING SYSTEM**

To establish a video stream from the sensor to a screen or a PC, a set of analog and digital signals must be provided and captured to and from the sensor respectively. All the digital signals such as clocks, data to the sensor, ADC, memory, and streaming the data to a DVI or a PC are handled by the HDL driver synthesized on Xilinx FPGA ML605 development platform. The interface between the FPGA development platform and the sensor was established by a custom designed PCB. In this chapter will present the design of the testing system and the achieved results.

3.1. Interface Board:

A printed circuit board (PCB) was designed (Figure 3-1) to set up a complete interface between the presented sensor and the FPGA with the following peripherals:

1) 6 Configurable low dropout regulators (LDO) – to provide the required analog voltages to the internal modules in the sensor such as amplifiers, comparators, etc.
2) Pin grid array (PGA) socket to connect the sensor physically and electrically with the designed board.
3) 12-bit ADC with a sampling rate of 60Msps
4) Differential driver to create a differential signal from a single analog output from the sensor. The differential signal provided to ADC increases its SNR abilities.
5) Level shifters – to match digital IO sensor voltages with the values required by the FPGA.

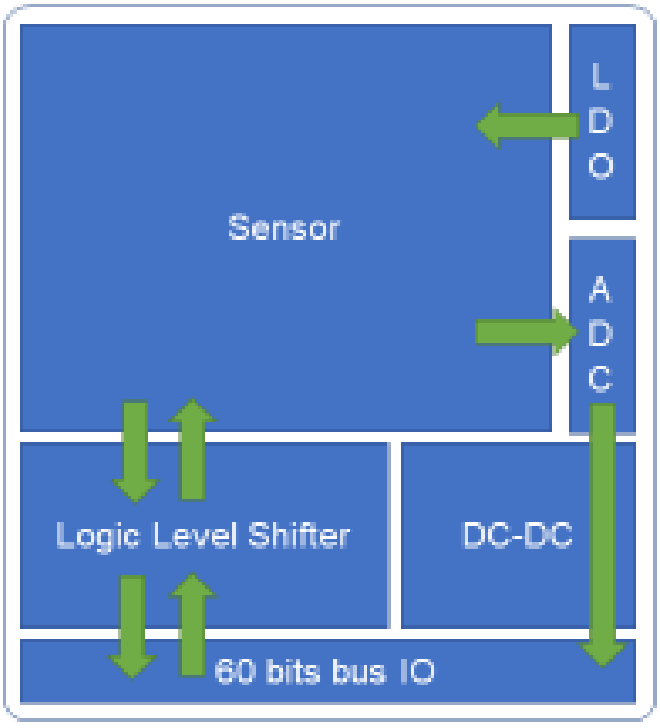


Figure 3-1 - Modular representation of the designed PCB

6) DC-DC chain– to provide voltages to all components from a single 12V input. The chain supplies the board with 4 different voltages: 5V for the ADC, 3.3V analog & digital for the sensor, level shifters and ADC, and 1.8V for level shifters.

A complete schematics and PCB layout are presented in 7.1: Appendix: PCB Schematics

3.2. HDL Controller

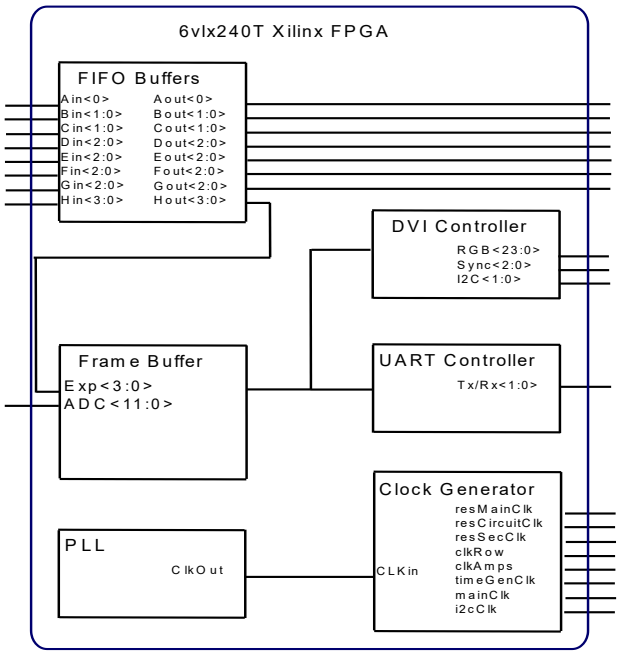


Figure 3-2 - HDL sensor driver in modular view

To provide a complete operation of the sensor, a set of digital signals like synchronized clocks and direct interface with memory buffers must be provided. Along with the signals required by the sensor, the controller handles the onboard ADC, embed and controls the frame buffer, and streams the picture to digital video interface (DVI) or a PC.

The sensor HDL driver includes the following main modules: FIFO buffers, frame buffer, clock generator, DVI controller, UART controller, and PLL.

The FIFO buffers purpose is to store the exponent (number of resets) values generated by the sensor and to stream these values back (size and number required FIFO buffers are explained in the previous - Sensor Operation section.). Only the values from the final buffer (H_{out}) are later combined with the values received from ADC and represents the final value of the pixel's light intensity in a mantissa-exponent format where the mantissa is the output from the ADC, and the exponent is the output from the last FIFO buffer. The combined value is stored in a full duplex frame buffer. A frame buffer is designed to allow read and write simultaneously from different 2-dimensional locations and with different frequencies, so the data populates the buffer with the frequency it arrives from the sensor independent of the read with a frequency is as required by the DVI or UART interface. The data from the frame buffer is then redirected either to the monitor via DVI or to a PC via UART. DVI and UART controllers both feature the required by the interface physical layers (PHY) to meet the communication standards on the receiving side. Another essential module is the clock generator. This module provides 7 different clocks to the sensor, ADC, DVI chip, and to some of the internal modules. The code is presented in HDL Driver Top-Level Source Code and the internal modules can be provided by request

3.3. M-E sensor Results

The proposed image sensor design was plugged into a designated PGA socket on the designed PCB, and the entire system was activated by a Xilinx ML 605 FPGA development kit, the designed HDL driver took control over the sensor and the required peripherals to capture and stream the data to a monitor or a PC. To capture the image data from the FPGA via UART interface (COM port), a dedicated software was designed on a C# programming language using a Visual Studio (VS) software development tool (Figure 3-3). The presented software is able to reconstruct the image data from the input stream based on the synchronization signals from the

FPGA. Bottom two pictures are showing not synchronized data, and the top pair are showing synchronized exponent and the mantissa data on two separate images.

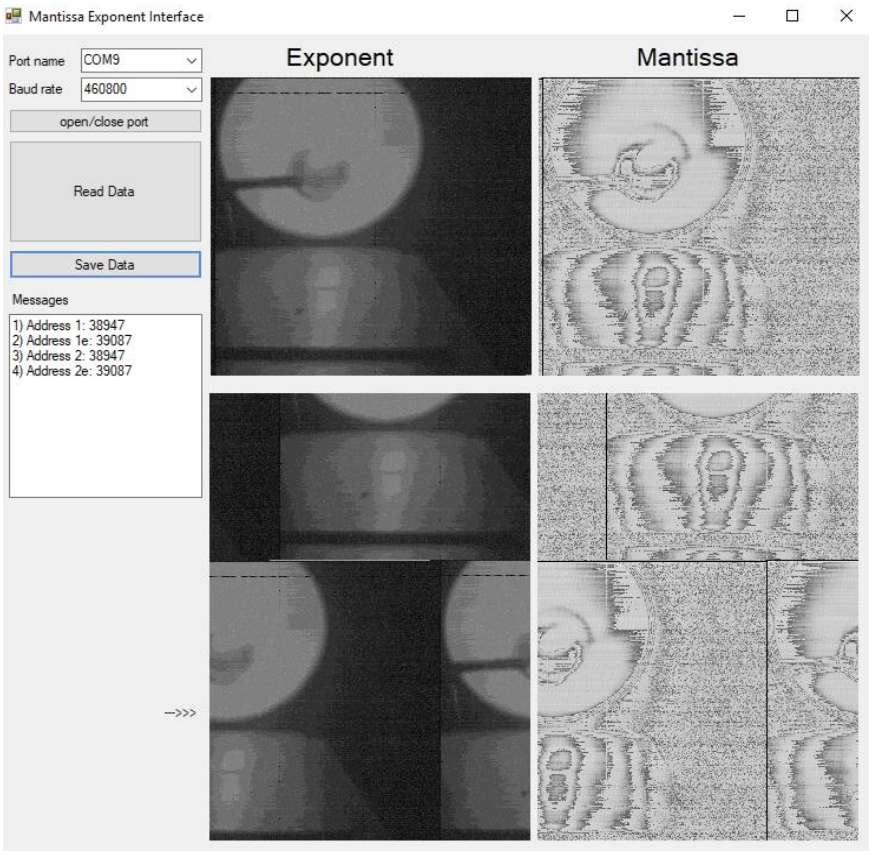


Figure 3-3 : PC software to capture and synchronize the image data stream from FPGA

To measure the sensor response to the light and the dynamic range the following experiment was set up (Figure 3-4). A controllable 200W light source was set up 1m from the designed sensor and from the light power meter (PM100usb). In this experiment, we can control the light intensity, see the output of the sensor and measure the received by the sensor light intensity by measuring its power.

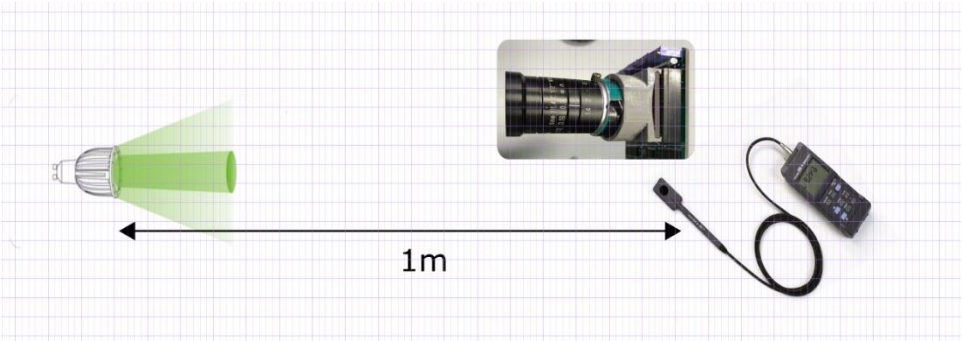


Figure 3-4 : Experiment setup

Since the rendering devices aren't able to present WDR data, the output from the image sensor is presented separately on two different images. The first image (Figure 3-6) shows the mantissa which is the output of the ADC and represents a detail level for each of the exponents. The second image (on the right) shows the exponent values – where the darker pixels represent the lower exponent values and the brightest the higher exponent values. The separation allows seeing the conditional reset performances along with the detail levels represented by the mantissa. We can see the successful generation of conditional resets and general performance of the designed pipeline. Even though the sensor shows a successful generation conditional resets based as a function of the received light intensity, it is lacking to detect objects in low light due to high noise levels and significant leakage current (Figure 3-6). The Mantissa shows all the details captured from the second exposure mostly when the exponent equal to 2 in most of its regions. More than that, the leakage current is strong enough to generate the first and the second conditional resets automatically when the light intensity is about 2uW (room light), and this has a direct impact on the dynamic range of the sensor. Considering we have only 6 of the 8 conditional resets instead of 8 and the data from the ADC has 10 valid bits, the total achieved

dynamic range of the sensor can be estimated as following (Equation I), with total resulting dynamic range of 96 dB ($dynamic\ range = 20\log_{10}(2^{10} * 2^{6}) = 96dB$).

Mantissa Exponent

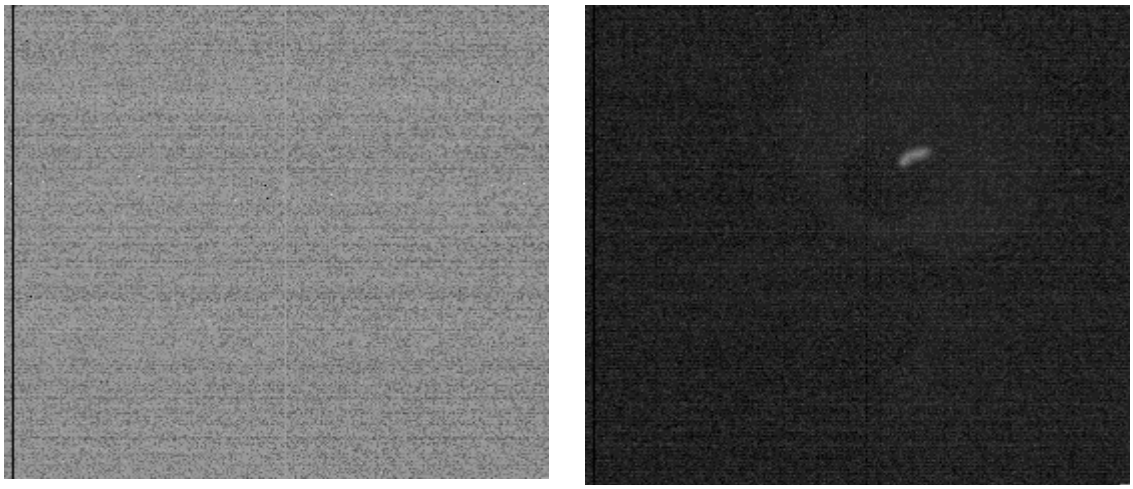


Figure 3-6 - lowest possible detection, ~ 2uW of light intensity generate first conditional reset (Mantissa on the left, Exponent on the right)

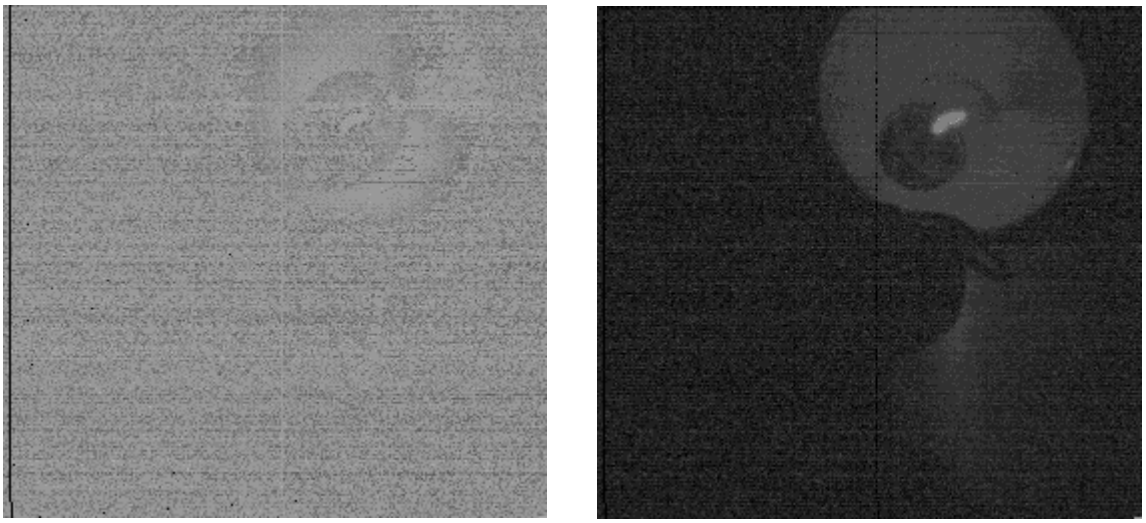


Figure 3-5 - Power of ~50 uW, Exponent has 3 values, mantissa shows the details for higher exponent values

Stronger light with 50uW of power (Figure 3-5) causes more conditional resets. Each brightness region in the exponent image represents a different exponent value where the gradient values in the mantissa image represent the detail levels. Here again, low light objects are

impossible to detect due to the high dark current, but for the third exponent region, we can see brightness gradient in the mantissa image.

For a stronger light with 430 uW of power (Figure 3-7), we can observe a higher number of generated conditional resets by the pipeline. Up to 5 conditional resets are generated and appear on the exponent image (higher number of conditional resets represent higher exponent value) and accordingly for each of the exponent, we can observe the gradient level represented by the mantissa.

Finally, for the DR measurement and comparison with single exposure DSLR camera, we set up two light emitting objects with different light intensity as presented in Figure 3-9Figure 3-8. A lamp (on the right) is emitting light with 1.3 mW of power and a phone on the left which emits light with about 1 uW of power. The brightness of the phone is about 1000 times lower than the brightness of the lamp. Even though the lamp is behind the phone and slightly out of focus, it is possible to see the impact of the strong light on the integration times of the affected pixels (representing the lamp) which reach the maximum of possible conditional resets (8) and

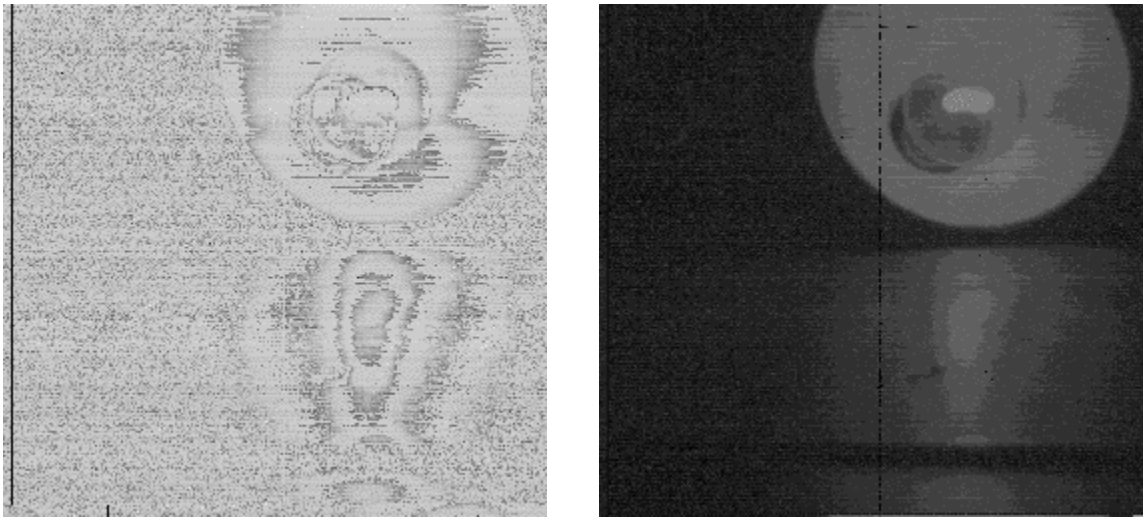


Figure 3-7 - Light with ~430uW of power, 5 different exponent values appear, detail levels for each exponent also appear and dominant over the noise

on the pixels showing the phone where the light from the phone can generate only one conditional reset (above the existing two generated by the leakage) (Figure 3-8).

For comparison, the same scene was captured using a DSLR camera with manual settings (Figure 3-9). The image on the left was integrated 1/50 sec whereas the image on the right was integrated 1/2000 second. This difference represents a ratio of ~6 EV which is exactly the number of stops our sensor is able to generate.

Unfortunately, the leakage current is strong enough to saturate the detail levels of the low light objects completely but, the details of the bright objects are still possible to detect due to the low impact of the dark current on short integration times.

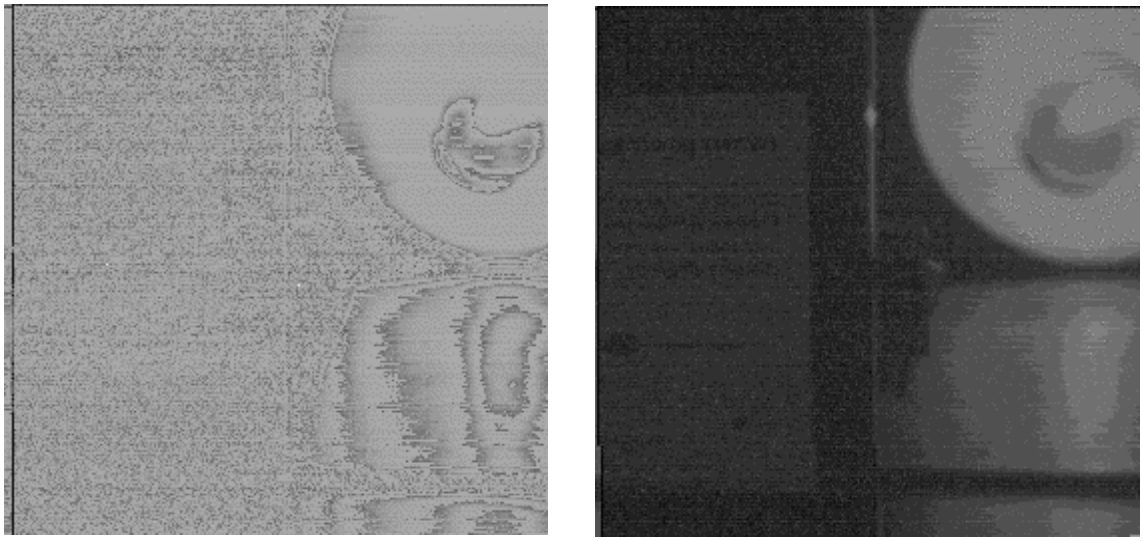


Figure 3-8. - low light object on the left (~1uW), Strong light on the right (1.3mW), and mantissa is lacking presenting details of the low light object

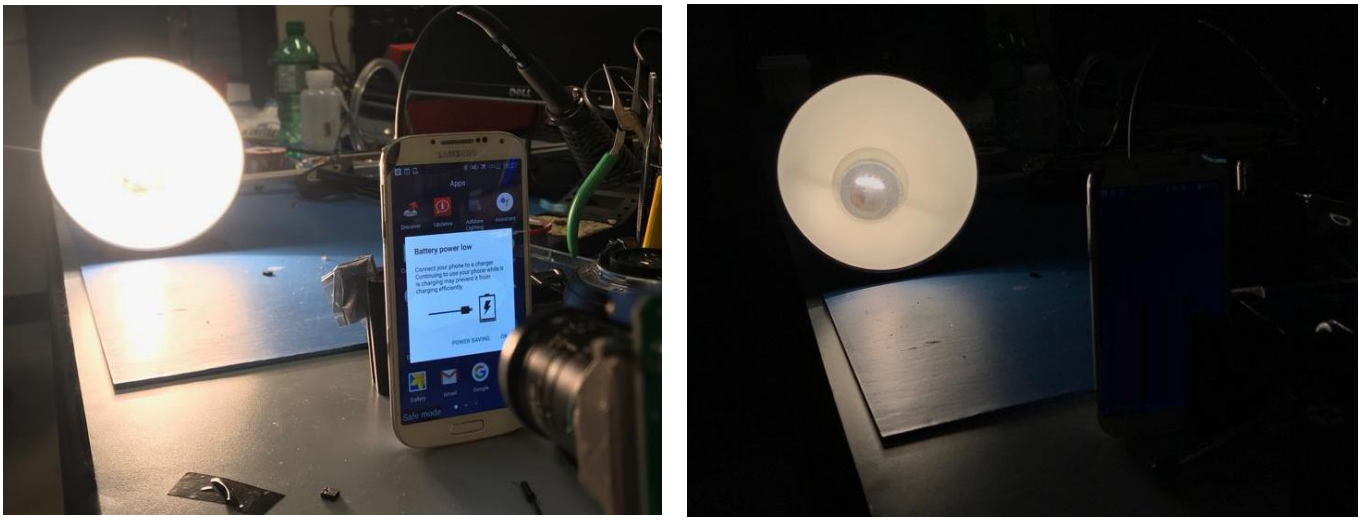


Figure 3-9 - The same scene captured with different EV setup.

In this chapter, we presented a complete hardware system including WDR sensor, interface board to activate the sensor and HDL driver to take control over all the processes in the system. The presented a WDR system that updates the integration time directly during the scene acquisition process and thus has a significant advantage compared to multiple capture systems or logarithm sensors for the following reasons.

- Multiple capture systems require multiple frame buffers, i.e. memory to store a few exposures before generating WDR data. Also, they are prone to generate image lag due to a relatively bigger time difference between exposures.
- Logarithmic sensors are prone to lose contrast in bright areas due to logarithmic response whereas in our case the WDR data is always linear.

Possible modifications are required to improve the visual performance of the sensor, and especially to increase the dynamic range into the low end.

- Multiple access pixel has to be redesigned as the existing configuration inserts a destructive level of noise into the pixel and reduces its performance in low light conditions.
- Internal ADC must be integrated with the image sensor to read the analog signal internally.
- The drain terminal of the reset transistor has to be connected to the highest positive voltage instead of being implemented a logic gate to eliminate a leakage current into the photodiode.
- Pinned photodiode must be considered as a replacement for an existing solution.

Possible improvements to the pipeline are:

- Inserting a decision module based on a numeric value stored in the memory buffer at the end of FIFO will allow streaming a single bit to all reset logic circuitries which will equalize the propagation delay of all reset circuits and reduce the interface with the buffers from 38 bits to 15.
- With pinned photodiode, a row parallel scanning technique must be implemented to reduce the decoupling noise on the non-accessed pixels.

In the next chapters, we will discuss a possible solution for visual WDR data rendering on LDR devices.

Chapter 4 **TONE MAPPING**

Once we obtain the WDR data either by a WDR sensor or by any other technique, the next step will be presenting this data on an LDR display. As described in section 1.2, the brightness intensities of the WDR data can significantly exceed the dynamic range of the displays; therefore, a to present WDR on any display device a special compression is required which is called Tone Mapping (TM) or Tone Operator (TMO). Without compression, we will have to select a brightness region to display, and this will cause to either underexposed, either overexposed objects to appear while presenting a selected range from WDR data a display.

Figure 4-1: Manually picked brightness intensities from WDR data

Many TMOs been developed for the last few decades and have different visual and RT performance (as described in section 1.2). The goal of our team was to create a TMO which will take advantage of the mantissa-exponent output data format (from the presented image sensor) for tone mapping efficiency and quality purposes. The TMO has to have improved visual performance compared to previously designed by our group solutions [47], [48] and be able to be easily implemented on a hardware platform which considers low footprint, low power consumption, and real-time performance.

In this chapter, we first present the TM algorithm developed by Dr. Alain Horé (not published resource) operating on a mantissa-exponent format along with its visual performance comparisons to related work and later we present its hardware implementation (part of this thesis research work) with synthesis results and comparisons to the model developed in MATLAB.

4.1. Tone-mapping algorithm for mantissa-exponent format

The proposed by Dr. Alain Horé algorithm for tone mapping assumes that the output intensity of each pixel x (p) is described as follows:

Equation 4-1 $$\boldsymbol{x(p) = m(p) * 2^{e(p)}}$$

Where $m(p)$ is the mantissa part which varies between 0 and 1 with a precision of 12 bits, and $e(p)$ (the exponents' part) which varies between 0 and 8. The proposed tone mapping operator for this data type has three main components: a base-2 logarithm (Equation 4-3), linear scaling (Equation 4-4), and contrast enhancement (Equation 4-5). We note: equation 4-3 is derived using the decomposition of the natural logarithm by using a second order Taylor approximation as present on Equation 4-2.

Equation 4-2 $$\boldsymbol{log_e x = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(x-1)^n,\ x \in [0,1)}$$

Equation 4-3 $$\boldsymbol{f(p) = log_2 x(p) = \frac{m(p)-1-\frac{(m(p)-1)^2}{2}}{log_e 2} + err(p)}$$

Equation 4-4 $$\boldsymbol{h(p) = d_{max} * \frac{f(p)-f_{min}}{f_{max}-f_{min}}}$$

Equation 4-5 $$\boldsymbol{y(p) = |2 * h(p) - h * l_{LPF}(p)|}$$

The first step is a global operator applied to all the pixels' intensities to compress their dynamic range based on a logarithmic curve. The next linear scaling step is performed to match the output values of the logarithm operator to the LDR values of the rendering devices (screens,

printers, TVs, etc.) for which the maximal pixel intensity value is given by d_{max}. The last contrast enhancement step is required to reduce the blur generated by the global compression performed with the log operator, thus making the output images sharper. This contrast enhancement step, called sharp masking, uses a low-pass filter (a Gaussian filter in our experiments) denoted as l_{LPF}, which aims at emphasizing the high frequencies of an image by subtracting low frequency component from the signal. On the Figure 4-2 and Figure 4-3, we can compare the visual performance the presented TMO with comparisons to related works [31], [34], [49], [50] in terms of low light object enhancement, artifacts, and overall naturality preservation of the scene.

Figure 4-2 : Visual comparisons the presented TMO with Drago, Durand, Fattal and Mantiuk

Figure 4-3 : Visual comparisons the presented TMO with Drago, Durand, Fattal and Mantiuk

4.2. Hardware Implementation

In the proposed design, we use a real-time data flow to make the design fast and timed (real time), for further possible integration with the image sensor. To reduce the pipeline delay and the design footprint, we use common to the entire frame parameters, like minimum and maximum intensities, computed during the processing of a previous frame on the current frame and so on. If we use global parameters from the previous frame, we don't need to store a full frame in the memory for any parameter extractions, which enables us to save a lot of resources and significantly shorten the pipeline. This is possible because these values don't change drastically between two subsequent frames. From 30 fps and above (TMO was tested at 60 and 120 fps), the difference of these parameters between two frames can be assumed to be small enough to be simply set to 0 without affecting the quality of the images. The overview of our

design is presented in Figure 4-4 and is described as follows: the mantissa and exponent values of each pixel arrive at every clock cycle and feed the tone mapping algorithm into the block f(p). This block has a defined constant latency of 3 clocks cycles. After 3 clocks, f(p) ends the calculation of the first input and will then give successive outputs at each clock. The output of f(p) goes to module h(p), which has a defined latency of 17 clocks. After the first output of h(p) is generated, this module will give successive outputs at each clock. Each of the output of h (p) is sent to the last module y (p), which has a latency of 2066 clocks. The reason for a big pipeline delay is discussed in section - Pixels allocation module. Overall, we have a latency of 2088 clocks for getting the first tone mapped pixel, and then successive pixels are output at each clock. We note that for simulations the bit-width for each input pixel is 24, where 16 bits are used for the mantissa and 8 bits for the exponent. The bit-width of each output pixel of the tone mapped image is an 8-bit integer value.

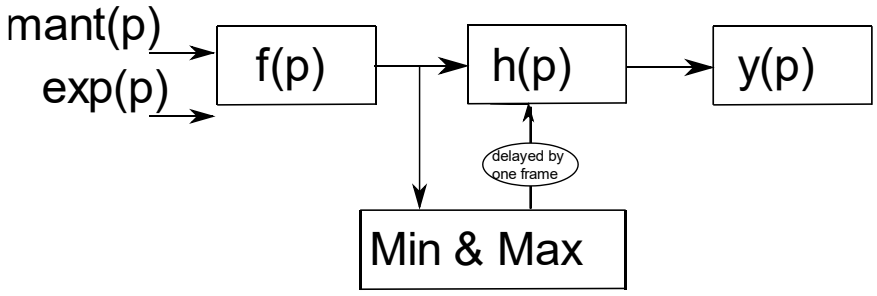


Figure 4-4 – Block diagram of the TMO

4.2.1 Log Computation module

The purpose of this module is to compute Equation 4-3. This process normally requires complicated arithmetic operations, which are described below, but we introduce a simplified method for the execution of this module.

Discarding signed operations

As mentioned earlier, $m(p)$ varies from 0 to 1. Thus, on subtracting 1 from $m(p)$, we get a negative result, which means that we need to perform signed operations. Also, when performing the power of 2 operations for two signed inputs, bigger multiplier must be used for computing the result. Also, when subtracting $\frac{(\mathrm{m(p)}-1)^2}{2}$ from $(m(p) - 1)$, the absolute value of the minuend is bigger than the subtrahend, and thus we get another signed operation. All these calculations require wider buses, more computational effort and more time to be executed. This complexity is not suitable for our goal. Consequently, we want to avoid this complexity by using unsigned operations in order to realize a small and efficient design. Indeed, it is easier to operate with unsigned values during development, verification and maintenance processes. Thus, using a few arithmetical operations, we rewrite Equation 4-3 and get the following equation:

Equation 4-6 $$\boldsymbol{f(p) = log_2 x(p) = \left(2m(p) - \frac{(m(p))^2}{2}\right)\left(1 + \frac{1}{2} - \frac{1}{16}\right) + e(p) + err}$$

Where the constant $err = -\frac{69}{32}$. It is easy to notice that we get fewer signed operations and the value of $2m(p) - \frac{(m(p))^2}{2}$ is always positive.

Division by a constant

Dividing by $\log_e(2)$ is an expensive operation and requires of using a hardware divider. Thus, instead of division by a constant, we perform multiplication by approximating the value of 1/log (2) as shown in Equation 4-7 with approximation error of O(3)

Equation 4-7 $$\boldsymbol{\frac{1}{log(2)} = 1 + \frac{1}{2} - \frac{1}{16} + O(3);\ O(3) = 5.2e - 3}$$

Since the approximation is a sum of power of 2 elements, bit selection (shift operation) with summing operator can be preferred over a division. Using this technique, we get the result of the

division in only 1 clock. The architecture of the division module is represented in Figure 4-5 between latency 1 and 2. With the optimization described, the latency for computing Equation 4-6 is only 3 clocks@100Mhz. Equation 4-6

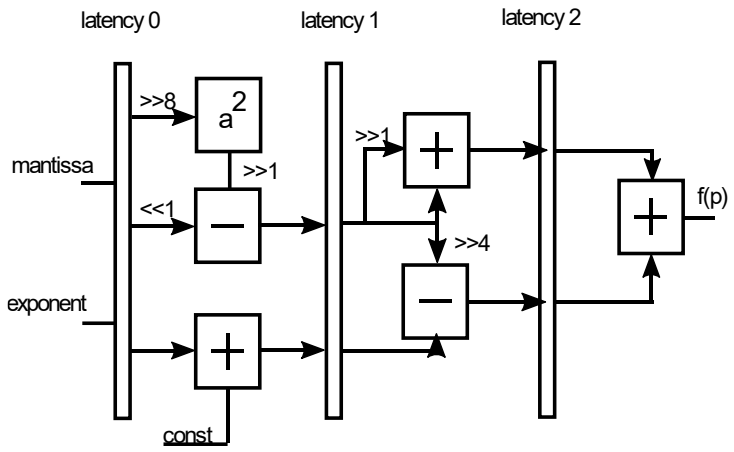


Figure 4-5 - Block f (p): pipeline data flow

4.2.2 Linear Scaling Module

This module, h (p) block, scales the outputs from the f (p) block to an 8-bit representation (here, we assume that the maximal intensity of the display is d_{max}=255. This module has 3 inputs, the minimum and maximum values of brightness calculated and saved from the previous frame, and the pixel intensity of the current frame. Based on this information, a linear scaling is performed according to Equation 4-4. In this block, we use a built-in pipeline DSP divider to estimate the result in real-time. In order to compute more accurate results of h(p), we first perform a lossless 8-bit left shift on the input data of the block (this approximates the multiplication by d_{max}, and then we perform the division. This reduces the division error generated by the divider. The digital divider outputs the quotient and the remainder without a fractional part. Therefore, on performing the multiplication before division, the quotient

maintains more precision, as opposed to performing multiplication after division. Overall, the proposed design for the linear scaling module as shown in Figure 4-6.

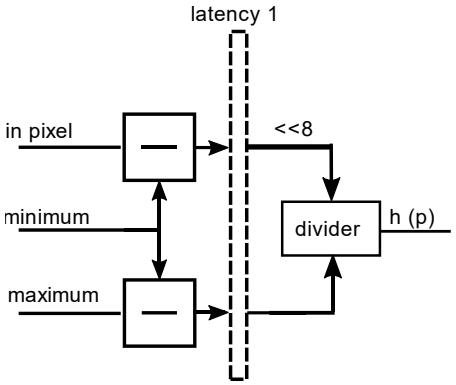


Figure 4-6 - Block h (p) - pipeline data flow

4.2.3 Contrast Enhancement Implementation

The result from the previous block (linear scaling) enables us to highlight some details in tone mapped images, but they may also contain some noticeable blur. To tackle this issue, a contrast enhancement filter Equation 4-5 is applied. This module comprises of the hardware implementation of the contrast enhancement filter (Figure 4-8) and the hardware implementation of the block needed to allocate the required pixels from the input stream (Figure 4-7).

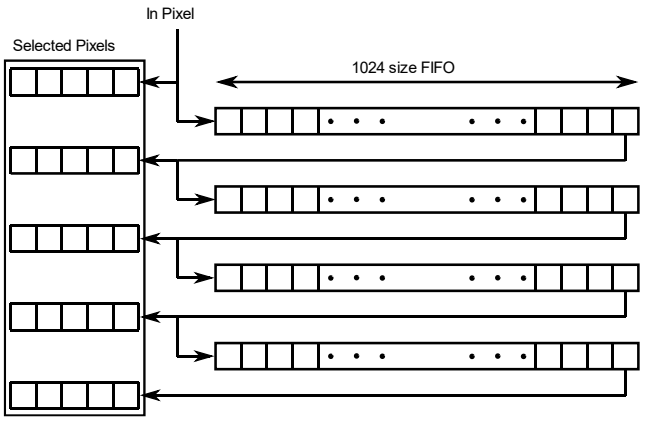


Figure 4-7 - Allocation of a 5x5 matrix of pixels by using 4 FIFOs

Pixels allocation module

Since the pixels are read out line by line, it is required to store a few lines of the image to implement the 2D filter and to access the vertically located pixels at the same time. The number of stored lines equals the smallest dimension of the 2D filter minus one. In our case the filter is symmetric, and its dimension is 5 by5 pixels, and consequently, the number of stored rows is 4. This storage is implemented by 4 FIFO shift registers. With this method, at each clock, we allocate the appropriate window of pixels for performing convolution. The delay generated by this module depends on the image and the filter size, and it is given by Equation 4-8

Equation 4-8 $$\boldsymbol{Pipeline\ Delay} = \boldsymbol{FIFOsize} * \left\lfloor \frac{\boldsymbol{Filter\ size}}{\mathbf{2}} \right\rfloor * \left\lceil \frac{\boldsymbol{Filter\ size}}{\mathbf{2}} \right\rceil$$

In our case, the delay is equal to $1024 * \left\lfloor \frac{5}{2} \right\rfloor * \left\lceil \frac{5}{2} \right\rceil = 2051\ clocks$. We note that filter size is the smallest dimension of the 2D filter used and FIFO size is the number of pixels in a row/line of the image. The implementation presented on Figure 4-7.

Contrast enhancement module

This part presents the hardware implementation of Equation 4-5. We perform high-pass filtering (HPF) by subtracting the result of the convolution of the allocated window with a low pass Gaussian filter, from the same allocated window (given by the previous block). The high-pass filtered image (with high-frequency content) is added to the original image, which increases the overall contrast. An absolute value operator is applied to ensure that the positive values are always obtained during the contrast enhancement step.

The block diagram of the filter module is presented in Figure 4-8. On the first step, we perform convolution by multiplying all the 25 selected fragments based on the filter coefficients.

The multiplications are performed by using shift and add operations, which is an efficient strategy in terms of power consumption and hardware resources. All the products obtained after multiplications are summed up, and the result is divided by 256 by using an 8-bit logical right shift operator (256 is the denominator of the filter used). On the second step, an absolute operation is performed to eliminate the negative values after the filtering process. The implementation of the absolute operation was done by using one comparator, two adders (performing two subtractions) and one multiplexer.

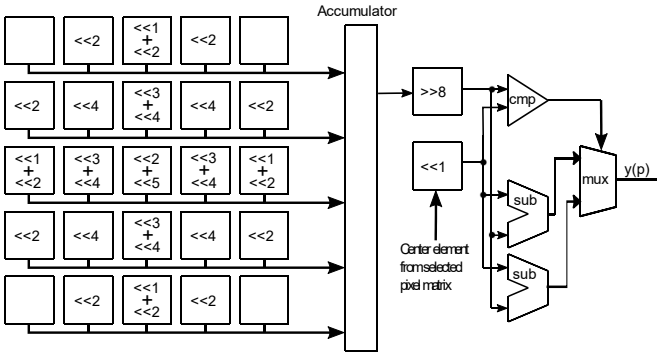


Figure 4-8 - Implementation of the contrast enhancement filtering

4.3. Simulation and Synthesis Results

4.3.1 Simulation results

To determine the quality of the presented hardware implementation, we compare it with a software implementation in MATLAB by using subjective and objective evaluations. In Figure 4-9, we show four tone mapped images obtained with hardware design and with the MATLAB implementation. As we can notice, the tone mapped images look very similar and quite indistinguishable, which indicates that the hardware implementation is reliable. To confirm that objectively, we have used two well-known image quality measures: the peak signal-to-noise ratio

(PSNR) and the structural similarity image index (SSIM) [51], [52]. In general, high values of the PSNR (in theory, the PSNR varies from 0 to infinity) give an indication that two images might be very similar. The two images can be deemed similar when the SSIM value, which varies from 0 to 1, is close to 1. In Table 4-1, we compare the PSNR and SSIM results between the MATLAB implementation and the hardware implementation. As we can notice, we have high values of the PSNR and the SSIM. The average PSNR is 55.87 dB, and the average SSIM is 0.9996. These values confirm that the images obtained are similar. We note that in general, with PSNR above 30 dB the difference between images will not be noticeable, and our results are quite satisfying when compared to other hardware TM implementations [27], [41], [53] with PSNR of 30, 40 and, 55 dB accordingly.

	TMQI			PSNR	ssim
	Q	S	N		
Hercules cave	0.9674	0.9288	0.8969	54.63	0.9998
Lausanne	0.9348	0.9581	0.6342	54.47	0.9998
MtTamWest	0.8467	0.8411	0.3099	53.91	0.9996
Needle	0.6197	0.3945	0.0291	56.72	0.9995

Table 4-1- PSNR and SSIM results

4.3.2 Synthesis results

Our design was synthesized on Altera Cyclone III FPGA. The power consumption was provided by a development tool estimation and is 149.5 mW, where part of the power is consumed as static power by the chip peripheral and is dissipated as the I/O thermal power. In fact, the dynamic power of our design (that also has been estimated by the simulation tool) is only 19.84 mW.

FIFO shift registers have been implemented using embedded memory. The exact amount of memory used is $FIFO_{length}$ x $FIFO_{width}$ x number of FIFOs. In our case, we have 32.7 Kbits. In general, we have a very small and efficient design. Compared to other on-chip/GPU image processing designs [31], [54] with 177mW and 18W respectively and especially to other TM FPGA designs [25], [55] with 37W and 22W respectively, our implementation can be a good competitor mainly because of the speed, hardware-efficiency, quality, and the total pipeline delay. Table 4-2 summarizes the power consumption and the hardware resources of our design where the static consumption can be significantly reduced by choosing smaller FPGA or integrating the design with the CIS.

Chip level used		Total power Consumption	149.5 mW
Total logic elements	4020	Core dynamic Power	19.84 mW
Total registers	3031	Core static power	99.18 mW
Total memory used	33 kB	I/O power	30.5 mW

Table 4-2 - Hardware resources summary

4.4. Tone mapping algorithm results conclusion

In this part, we have presented a tone mapping algorithm that can process WDR images based on a mantissa-exponent representation. The algorithm has been efficiently implemented in FPGA. Indeed, we have obtained low power consumption, high processing speed, and small footprint. Also, we have been able to obtain similar tone mapped images between the hardware implementation and a software implementation of our algorithm, which was also confirmed objectively by using the PSNR and the SSIM. Based on the reliability of our hardware

implementation and the good performance obtained, we have good confidence that we can embed our algorithm and an imager on the same chip.

In the next part, we describe the designed real-time testing system for a TMOs with a mantissa-exponent data input format.

Figure 4-9 - Tone mapped images. Software implementation with Matlab (Left), Hardware implementation (Right).

Chapter 5 : TONE MAPPING TESTING PLATFORM

In this part, we present a platform to perform visual and performance tests for algorithms based on mantissa-exponent (floating point) representation in a real-time mode. Standard WDR sensors provide the brightness values in fixed-point format (RAW) which is usually a direct output from the ADC whereas the M-E sensor streams the data in floating-point representation and special TMO is required to process this output (see 4.1). The simulation system is based on a standard WDR sensor but gives the ability to test variable TM algorithms suitable for direct integration with sensors with M-E output.

Figure 5-1 - Hardware system overview: WDR Image sensor on the left, development FPGA board in the middle and DVI PHY on the left

5.1. FPGA Platform for M-E algorithms

To capture with data and present the results of the TM algorithms in real time mode, we need a peripheral system which manages and provide all necessary peripheral signals to the camera and display modules along with the TM algorithm and preprocessing steps. These features Cyclone III Altera FPGA with WDR Aptina image sensor (MT9M034) and Terasic extension board with DVI PHY as represented in Figure 5-1. The WDR sensor is manually configured to provide the maximum dynamic range of 120dB. Data flow in this system is the following: Generated WDR video data streamed from the sensor to the FPGA, The data is

preprocessed to meet the requirements of the tone-mapping algorithm, and the data is compressed by the tested tone mapping algorithm and streamed out to DVI.

Here we explain the hardware design of all the pre-processing steps made on the input data to meet the constraints of the based on a mantissa-exponent tone mapping algorithm. The steps are including the decompression of the data arrived from the image sensor, reconstructing a full-color image from incomplete color samples (CFA to RGB), extracting the brightness intensities (RGB to YUV), and converting the data into floating point format (Figure 5-2).

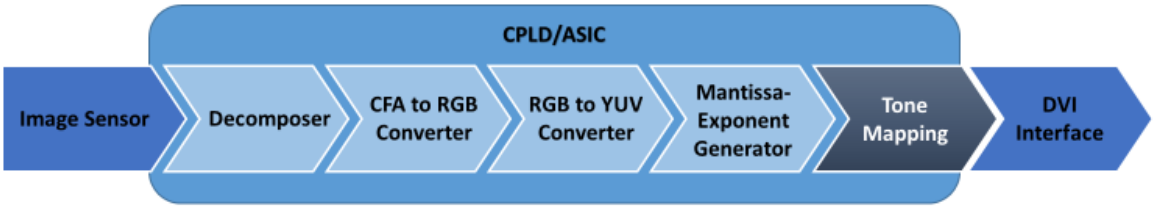


Figure 5-2 - Pre-processing steps to stream WDR data to tested TMO

5.1.1 Decomposer

The image sensor is programmed to provide 120dB of dynamic range, and this requires an allocation of 20-bits per pixel. However, the data is compressed and streamed out via 12-bit bus, and thus a hardware decompression module is required to retrieve the data.

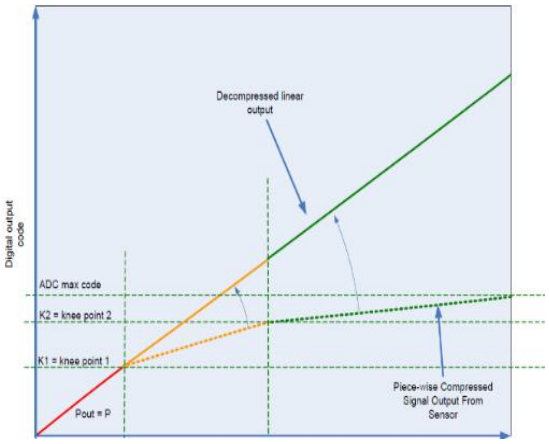


Figure 5-3 - Signal response to light intensity

Figure 5-3 and Figure 5-4 show possible DR & compressions as a function of the ratio of T1/T2 and T2/T3. Based on the data sheet, the compression works as follows: the brightness values from 0 to 2048 are not compressed and represented as is in the output register (let's call it range 1). The brightness values from 2048 to 2^16 (65535) are compressed linearly into the range between 2049 and 3040 (let's call it range 2). And the last range of brightness values start from

T1/T2 Exposure Ratio (R1) R0x3082[3:2]	P1	P_{OUT}^{1} = P1	P2	P_{OUT2} = (P2 - P1)/ (R1* 4)+ P_{OUT1}	T2/T3 Exposure Ratio (R2) R0x3082[5:4]	P_{MAX}	$P_{OUT_{MAX}}$ = (P_{MAX} - P2)/ (R1*R2*4) + P_{OUT2}
4x	2^{11}	2048	2^{14}	2944	4x	2^{16}	3712
					8x	2^{17}	3840
					16x	2^{18}	3904
8x	2^{11}	2048	2^{15}	3008	4x	2^{17}	3776
					8x	2^{18}	3904
					16x	2^{19}	3968
16x	2^{11}	2048	2^{16}	3040	4x	2^{18}	3808
					8x	2^{19}	3936
					16x	2^{20}	4000

Figure 5-4 - List of possible DR and compressions

65536 to 2^20 (1M), are also compressed linearly but represented in the output register as discrete values between 3041 and 4000 (let's call it range 3). Once we know the compression, we can define the decompression piecewise equations -

Equation 5-1

Equation 5-1
$$\boldsymbol{out(p)} = \begin{cases} \boldsymbol{p}; & \boldsymbol{p} \in [\mathbf{0}, \mathbf{2^{11}}] \\ \boldsymbol{(p - 2048) * 64} & \boldsymbol{+ 2^{11}}; \ \boldsymbol{p} \in (\mathbf{2^{11}}, \mathbf{2^{16}}] \\ \boldsymbol{(p - 3040) * 1024 + 2^{16}}; & \boldsymbol{p} \in (\mathbf{2^{16}}, \mathbf{2^{20}}] \end{cases}$$

The pipeline of the decompression module is presented in Figure 5-5. The overall decompressing process latency is 2 clocks. On the first clock, we perform a comparison of the input value with P1 and P2 were the result of the comparators is bonded and used to select the value to subtract from inP. And on the second clock cycle, we are selecting the multiplicand and the added values to the result from the previous cycle depending on the same bonded values from the comparators. Finally, the result after the summation represents WDR data in 20-bits per pixel.

5.1.2 CFA to RGB Converter

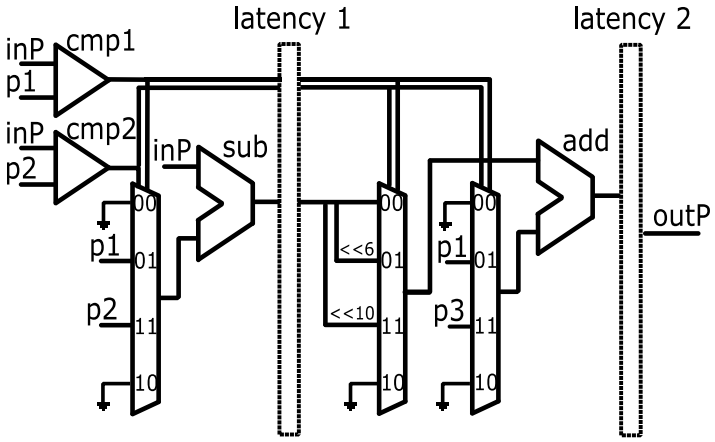


Figure 5-5 - Decomposer module, pipeline view

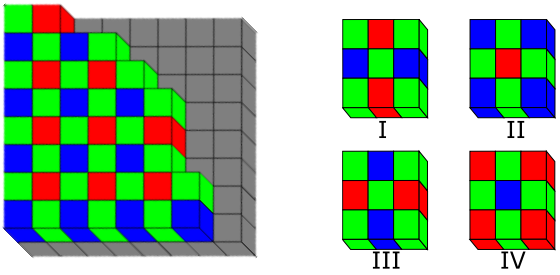


Figure 5-6 – CFA (on the left) and its 4 patterns (on the right)

The data from the color sensor arrives in CFA format, and demosaicing is required to reconstruct the colors from the samples output.

The transformation is not linear and is based on the regional location of the selection on the Bayer RGBG matrix. Every single phase, we know only one color, whereas the other two colors are calculated based on the neighborhood pixels. In total, we have 4 different cases (Figure 5-6) in the CFA matrix, and during the color estimation process, only one color is known whereas the other two must be estimated. I.e., the first case (I). The central pixel is green; and so, the green value is known, but blue and red values must be calculated based on the adjacent blue and red pixels accordingly, the same logic is applied for the rest of the three CFA patterns, but different colors must be reconstructed.

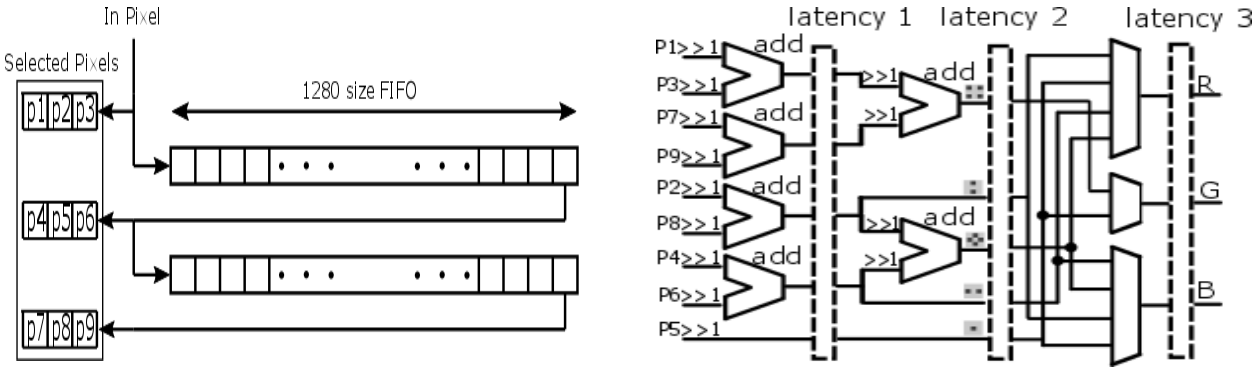


Figure 5-7 - Matrix allocation module (left), Bayer cases calculation (right)

The implemented module consists of 2 main modules as presented in Figure 5-7. The first module is '3×3 Matrix Extractor' allocates 3 by 3 matrix of adjacent pixels from a continuous pixel video stream. This module requires 2 FIFO shift registers. The delay generated by this module can be calculated by Equation 5-2. In our case, the FIFO size equals 1280 elements, and the filter size equals 3. Thus the overall delay generated by this module equals 1282 clocks. The second module is the 'Color Extractor' module which selects the appropriate mash based on the location and performs the conversion of the current 3 × 3 CFA matrix into RGB channels.

Equation 5-2 $$\boldsymbol{delay = FIFO\_size} * \left\lfloor \frac{\boldsymbol{filter\_size}}{\mathbf{2}} \right\rfloor + \left\lceil \frac{\boldsymbol{filter\_size}}{\mathbf{2}} \right\rceil$$

This module works in continuous mode, and every clock generates 3 colors (red, green and blue). The calculation of all the cases is done in parallel and depending on the location of the incoming pixels in the image steam a required case is selected for each of the color channels. The transformation from CFA to RGB representation can be done using any of the possible masks with different interpolation types and sizes [56].

5.1.3 RGB to YUV Converter

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Figure 5-8 - Transformation matrices between YUV and RGB color spaces

The next step is to extract the luma (Brightness) channel from the RGB representation. The luma is one of the channels in Y`UV color space where Y represent the luma, and the UV is a chrominance constituent of the image. The YUV color space is defined as a linear coordinate transformation from an RGB color space (Figure 5-8). To calculate the Y` we need to multiply the red channel by 0.299, green channel by 0.587, blue channel by 0.144 and sum the results. The calculation above can be done using 3 DSP multipliers and adders. But there is a more efficient way to estimate the required value, and we present this technique in our design. Let's consider for example the multiplication of the red channel by 0.299. We can rewrite the value of 0.299 using a sum of a power of two numbers with acceptable precision as presented in Equation **5-3**. This approximation gives us a 0.2% error of the original value. However, the error can be reduced by using more and smaller than 10 bits of precision.

Equation 5-3 $$\mathbf{0.299} \approx \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{512} + \frac{1}{1024} = \mathbf{0.2998046875}$$

Equation 5-4 $$\mathbf{0.299} * \boldsymbol{R} \approx \mathbf{0.2998}..* \boldsymbol{R} = \frac{\boldsymbol{R}}{\mathbf{4}} + \frac{\boldsymbol{R}}{\mathbf{32}} + \frac{\boldsymbol{R}}{\mathbf{64}} + \frac{\boldsymbol{R}}{\mathbf{512}} + \frac{\boldsymbol{R}}{\mathbf{1024}}$$

Equation 5-5 $$\mathbf{0.587} * \boldsymbol{G} \approx \mathbf{0.5869}..* \boldsymbol{G} = \frac{\boldsymbol{G}}{\mathbf{2}} + \frac{\boldsymbol{G}}{\mathbf{16}} + \frac{\boldsymbol{G}}{\mathbf{64}} + \frac{\boldsymbol{G}}{\mathbf{128}} + \frac{\boldsymbol{G}}{\mathbf{1024}}$$

Equation 5-6 $$\mathbf{0.114} * \boldsymbol{B} \approx \mathbf{0.1142}..* \boldsymbol{B} = \frac{\boldsymbol{B}}{\mathbf{16}} + \frac{\boldsymbol{B}}{\mathbf{32}} + \frac{\boldsymbol{B}}{\mathbf{64}} + \frac{\boldsymbol{B}}{\mathbf{256}}$$

Now, we can rewrite the multiplication as presented in Equation 5-4. It is easy to follow, the required value of 0.299×R is a sum divided by 2(right shifted) values of R, which are very easy to be implemented on hardware platforms. The same approximation is made to the result of 0.587*G and 0.114*B, as presented in Equation 5-5 and Equation 5-6. For faster design and area efficiency, the shifting operations are performed using bit selection and concatenation operator instead of using a shift register. Also, since the adding sequence will not affect the result, the size of each addend was considered during the pair selection for each adder. I.e., as we have different size addends during the summation, the similar sized ones were selected to be a pair. For example, R/1024 with G/1024 became a pair since their bit width result will be smaller than the result of G/1024 with, i.e. G/128 which has a different size. The overall block latency is 4 clocks, as we use 2 input adders with 1 clock latency and 14 addends, we need $\lceil log_2 14 \rceil$ clocks to calculate the value of Y`. The other parts of the Y'UV color matrix (U and V) are not calculated since there is no use in these channels.

5.2. Mantissa-Exponent Generator

The final step is to create mantissa-exponent (M-E) representation from fixed point 20bit integer (

Equation **5-7**). The integer values varying from 1 to 1M into 16 bits M-E representation, where its first 12 bits represent mantissa values varying from 0.5 to 1, and the remaining 4 bits are allocated to represent the exponent which can vary from 0 to 15. In our case, the maximum exponent is set to 10 since we already covered the sensor dynamic range and there is no need to

make the design more complicated. We note the full scale of the representation gives us the ability to represent up to $20 \times \log_{10}(2^{12} \times 2^{16}) = 162$ dB of dynamic range.

Equation 5-7 $$\boldsymbol{luminance = mantissa * 2^{exponnent}}$$

The conversion is working as follows. The M-E representation must have a one-to-one correspondence with the integer representation, and thus the mantissa' values must vary between 0.5 and .999 (the precision is based on the number of bits predefined to represent the mantissa) and the exponent values must be strictly based on the range of the luminance values. I.e., if the luminance value is between 2^9 and $2^{10} - 1$, the exponent value can be only 10.

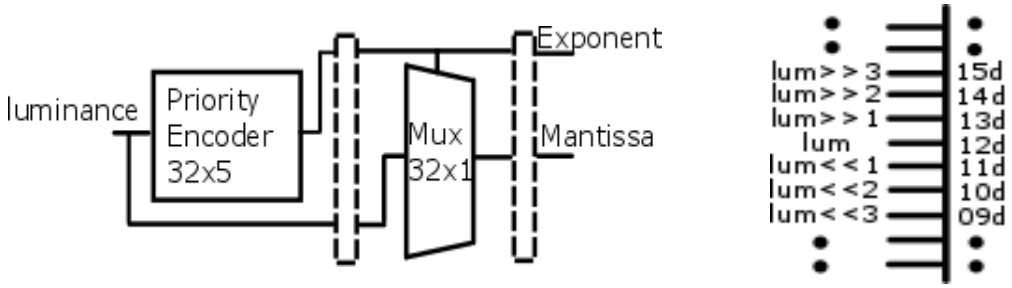


Figure 5-9 - Floating point generator

Hardware implementation of this module is made in the following way; on the first stage, we define the range of the luminance using priority encoder. This scale the luminance range based on the MSB and represents the value of the exponent. On the next clock cycle, the exponent value is used to control the multiplexer which selects the shifted value of the luminance to be forwarded to form the final value of the mantissa. The block is designed to generate mantissa-exponent representation within two clock cycles. This module can support in a maximum of 20 bits mantissa with the same 4 bits of the exponent for better precision and without affecting the calculation timing of 8.3ns per clock.

5.3. Results

Indeed, the design was successfully synthesized on Altera Cyclone 3 FPGA and created the required data representation from the input data video stream. Synthesis results are presented in Table 5-1. Further, the data was sent to the designed TMO for compression and after that streamed out to an 8-bit LCD monitor (Figure 5-11 and Figure 5-10). The overall testing frequency was set to 108MHz (provided by the FPGA and measured by scope) which allowed to stream video with 1.25MP (1280×960) resolution in a continuous real-time mode at 60fps.

Module	Pipeline depth [Cycles]	Memory allotted[kB]	Logic elements	Registers	Pins	Generated Error [PSNR]
Camera Driver	-	**4.096**	**450**	**272**	**47**	-
Decompressor	**2**	**0.08**	**91**	**54**	**38**	**None**
CFA to RGB	**1255**	**55**	**433**	**399**	**86**	**120.4**
RGB to YUV	**4**	-	**286**	**240**	**126**	**90.3**
M/E Generator	**2**	-	**202**	**60**	**46**	**56.18**
Total	**1263**	**63.272**	**1462**	**1025**	-	-

Table 5-1 - testing system synthesis results

Figure 5-10 - Two different exposures of wide dynamic range scene captured with DSLR camera

Figure 5-11 - Tone mapped WDR scene

Chapter 6 **: CONCLUSIONS**

In this thesis, two different parts of the WDR imaging platform has been presented. The first part presents a sensor for capturing WDR scenes whereas the second part presents a hardware implementation of an algorithm to display this data. For both parts, hardware platforms were developed to test and analyze visual and real-time performance.

6.1. WDR Sensor

The presented sensor, with 96 dB (see: M-E sensor Results) of dynamic range, was successfully designed and manufactured (Appendix: Mantissa – Exponent Sensor Die). The concept of the integration time multiplexing based on a time and space tradeoffs with up to 8 conditional resets to capture WDR scenes was integrated with the light acquisition process using a multi-dimensional pipeline technique and successfully generates an output directly in a mantissa-exponent format (without converting it into a fixed-point representation). Regardless to the achieved dynamic range, the out format allows us to stream a significantly larger range of values using the same number of bits and enables the embedding of the sensor output directly with floating point arithmetic modules for TMOs. Low amount of embedded memory to capture WDR scenes was achieved combined with minimized interface makes this sensor suitable for medical and low dimensions WDR imaging applications.

The developed supporting system based on a Xilinx development kit along with the designed interface board has a wide digital interface with the tested sensor and features 12-bit ADC with multiple analog references voltages which allows testing sensors that will be developed in the future and to stream the data via DVI or UART for analyses or presentations.

6.1.1 Further work

To reduce the leakage current and to increase the performance in low light conditions a new variation of pixel design, based on a pinned photodiode, must be developed and components such as ADC and FIFOs that are implemented on a peripheral system to reduce ASIC prototyping costs must be integrated with the sensor.

6.2. Tone mapping

The suggested hardware implementation for the proposed TMO compress WDR data and provide output in a real-time mode. The design generates images similar to a 64-bit software implementation tone mapped by MATLAB, which was also confirmed objectively by using the PSNR and the SSIM. We have obtained low power consumption, high processing speed and a small footprint which allows integrating this design with the image sensor.

A dedicated system was designed to perform visual and performance tests on this type of TMOs in real-time mode. This testing system emulates an output from an image sensor with 120 dB of dynamic range in a half precision floating point format and allows us to test many hardware-implemented floating-point representation tone mapping algorithms.

Additional to the achieved quality and small footprint, this algorithm is based on a global operator (logarithmic) which is prone to lose contrast in bright regions and therefore, required a contrast enhancement module. More complex TMOs, based on a local or a combination of local and global operators must be considered as an optional solution to provide better image quality after the compression.

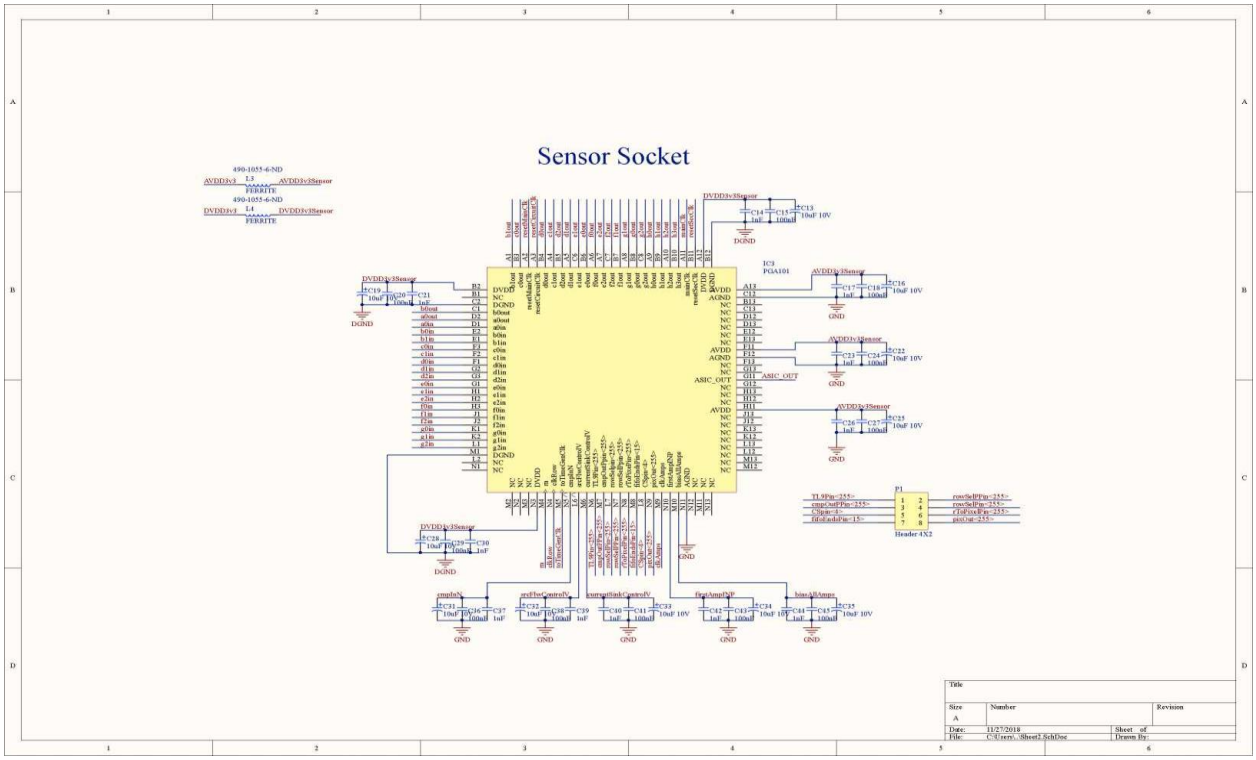
6.3. Summary

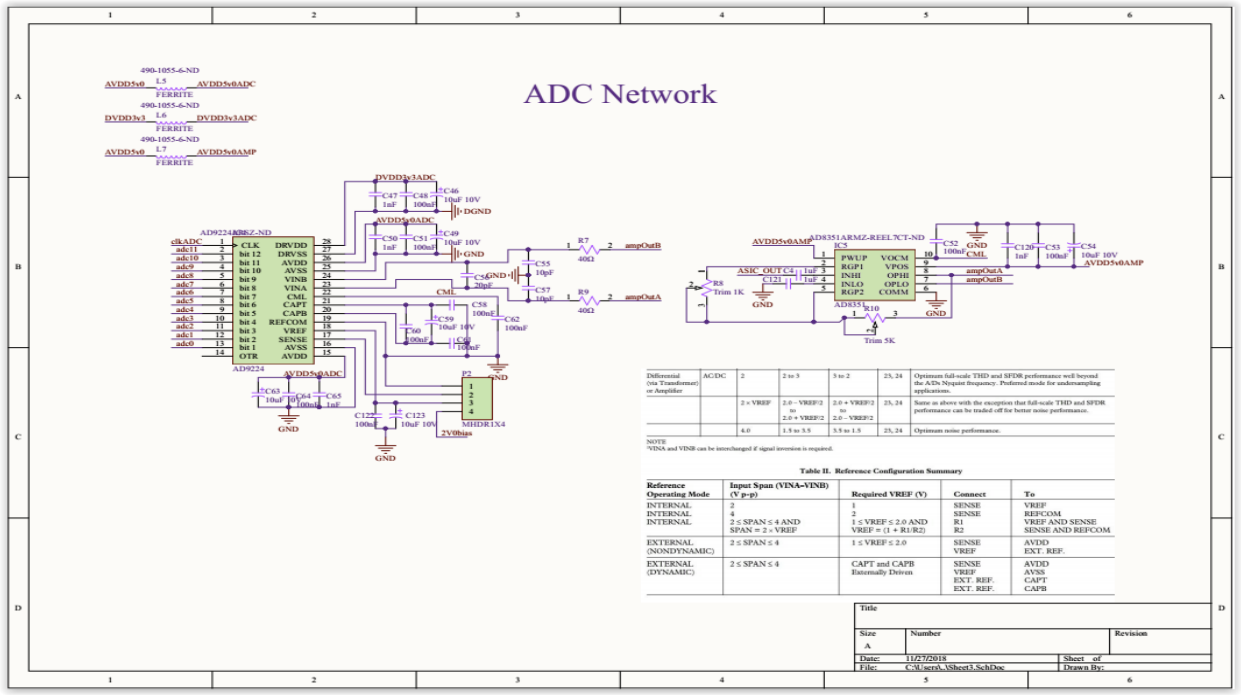
In this work, we presented solutions for capturing and rendering imaging/video system based on a mantissa-exponent representation. The designed image sensor is unique in its way of working as it needs a minimal amount of memory to generate a WDR output. It, also, has a linear response function and the output format is in floating-point representation. The sensor has some limitations in low light conditions, as was discovered during empirical tests, and requires improvements, as described earlier.

After a successful capture, the data can be compressed by the suggested TMO and presented on an LDR device.

Chapter 7 **: APPENDIX**

7.1. PCB Schematics





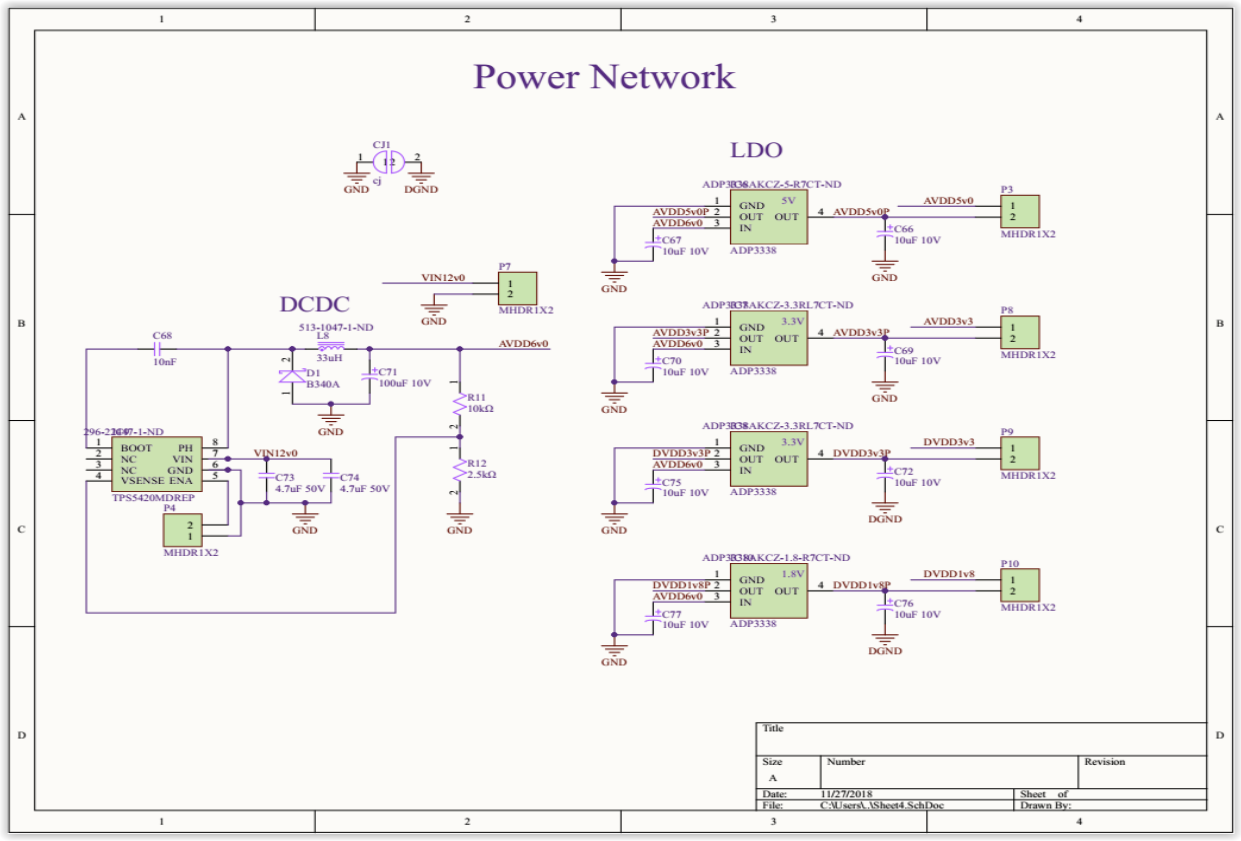
Differential (via Transformer) or Amplifier	AC/DC	2	2 to 3	3 to 2	23, 24	Optimum full-scale THD and SFDR performance well beyond the A/D's Nyquist frequency. Preferred mode for undersampling applications.
		2 × VREF	2.0 – VREF/2 to 2.0 + VREF/2	2.0 + VREF/2 to 2.0 – VREF/2	23, 24	Same as above with the exception that full-scale THD and SFDR performance can be traded off for better noise performance.
		4.0	1.5 to 3.5	3.5 to 1.5	23, 24	Optimum noise performance.

NOTE
VINA and VINB can be interchanged if signal inversion is required.

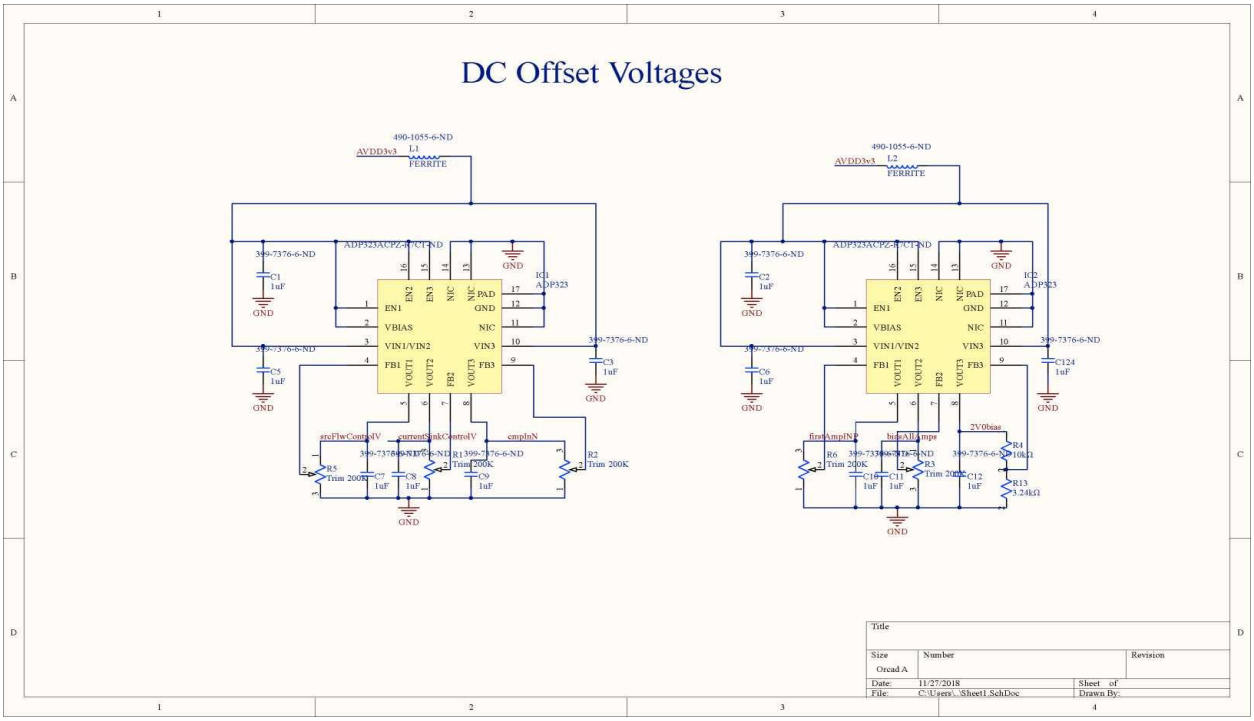
Table II. Reference Configuration Summary

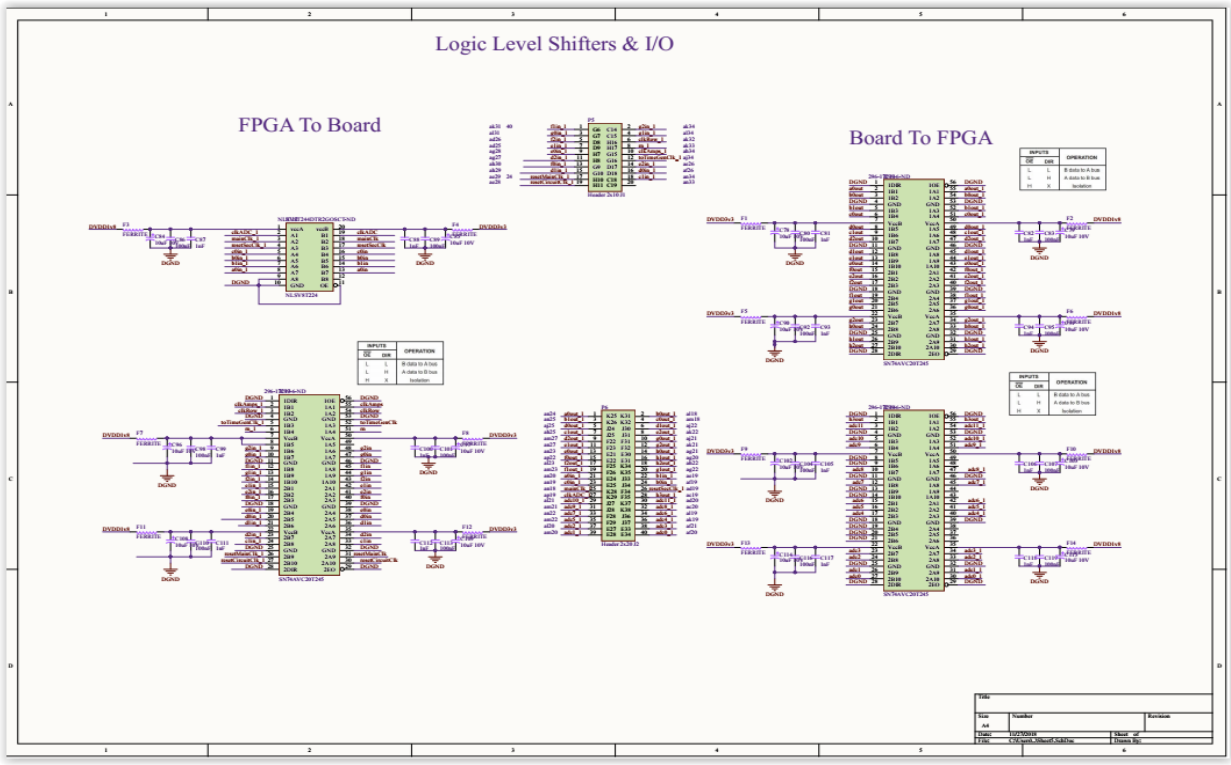
Reference Operating Mode	Input Span (VINA–VINB) (V p-p)	Required VREF (V)	Connect	To
INTERNAL INTERNAL INTERNAL	2 4 2 ≤ SPAN ≤ 4 AND SPAN = 2 × VREF	1 2 1 ≤ VREF ≤ 2.0 AND VREF = (1 + R1/R2)	SENSE SENSE R1 R2	VREF REFCOM VREF AND SENSE SENSE AND REFCOM
EXTERNAL (NONDYNAMIC)	2 ≤ SPAN ≤ 4	1 ≤ VREF ≤ 2.0	SENSE VREF	AVDD EXT. REF.
EXTERNAL (DYNAMIC)	2 ≤ SPAN ≤ 4	CAPT and CAPB Externally Driven	SENSE VREF EXT. REF. EXT. REF.	AVDD AVSS CAPT CAPB

Title		
Size: A	Number	Revision
Date: 11/27/2018		Sheet of
File: C:\Users\..\Sheet3.SchDoc		Drawn By:

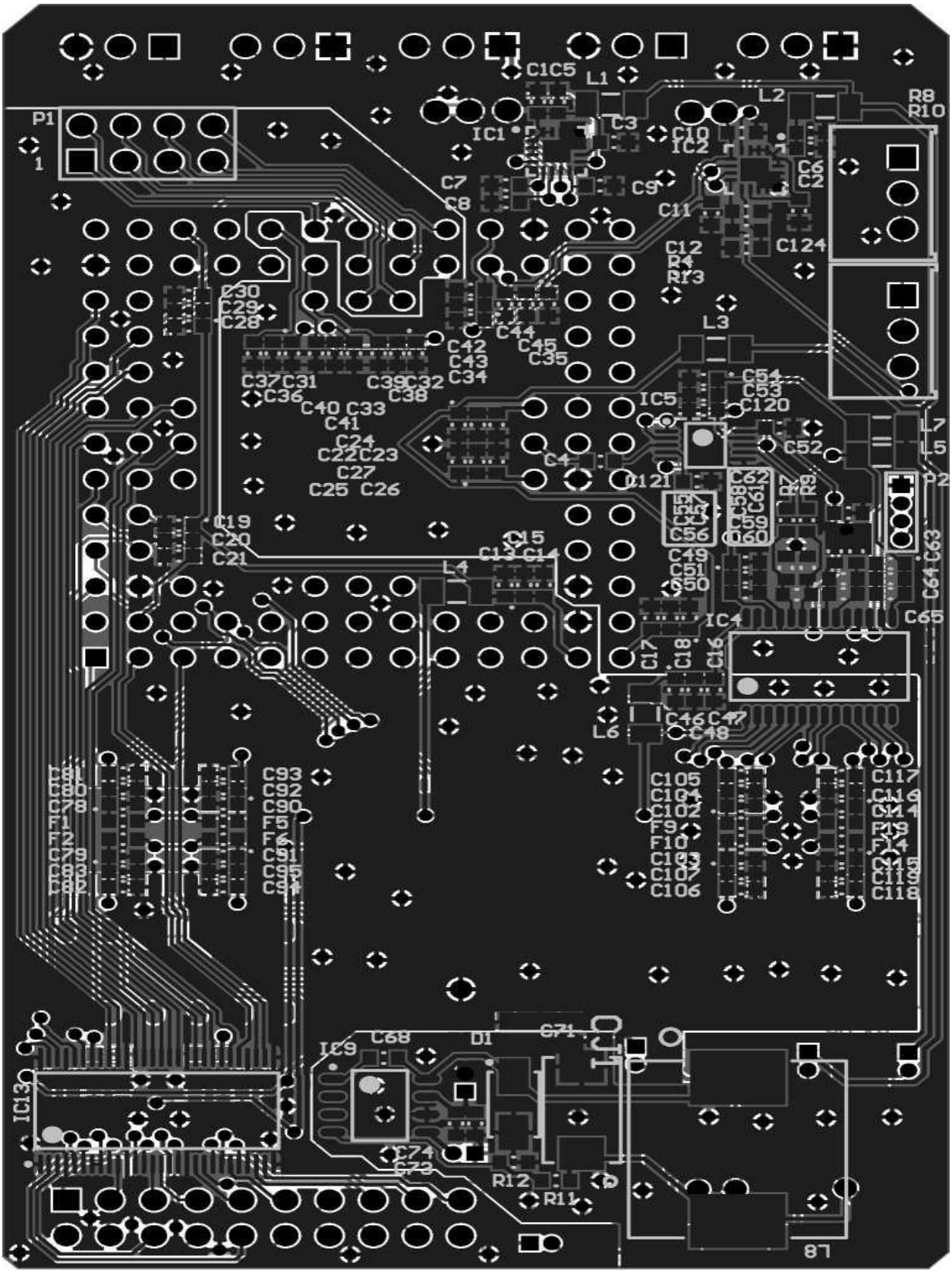


Title		
Size: A	Number	Revision
Date: 11/27/2018		Sheet of
File: C:\Users\..\Sheet4.SchDoc		Drawn By:


DC Offset Voltages
AVDD3v3
FERRITE
GND
EN1
EN2
EN3
VBIAS
VIN1/VIN2
VIN3
FB1
FB2
FB3
VOUT1
VOUT2
VOUT3
PAD
NIC
srcFlwContrlV
currentSinkControlV
cmplnN
firstAmpINP
biasAllAmps
2V0bias
Title
Size
Number
Revision
Date:
Sheet of
File:
Drawn By:


Logic Level Shifters & I/O
FPGA To Board
Board To FPGA
DGND
DVDD3v3
DVDD5v0
DVDD1v8
INPUTS
OUTPUT
Title
Size
Number
Revision
Date:
Sheet of
File:
Drawn By:

Comment	Description	Designator	Footprint	Quantity
1uF	Capacitor	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C121, C124	1608smd	14
100uF 10V	Polar Capacitor	C13, C16, C19, C22, C25, C26, C31, C32, C33, C34, C35, C46, C49, C54, C59, C63, C66, C67, C69, C70, C72, C75, C76, C77, C78, C79, C84, C85, C90, C91, C96, C97, C102, C103, C106, C109, C114, C115, C123	1608smdPolar	39
1nF	Capacitor	C14, C17, C21, C23, C26, C30, C37, C39, C40, C42, C44, C47, C58, C65, C81, C82, C87, C88, C93, C94, C99, C100, C105, C106, C111, C112, C117, C118, C120	1608smd	29
100nF	Capacitor	C15, C18, C20, C24, C27, C29, C36, C38, C41, C43, C45, C48, C51, C52, C53, C56, C60, C61, C62, C64, C80, C83, C86, C89, C92, C95, C98, C101, C104, C107, C110, C113, C116, C119, C122	1608smd	35
10pF	Capacitor	C55, C57	1608smd	2
20pF	Capacitor	C56	1608smd	1
10nF	Capacitor	C68	1608smd	1
100uF 10V	CAP TANT 100UF 10V 20% 2917	C71	CAPM3216X18N	1
4.7uF 50V	CAP CER 4.7UF 35V X5R 0603	C73, C74	1608smd	2
cj	PCB Connector	CJ1	bCan2	1
B340A	DIODE SCHOTTKY 40V 3A SMA	D1	DIOM4326X24N Molded Diode, 2-leads, Body 4.30x2.60mm, IPC Medium Density	1
FERRITE	180 Ohm @ 100MHz	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F14	1608smd	14
ADP323	IC REG LINEAR POS ADJ 1.5A FCSP	IC1, IC2	DFN50P300X300X80-8S-17L	2
PGA101	PGA101	IC3	PGA101x13	1
AD9224	IC ADC 12BIT 40MSPS 28-SSOP	IC4	SSOP127P600-8N-2	1
AD8351	IC DIFF AMP RF/IF LOWDIST 10MSOP	IC5	MSOP127P500-8N	1
ADP3338	IC REG LINEAR 5V 1A SOT223-3, IC REG LINEAR 3.3V 1A SOT223-3, IC REG LINEAR 2.5V 1A SOT223-3, IC REG LINEAR 1.8V 1A SOT223-3	IC6, IC7, IC8, IC10	SOT230P700X160-4N	4
TPS5420MDREP	IC REG BUCK ADJ 2A 8SOIC	IC9	SOIC127P600-8N	1
SN74AVC20T245	IC BUS TRANSCVER 20BIT 56TSSOP	IC11, IC13, IC14	TSSOP127P600-9N	3
NLSV8T244	TRANSLATOR 8BIT DUAL 20-TSSOP	IC12	TSSOP20-LevelShift	1
FERRITE	FERRITE BEAD 50 OHM 1206 1LN	L1, L2, L3, L4, L5, L6, L7	INDC3216X110N	7
33uH	FIXED IND 33UH 3.23A 60 MOHM SMD	L8	CAPM125125	1
Header 4X2	Header, 4-Pin, Dual row	P1	HDR2X4	1
MHDR1X4	Header, 4-Pin	P2	MHDR1X4	1
MHDR1X2	Header, 2-Pin	P3, P4, P7, P8, P9, P10	MHDR1X2	6
Header 2x10 J1	J1 Debug Board Header	P5	Header10x2	1
Header 2x20 J2	J2 Debug Board Header	P6	Edge Connector 20x2 - Inv	1
Trim 200K	potentiometer 3259W	R1, R2, R3, R5, R6	THmmer	5
10kΩ	RES SMD 10K OHM 1% 1/10W 0603	R4, R11	1608smd	2
40Ω	RES SMD 40.2 OHM 1% 1/10W 0603	R7, R9	1608smd	2
Trim 1K	potentiometer 3299W	R8	THmmer	1
Trim 5K	potentiometer 3299W	R10	THmmer	1
2.5kΩ	RES SMD 2.4K OHM 1% 1/10W 0603	R12	1608smd	1
3.24kΩ	RES SMD 3.32K	R13	1608smd	1


P1
IC1
IC2
IC4
IC5
IC9
IC13
L1
L2
L3
L4
L5
L6
L7
P2
D1
R8
R10
R4
R13
R12
R11
C1C5
C3
C10
C7
C8
C9
C11
C12
C124
C30
C29
C28
C44
C42
C43
C45
C35
C37
C31
C39
C32
C34
C36
C38
C40
C33
C41
C24
C22
C23
C27
C25
C26
C4
C121
C54
C53
C120
C52
C62
C61
C58
C59
C60
C56
C49
C51
C50
C19
C20
C21
C15
C13
C14
C17
C18
C16
C46
C47
C48
C65
C64
C63
C93
C92
C90
C91
C95
C94
C105
C104
C102
C103
C107
C106
C117
C116
C114
C115
C119
C118
C68
C71
C74
C73
87

7.2. HDL Driver Top-Level Source Code

```
module top(
// clk input
input wire clk200Mn, clk200Mp,

// user control
input wire pbN,pbE,pbW,pbC,pbS,
output wire [7:0] led,
input wire [7:0] sw,


// to on FPGA dvi Chip
output resetDviN,                   //DVI reset
output SPC,                         //SPC
output SPD,                         //SPD
output XCLK108Mp,                   //DVI clk
output XCLK108Mn,                   //DVI clk
output lv,                          //horizontal sync
output fv,                          //vertical sync
output de,                          //data enable
output [11:0] Dout,     //data

// my board interface
input [11:0] adc,
output rn,
output resetMainClk,
output resetCircuitClk,
output clkRow,
output clkAmps,
output toTimeGenClk,
output mainClk,
output clkADC,
output resetSecClk,


input a0out,
output a0In,
input b0out, b1out,
output b0In, b1In,
input c0out, c1out,
output c0In, c1In,
input d0out, d1out, d2out,
output d0In, d1In, d2In,
```

```
input e0out, e1out, e2out,
output e0In, e1In, e2In,
input f0out, f1out, f2out,
output f0In, f1In, f2In,
input g0out, g1out, g2out,
output g0In, g1In, g2In,
input h0out, h1out, h2out, h3out,

// UART
input uartRx,
output uartTx,

//temp debug
output somePinOut
// end temp debug

);


////////////////////////////////////// internal signals //////////////////////////////////////
wire reset, resetN;
wire clk135M;           // freqs from PLL
wire clk108Mp, clk108Mn;
wire XCLK216Mp, XCLK216Mn;
wire clk23p04M;
wire [11:0] timeX, timeY;
reg [7:0] toDvi;
wire [15:0] frameBuffOut;
wire [3:0] exp;
wire [7:0] CompressionDout;
wire [3:0] debug;
wire Active;
wire [7:0] uarBuffOut;
wire [23:0] colorsTest;
wire iicClk;

////////////////////////////////////// assignments //////////////////////////////////////

//board interface
assign rn = resetN;
assign reset = !resetN;

assign led[0] = debug[3];//exp[0]; //
assign led[1] = debug[2];//exp[1];     //
assign led[2] = debug[1];//exp[2];     //
```

```
assign led[3] = debug[0];//exp[2];      //
assign led[4] = h3out; //h3out;//sw[1];//exp[3];         //
assign led[5] = h2out; // b0In;  //
assign led[6] = h1out;// b0out;          //
assign led[7] = h0out;// sw[7];



// IIC
assign XCLK108Mp = XCLK216Mp;//clk108Mp;
assign XCLK108Mn = XCLK216Mn; //clk108Mn;
assign XCLK216Mn = !XCLK216Mp;


wire [15:0] tD0,tD1;

assign tD0[15:12] = exp;
assign tD0[7:0] = adc[11:4];
assign tD0[11:8] = adc[3:0];

//assign tD1[15:12] = debug;
//assign tD1[11:0] = 12'b0;

////////////////////////////////////// instances ////////////////////////////////////
uartFifoTest uartFifoTestInst(
        .reset(reset),
        .writeClk(resetSecClk),
        .readClk(Active),
        //.dataIn(sw[1]?tD0:tD1), // 4b-EXP,12b-ADC
        .dataIn({exp,adc[3:0],adc[11:4]}), // 4b-EXP,12b-ADC
        .dataOut(uarBuffOut),
        .readActive(readActive),
        .wrEn(wrEn)
);

TopUART TopUART_uut (
        .clk200Mp(clk23p04M), // 112.5M
        .reset(reset),
        .TxData(uarBuffOut),
        //.TxData({debug,4'b0000}),
        .Active(Active),
        .TxDone(TxDone),
        .TX(uartTx)
        //assign uartTx = 1'b0;
);
```

```
// temp debug
assign somePinOut = 1'b1;
// endTempShit

expBuffers expBuffersInst(
	.reset(reset),
	.debug(debug),
	.clkW(resetSecClk),
	.clkR(resetSecClk),

	// 272x64 = 17408, 1 bit
	.aStart(a0out),
	.aEnd(a0In),
	// 272x32 = 8704 2 bits
	.bStart({b1out,b0out}),
	.bEnd({b1In,b0In}),
	// 272x16 = 4352 2 bits
	.cStart({c1out,c0out}),
	.cEnd({c1In,c0In}),
	// 272x8 = 2176 3 bits
	.dStart({d2out,d1out,d0out}),
	.dEnd({d2In,d1In,d0In}),
	//272x4 =1088 3 bits
	.eStart({e2out,e1out,e0out}),
	.eEnd({e2In,e1In,e0In}),
	// 272*2 = 3 bits
	.fStart({f2out,f1out,f0out}),
	.fEnd({f2In,f1In,f0In}),
	//g = 272*1  = 3 bits
	.gStart({g2out,g1out,g0out}),
	.gEnd({g2In,g1In,g0In}),
	// h = 272*1  = 4 bits
	.hStart({h3out,h2out,h1out,h0out}),
	.hEnd(exp)
	);

imageBuffer imageBufferInst(
	.writeClk(iicClk),
	.readClk(clk108Mp),
	.reset(reset),
	.sw(sw[0]),
	.timeX(timeX),
	.timeY(timeY),
	.adcIn(adc),
```

```
        //.adcIn({colorsTest[7:0],4'b0}),
        .expIn(exp),
        .dataOut(frameBuffOut)                                  //
);


I2Ctop I2CtopInst(
        // Inputs
        .Din({toDvi,toDvi,toDvi}),
        //.Din(colorsTest),
        .XCLK108Mp(clk108Mp),
        .XCLK108Mn(clk108Mn),
        .XCLK216Mp(XCLK216Mp),
        .XCLK216Mn(XCLK216Mn),
        .reset(reset),

        // Outputs
        .Dout(Dout),
        .resetDviN(resetDviN),
        .done(),
        .lv(lv),
        .fv(fv),
        .de(de),
        .tX(timeX),
        .tY(timeY),
        .SPC(SPC),
        .SPD(SPD)

 );


sensorClocks sensorClocksInst(
        .clk(XCLK216Mp),    //clk108Mp
        .reset(reset || sw[2]),
        .resetMainClk(resetMainClk),                    // f/128
        .resetCircuitClk(resetCircuitClk),              // f/128
        .clkRow(clkRow),                                // f/128
        .clkAmps(clkAmps),                              // f/128
        .toTimeGenClk(toTimeGenClk),                    // f/128/272
        .mainClk(mainClk),                              // f/128
        .clkADC(clkADC),                                // f/128
        .resetSecClk(resetSecClk),                      // f/128
        .iicClk(iicClk),
        .debug(debug)
);
```

```
pll108M pll108Minst (// Clock in ports
        .CLK_IN1_P(clk200Mp),    // IN
        .CLK_IN1_N(clk200Mn),    // IN
        // Clock out ports
        .CLK_OUT1(clk23p04M),     // OUT 23.04M
        .CLK_OUT2(XCLK216Mp),     // OUT 216M
        // Status and control signals
        .RESET(pbN),// IN
        .LOCKED(resetN)
         );
// clock patch!
clockPatch clockPatchInst(
        .clk216Mp(XCLK216Mp),
        .reset(reset),
        .clk108Mp(clk108Mp),
        .clk108Mn(clk108Mn)
);

endmodule
```

7.3. TMO & Test System Source Code TOP LEVEL

```
`include "C:\\Univ\\Research\\Verilog\\chip_def.sv"

module CamToDvi(
//=======================================================
//  PORT declarations
//=======================================================
clkin_50,
cpu_resetn,

//////////////////////////////// UI buttons & leds ////////////////////////////////
// all deep switches
deepSw,
inputSelectBtn,
reslBtn,
ledOut,
//////////////////////////// DVI BOARD - Pin Declare ////////////////////////////
DVI_TX_CTL,
DVI_TX_DKEN,
DVI_TX_D,

DVI_TX_DDCSCL,
DVI_TX_DDCSDA,

DVI_TX_CLK,
DVI_TX_DE,
DVI_TX_VS,
DVI_TX_HS,

DVI_TX_HTPLG,
DVI_TX_ISEL,
DVI_TX_MSEN,
DVI_TX_SCL,

DVI_TX_SDA,

DVI_HSMC_SCL,
DVI_HSMC_SDA,

TX_PD_N,


//////////////////////////////// HSMC connect to AHA - Aptina Head-Board Adaptor
CK_FPGA_MCLK,
```

```
CK_IMG_IN_PIXCLK,
DEMO2_I2C_SCL,
DEMO2_I2C_SDA,
IMG_DIN,
IMG_IN_FV,
IMG_IN_LV,
SENSOR_RST,
SHUTTER

);
//=======================================================
//  PORT definition
//=======================================================
// Global clock & reset
input         clkin_50;
input         cpu_resetn;

//////////////////////////////// DVI TX BOARD Pin Declare

// ports names came from AHA demo. explanatiopns came from portB_Dvi_demo
output  [ 3: 1] DVI_TX_CTL;          // 3'h0
output         DVI_TX_DKEN;          // 1'h0
output  [23: 0] DVI_TX_D;

inout           DVI_TX_DDCSCL; // nothing
inout           DVI_TX_DDCSDA; // nothing

output         DVI_TX_CLK;   //
output         DVI_TX_DE;              //
output         DVI_TX_VS;              //
output         DVI_TX_HS;              //

output         DVI_TX_HTPLG;         // 1'h1-
output         DVI_TX_ISEL;   // 1'h0-
output         DVI_TX_MSEN;          // 1'h0
output         DVI_TX_SCL;    // 1'h1-

inout          DVI_TX_SDA;    // 1'h1-

output         DVI_HSMC_SCL;  // 1'h0
inout          DVI_HSMC_SDA;  // nothing

output         TX_PD_N;                //  1'h1-
```

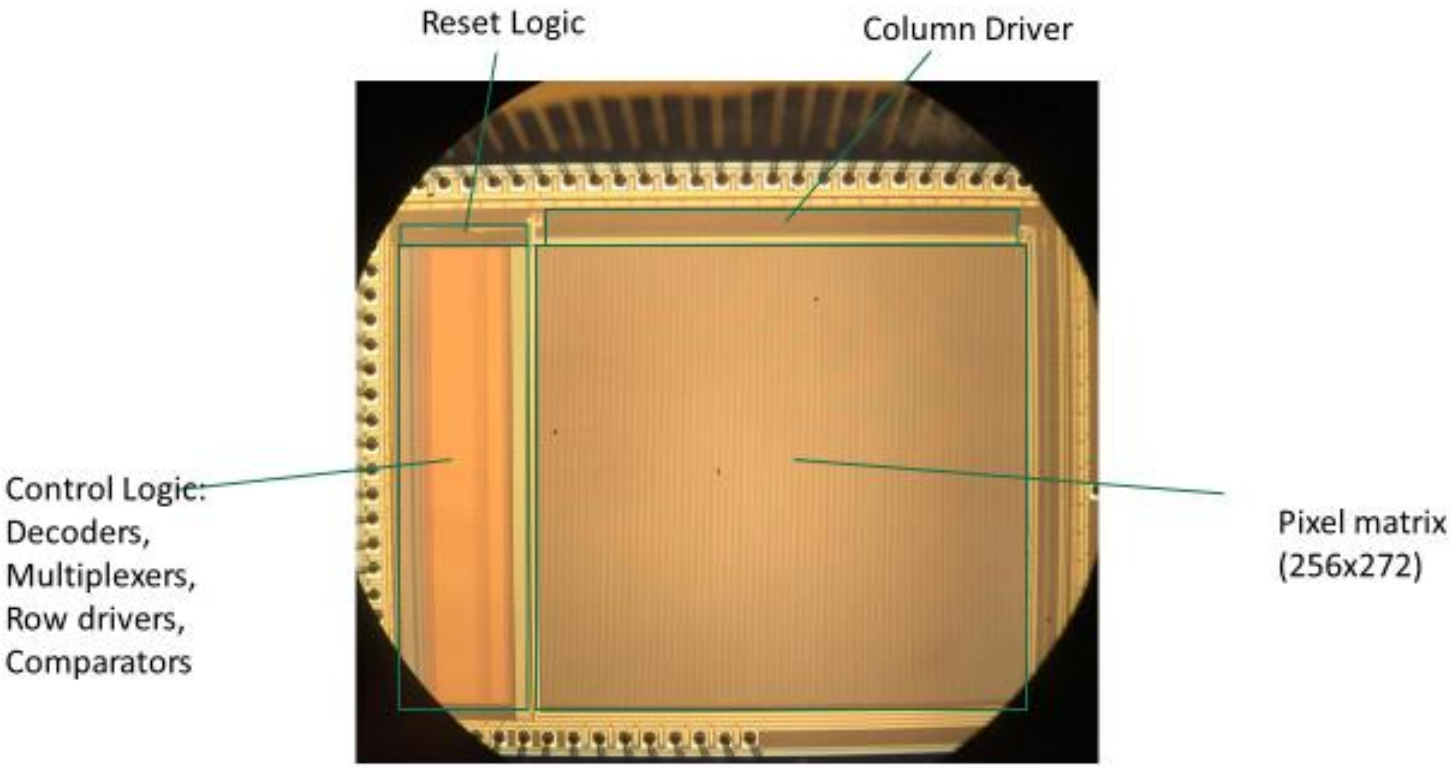
```
//////////// HSMC connect to AHA - Aptina Head-Board Adaptor //////////

output        CK_FPGA_MCLK;
input         CK_IMG_IN_PIXCLK;  //
output        DEMO2_I2C_SCL;
inout         DEMO2_I2C_SDA;
input           [11: 0] IMG_DIN;
input         IMG_IN_FV;
input         IMG_IN_LV;
output        SENSOR_RST;
output        SHUTTER;

////////////User controls buttons and leds ////////////////////////////////
input  [7:0]     deepSw;
input            inputSelectBtn;
input            reslBtn;
output [7:0]   ledOut;

////////////          DVI_TX not needed pins closing   ////////////
assign DVI_TX_ISEL     = 1'b0 ;
assign DVI_TX_SCL      = 1'b1 ;
assign DVI_TX_HTPLG    = 1'b1 ;
assign DVI_TX_SDA      = 1'b1 ;
assign TX_PD_N         = 1'b1 ;
assign DVI_TX_CTL[3:1] = 3'b0 ;
assign DVI_TX_DKEN     = 1'b0 ;
assign DVI_TX_MSEN     = 1'b0 ;
assign DVI_HSMC_SCL    = 1'b0 ;

////////////          AHA not needed pins closing  ////////////
assign SHUTTER = 1'b0;

////////////          Assignments   ////////////
assign DVI_TX_CLK = CK_IMG_IN_PIXCLK;
assign CK_FPGA_MCLK = clkin_50;

////////////          Instantiations   ////////////
decomposer decomposerINST(
.pixIn(IMG_DIN),
.clk(CK_IMG_IN_PIXCLK),
.reset(~reslBtn),
.pixOut(pixFromDecomposer),

.lvIn(IMG_IN_LV),
.fvIn(IMG_IN_FV),
```

```
.lvOut(lvFromDecomposer),
.fvOut(fvFromDecomposer)
);



BayerToRGB BayerToRGBInst(
.reset(reslBtn),
.pixIn(pixFromDecomposer),
.lvIn(lvFromDecomposer),
.fvIn(fvFromDecomposer),
.clk(CK_IMG_IN_PIXCLK),
.rOut(rFromBayer2RGB),
.gOut(gFromBayer2RGB),
.bOut(bFromBayer2RGB),
.lvOut(lvFromBayer2RGB),
.fvOut(fvFromBayer2RGB)
);


RGBtoYUV RGBtoYUVinst(
.clk(CK_IMG_IN_PIXCLK),
.reset(~reslBtn),
.lvIn(lvFromBayer2RGB),
.fvIn(fvFromBayer2RGB),
.lvOut(lvFromRGBtoYUV),
.fvOut(fvFromRGBtoYUV),
.r(rFromBayer2RGB),
.g(gFromBayer2RGB),
.b(bFromBayer2RGB),
.y(yFromRGBtoYUV),
.u(),
.v()
);

mantExpGen mantExpGenINST(
.inPix(yFromRGBtoYUV),
.clk(CK_IMG_IN_PIXCLK),
.mant(mant),
.exp(exp),
.reset(reslBtn),

.lvIn(lvFromRGBtoYUV),
.fvIn(fvFromRGBtoYUV),
.lvOut(lvFromMantExpGen),
```

```
.fvOut(fvFromMantExpGen)
);

toneMap toneMapINST(
.reset(reslBtn),
.mant({mant,4'b0}),
.exp(exp),
.clk(CK_IMG_IN_PIXCLK),
.logOut(), // working well
.maxOut(),
.minOut(),
.quotient(),
.resultOut(testLine),

.lvIn(lvFromMantExpGen),
.fvIn(fvFromMantExpGen),

.fvOut(fvFromToneMap),
.lvOut(lvFromToneMap),
);

wire [23:0]testLine;
wire [23:0]testLine1;
wire [9:0] resultOut;
wire fvFromToneMap,lvFromToneMap;

wire [19:0] pixFromDecomposer;
wire lvFromDecomposer,fvFromDecomposer;

wire [19:0] rFromBayer2RGB,gFromBayer2RGB,bFromBayer2RGB;
wire lvFromBayer2RGB,fvFromBayer2RGB;

wire [19:0] yFromRGBtoYUV,uFromRGBtoYUV,vFromRGBtoYUV;
wire lvFromRGBtoYUV,fvFromRGBtoYUV;

wire [11:0] mant;
wire [6:0] exp;
wire fvFromMantExpGen,lvFromMantExpGen;




camDriverV2 camDriverV2INST(
.reset(~reslBtn),
```

```
.clk(clkin_50),
.scl(DEMO2_I2C_SCL),
.sda(DEMO2_I2C_SDA),
.camHardReset(SENSOR_RST)
);
endmodule
```

7.4. Mantissa – Exponent Sensor Die



References

[1] Orly Yadid-Pecht, "OPTICAL IMAGER USING AMETHOD FOR ADAPTIVE REAL-TIME EXPANDING OF THE DYNAMIC RANGE," US 6,831,689 B2, 1998.

[2] O. Yadid-Pecht and A. Belenky, "Autoscaling CMOS APS with customized increase of dynamic range," *2001 IEEE Int. Solid-State Circuits Conf. Dig. Tech. Pap.*, vol. 32, no. 2, pp. 100–101, 2001.

[3] U. Shahnovich, A. Hore`, and O. Yadid-pecht, "Hardware Implementation of a Real-Time Tone Mapping Algorithm Based on a Mantissa-Exponent Representation," no. 5, 2016.

[4] T. G. Etoh, A. Q. Nguyen, Y. Kamakura, K. Shimonomura, T. Y. Le, and N. Mori, "The theoretical highest frame rate of silicon image sensors," *Sensors (Switzerland)*, vol. 17, no. 3, pp. 8–13, 2017.

[5] OmniVision, "One of the World's Smallest Commercial Image Sensors with Industry-Leading Resolution and Image Quality for Medical Applications," 2017. [Online]. Available: https://www.ovt.com/sensors/OV6948. [Accessed: 15-Feb-2018].

[6] M. E. Becker, "Display Metrology & Systems." [Online]. Available: https://display-messtechnik.de/index.php?id=produkte&L=1. [Accessed: 01-Jan-2019].

[7] M. F. Tompsett, "Charge Transfer Imaging Devices," US4085456A, 1978.

[8] G. E. Smith and W. S. Boyle, "Charged Couple Semiconductor Devices," *Bell Syst. Tech. J.*, vol. 49, no. 4, pp. 587–593, 1970.

[9] S. N. Kazuya Matsumoto, Tsutomu Nakamura, Atsushi Yusa, "Related content A New MOS Phototransistor Operating in a Non- Destructive Readout Mode," *Jpn. J. Appl. Phys.*, vol. Volume 24, no. No 5, pp. 323–325, 1985.

[10] E. R. Fossum, "Active pixel sensors: are CCDs dinosaurs?," *Charg. Devices Solid State*

Opt. Sensors III, vol. 1900, pp. 2–14, 1993.

[11] R. Drivers, "CCD vs . CMOS : Choosing an imager means considering not only the chip , but," no. January, 2001.

[12] M. Robbins, P. Jorden, D. Jordan, J. Pratlong, and P. Jerram, "e2v CCD and CMOS sensors and systems designed for astronomical applications," *High Energy, Opt. Infrared Detect. Astron. VII*, vol. 9915, no. August 2016, p. 991504, 2016.

[13] A. Spivak, A. Belenky, A. Fish, and O. Yadid-Pecht, "Wide-Dynamic-Range CMOS Image Sensors—Comparative Performance Analysis," *IEEE Trans. Electron Devices*, vol. 56, no. 11, pp. 2446–2461, 2009.

[14] T. Willassen, J. Solhusvik, R. Johansson, S. Yaghmai, H. Rhodes, S. Manabe, Z. Lin, D. Yang, O. Cellek, S. Ma, and B. Zhang, "A 1280x1080 4 . 2μm Split-diode Pixel HDR Sensor in 110nm BSI CMOS Process," in *2015 International Image Sensor Workshop*, 2015, pp. 4–7.

[15] I. Takayanagi, N. Yoshimura, K. Mori, S. Matsuo, S. Tanaka, H. Abe, N. Yasuda, K. Ishikawa, S. Okura, S. Ohsawa, and T. Otaka, "An over 90 DB intra-scene single-exposure dynamic range cmos image sensor using a 3.0 μm triple-gain pixel fabricated in a standard BSI process," *Sensors (Switzerland)*, vol. 18, no. 1, pp. 1–11, 2018.

[16] S. Decker, R. D. McGrath, K. Brehmer, and C. G. Sodini, "A 256 × 256 CMOS imaging array with wide dynamic range pixels and column-parallel digital output," *IEEE J. Solid-State Circuits*, vol. 33, no. 12, pp. 2081–2090, 1998.

[17] M. Schanz, C. Nitta, A. Bußmann, B. J. Hosticka, and R. K. Wertheimer, "High-dynamic-range CMOS image sensor for automotive applications," *IEEE J. Solid-State Circuits*, vol. 35, no. 7, pp. 932–938, 2000.

[18] A. Belenky, A. Fish, A. Spivak, and O. Yadid-pecht, “Global Shutter CMOS Image Sensor With Wide Dynamic Range,” *Image (Rochester, N.Y.)*, vol. 54, no. 12, pp. 1032–1036, 2007.

[19] S. Cui, Z. Li, C. Wang, X. Yang, X. Wang, and Y. Chang, “A Charge Compensation Phototransistor for High-Dynamic-Range CMOS Image Sensors,” *IEEE Trans. Electron Devices*, vol. 65, no. 7, pp. 2932–2938, 2018.

[20] X. Liu and A. El Gamal, “Synthesis of high dynamic range motion blur free image from multiple captures,” *IEEE Trans. Circuits Syst. I Fundam. Theory Appl.*, vol. 50, no. 4, pp. 530–539, 2003.

[21] M. Mase, S. Kawahito, M. Sasaki, Y. Wakamori, and M. Furuta, “A wide dynamic range CMOS image sensor with multiple exposure-time signal outputs and 12-bit column-parallel cyclic A/D converters,” *IEEE J. Solid-State Circuits*, vol. 40, no. 12, pp. 2787–2795, 2005.

[22] J. J. Zarnowski, M. A. Pace, and M. Joyner, “1.5-FET-per-pixel standard CMOS active column sensor,” *Electron. Imaging’99*, vol. 3649, no. April 1999, pp. 186–196, 1999.

[23] J. Solhusvik, S. Yaghmai, A. Kimmels, A. Storm, J. Olsson, A. Rosnes, T. Willassen, P. Pahr, S. Eikedal, R. Bhamra, S. Velichko, D. Pates, S. Datar, S. Smith, L. Jiang, D. Wing, and A. Chilumula, “A 1280x960 3.75um pixel CMOS imager with Triple Exposure HDR,” *Proc. Int. Image Sens. Work.*, 2009.

[24] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dinamic Range Imaging*. 2006.

[25] P. Lapray, B. Heyrman, M. Rosse, and D. Ginhac, “A 1.3 megapixel FPGA-based smart camera for high dynamic range real time video,” in *2013 Seventh International Conference on Distributed Smart Cameras (ICDSC)*, 2013, pp. 1–6.

[26] V. Popovic, K. Seyid, and E. Pignat, “Multi-camera platform for panoramic real-time HDR video construction and rendering,” *J. Real-Time Image Process.*, 2014.

[27] R. Urena, P. Martinez-Canada, J. M. Gomez-Lopez, C. Morillas, and F. Pelayo, “Real-time tone mapping on GPU and FPGA,” *EURASIP J. Image Video Process.*, vol. 2012, no. 1, p. 1, 2012.

[28] A. K. Bachoo, “Real-time exposure fusion on a mobile computer,” *20th Annu. Symp. Pattern Recognit. Assoc. South Africa PRASA*, pp. 111–115, 2009.

[29] A. O. Akyüz, “High dynamic range imaging pipeline on the GPU,” *J. Real-Time Image Process.*, no. 2001, pp. 273–287, 2012.

[30] P. Debevec, E. Reinhard, G. Ward, and S. Pattanaik, “High dynamic range imaging,” *Vistas Astron.*, vol. 180, no. 2, pp. 291–295, 2004.

[31] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, “Adaptive Logarithmic Mapping for Displaying High Contrast Scenes,” *Comput. Graph. Forum*, vol. 22, no. 3, pp. 419–426, 2003.

[32] E. Reinhard, M. Stark, and P. Shirley, “Photographic Tone Reproduction for Digital Images,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 267–276, 2002.

[33] M. A. Ali and S. Mann, “Comparametric image compositing: Computationally efficient high dynamic range imaging,” *Proc. Int. Conf. Acoust., Speech, Signal Process.*, pp. 913–916, 2012.

[34] R. Fattal, D. Lischinski, and M. Werman, “Gradient domain high dynamic range compression,” *ACM Trans. Graph.*, vol. 21, pp. 249–256, 2002.

[35] J. Duan, M. Bressan, C. Dance, and G. Qiu, “Tone-mapping high dynamic range images by novel histogram adjustment,” *Pattern Recognit.*, vol. 43, no. 5, pp. 1847–1862, 2010.

[36] Q. Tian, J. Duan, M. Chen, T. Peng, A. Science, and H. Kong, “Segmentation Based Tone-Mapping for High Dynamic,” *Adv. Concepts Intell. Vis. Syst.*, pp. 360–371, 2011.

[37] A. Boschetti, N. Adami, R. Leonardi, and M. Okuda, “High Dynamic Range image tone mapping based on local Histogram Equalization,” in *2010 IEEE International Conference on Multimedia and Expo*, 2010, pp. 1130–1135.

[38] X. Yang, K. Xu, Y. Song, Q. Zhang, X. Wei, and R. W. H. Lau, “Image Correction via Deep Reciprocating HDR Transformation,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1798–1807.

[39] J. Cai, S. Gu, and L. Zhang, “Learning a Deep Single Image Contrast Enhancer from Multi-Exposure Images,” *IEEE Trans. Image Process.*, vol. 27, no. 4, pp. 2049–2062, Apr. 2018.

[40] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, “Deep bilateral learning for real-time image enhancement,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.

[41] T. Dobashi, T. Murofushi, M. Iwahashi, and H. Kiya, “A fixed-point global tone mapping operation for HDR images in the RGBE format,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E97A, no. 11, pp. 2147–2153, 2014.

[42] D. Hertel, A. Betts, R. Hicks, and M. Ten Brinke, “An adaptive multiple-reset CMOS wide dynamic range imager for automotive vision applications,” *IEEE Intell. Veh. Symp. Proc.*, pp. 614–619, 2008.

[43] Y. Dattner and O. Yadid-Pecht, “High and low light CMOS imager employing wide dynamic range expansion and low noise readout,” *Sensors Journal, IEEE*, vol. 12, no. 6, pp. 2172–2179, 2012.

[44] A. Spivak and O. Yadid-Pecht, “Design of A 128 ?? 128 CMOS APS with extended noise suppression for high and low light imaging applications,” *Proc. - IEEE Int. Symp. Circuits Syst.*, pp. 45–48, 2014.

[45] A. Spivak, A. Belenky, and O. Yadid-Pecht, “Very Sensitive Low-Noise Active-Reset CMOS Image Sensor with In-Pixel ADC,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 63, no. 10, pp. 939–943, 2016.

[46] O. Yadid-Pecht and A. Belenky, “In-pixel autoexposure CMOS APS,” *IEEE J. Solid-State Circuits*, vol. 38, no. 8, pp. 1425–1428, 2003.

[47] C. Ofili, S. Glozman, and O. Yadid-Pecht, “Hardware Implementation of an Automatic Rendering Tone Mapping Algorithm for a Wide Dynamic Range Display,” *J. Low Power Electron. Appl.*, vol. 3, no. 4, pp. 337–367, 2013.

[48] P. Ambalathankandy, A. Horé, and O. Yadid-Pecht, “An FPGA implementation of a tone mapping algorithm with a halo-reducing filter,” *J. Real-Time Image Process.*, pp. 1–17, 2016.

[49] F. Durand and J. Dorsey, “Fast bilateral filtering for the display of high-dynamic-range images,” in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’02*, 2002, p. 257.

[50] R. Mantiuk, K. Myszkowski, and H. Seidel, “Lossy compression of high dynamic range images and video,” *Proc. SPIE 6057, Hum. Vis. Electron. Imaging XI*, vol. 6057, p. 60570V–60570V–10, 2006.

[51] A. Horé and D. Ziou, “Image quality metrics: PSNR vs. SSIM,” *Proc. - Int. Conf. Pattern Recognit.*, pp. 2366–2369, 2010.

[52] H. Yeganeh and Z. Wang, “Objective Quality Assessment of Tone-Mapped Images,”

Image Process. IEEE Trans., vol. 22, no. 2, pp. 657–667, Feb. 2013.

[53] C.-T. Chiu, T.-H. Wang, W.-M. Ke, C.-Y. Chuang, J.-S. Huang, W.-S. Wong, R.-S. Tsay, and C.-J. Wu, "Real-Time Tone-Mapping Processor with Integrated Photographic and Gradient Compression using 0.13 μm Technology on an Arm Soc Platform," *J. Signal Process. Syst.*, vol. 9628, pp. 1–15, 2010.

[54] D. J. Natale, M. S. Baran, and R. L. Tutwiler, "High dynamic range (HDR) video processing for the exploitation of high bit-depth sensors in human-monitored surveillance," in *2014 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 2014, pp. 1–6.

[55] P. J. Lapray, B. Heyrman, M. Rosse, and D. Ginhac, "HDR-ARtiSt: High dynamic range advanced real-time imaging system," *ISCAS 2012 - 2012 IEEE Int. Symp. Circuits Syst.*, pp. 1428–1431, 2012.

[56] J. W. Glotzbach, "A Color Filter Array Interpolation Method Based on Sampling Theory," Georgia Institute of Technology, 2004.