A DIGITALLY PROGRAMMABLE APPLICATION SPECIFIC INTEGRATED CIRCUIT FOR DRIVE AND DATA ACQUISITION OF IMAGING SENSORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

NUSRET BAYHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2014

Approval of the thesis:

A DIGITALLY PROGRAMMABLE APPLICATION SPECIFIC INTEGRATED CIRCUIT FOR DRIVE AND DATA ACQUISITION OF IMAGING SENSORS

submitted by **NUSRET BAYHAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Tayfun Akın _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

Dr. Selim Eminoğlu _____
Co-supervisor, **Mikro-Tasarım San. ve Tic. Ltd. Şti.**

Examining Committee Members:

Assoc. Prof. Dr. Haluk Külah
Electrical and Electronics Eng. Dept., METU _____

Prof. Dr. Tayfun Akın
Electrical and Electronics Eng. Dept., METU _____

Dr. Selim Eminoğlu
Mikro-Tasarım San. ve Tic. Ltd. Şti. _____

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Eng. Dept., METU _____

Dr. Enis Ungan
Karel Elektronik San. Tic. A. Ş. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: NUSRET BAYHAN

Signature :

ABSTRACT

A DIGITALLY PROGRAMMABLE APPLICATION SPECIFIC INTEGRATED CIRCUIT FOR DRIVE AND DATA ACQUISITION OF IMAGING SENSORS

Bayhan, Nusret

M.S., Department of Electrical and Electronics Engineering

Supervisor : Prof. Dr. Tayfun Akın

Co-Supervisor : Dr. Selim Eminoğlu

February 2014, 77 pages

This thesis explains the implementation of a digital programmable Application Specific Integrated Circuit (ASIC) designed for imaging applications. The primary function of this ASIC is to drive imaging sensors and to do basic processing on the digital video data coming from the sensors.

The ASIC is designed to handle the communication between the imaging sensor and the system. Using command based high-level instructions, this two-way communication is simplified. The ASIC can also be used to store and update the sensor memory content using this communication interface.

The ASIC has a built in data path that can process the digital sensor data in real time while the sensor is being operated. This data path is capable of performing both arithmetic and encoding operations on the sensor data. The arithmetic operations are handled by an integrated arithmetic unit placed on the data path. This unit can be used to correct and enhance the sensor data at the pixel level using a reduced set of special commands and instructions. The ASIC also has a built in 8bit/10bit data encoder at the end of its data path, which is integrated to support high speed serial data interfaces by providing a DC-balanced digital output data.

The ASIC has an integrated programmable timing generator designed to generate the necessary timing signals for the imaging sensors and their peripherals. This module can be programmed to generate periodic timing signals at the period of line or frame times of the imaging sensor.

The logic implemented in the ASIC is simulated, synthesized, placed and routed in sequence using automated digital design tools using Hardware Description Language (HDL) design capture. Since IC implementation is typically expensive, the designed logic is first implemented and verified at FPGA level to assure its functionality. The results of both implementations show that IC implementation is advantageous in terms of power and area for a given speed at the expense of its cost.

Keywords: ASIC, Application Specific Integrated Circuit, Image Sensors, Data Acquisition, Hardware Description Language

ÖZ

GÖRÜNTÜLEME SENSÖRLERİNİ SÜRME VE SENSÖRLERDEN VERİ ALMA İÇİN UYGULAMAYA ÖZEL SAYISAL PROGRAMLANABİLİR TÜMDEVRE

Bayhan, Nusret

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. Tayfun Akın

Ortak Tez Yöneticisi : Dr. Selim Eminoğlu

Şubat 2014 , 77 sayfa

Bu tezde, görüntüleme uygulamalarında kullanılmak üzere tasarlanan, sayısal ve programlanabilir bir uygulamaya özel tümdevre anlatılmaktadır. Bu tümdevrenin başlıca kullanım alanı görüntüleme sensörlerini sürmek ve sensör video çıkışları üzerinde basit işlemler yapabilmektir.

Tezde tasarlanan tümdevre, görüntüleme sensörü ve sistem arasındaki haberleşmeyi kontrol edebilmektedir. Tasarımda gelişmiş komutlar kullanılarak, çift yönlü olan bu haberleşme basitleştirilmiştir. Bunun yanında, görüntüleme sensörünün hafıza içeriği de tasarlanan tümdevre üzerinde tutulabilir ve bu hafıza kolay bir şekilde programlanabilir.

Tümdevre, görüntüleme sensörü çıkışlarını gerçek zamanlı işleme kabililiyetine sahip olan bir veri yolu da içermektedir. Bu veri yolu, sensör verileri üzerinde aritmetik ve kodlama işlemleri uygulayabilir. Aritmetik işlemleri, veri yolunda yer alan aritmetik birim uygulamaktadır. Bu birim, tasarlanan özel bir komut seti yardımıyla, sensörden gelen veriler üzerinde piksel aşamasında düzeltme ve iyileştirme uygulayabilir. Aritmetik birimin çıkışına ise yüksek hızlı seri arayüzleri destekleyebilmek amacıyla bir 8bit/10bit kodlayıcı yerleştirilmiştir.

Bu birimlere ek olarak tümdevrede, sensörler ve çevresel birimler için kullanılmak üzere tasarlanmış olan bir zamanlama üreteci bulunur. Bu ünite, gerekli olması halinde kullanılabilecek periyodik zamanlama sinyallerini üretebilir.

Tezde tasarlanan tümdevre, Verilog Donanım Tanımlama Dili (DTD) kullanılarak, otomatik sayısal entegre devre tasarım yazılımları ile simülasyonlarla doğrulanmış, sentezlenmiş, yerleşimi ve bağlantıları yapılmıştır. Entegre devre üretimi pahalı olduğu için, tasarlanan tümdevre öncelikle FPGA üzerinde fonksiyonel açıdan doğrulanmıştır. Son olarak ise entegre devre ve FPGA üzerinde yapılan tasarımlar karşılaştırılmıştır. Sonuçta, pahalı olmasına rağmen, belirlenen bir çalışma hızında entegre tasarımının güç tüketimi ve alan açısından daha avantajlı olduğu görülmüştür.

Anahtar Kelimeler: ASIC, Uygulamaya Özel Tümdevre, Görüntüleme Sensörleri, Veri Alma, Donanım Tanımlama Dili

To my family

ACKNOWLEDGMENTS

I would like to express my gratitude towards my advisor Prof. Dr. Tayfun Akın for his supervision and guidance throughout this study. I would like to thank Dr. Selim Eminoğlu for his endless support and trust through my graduate life.

I would like to thank Siner Gökhan Yılmaz for his technical support in software and Cem Yalçın for being such a patient model in front of the camera. I also would like to add my other colleagues at Mikro-Tasarım, Osman Samet İncedere, Mehmet Ali Gülden, Suhip Tuncer Soyer, Murat Işıkhan, Serhat Koçak, Mithat Cem Boreyda Üstündağ, Özge Turan, Burak Baycan and Mehmet Akdemir, to this list. In addition, I would like to thank Mehmet Ufuk Büyükşahin for his helps and supports throughout this study. I would not be able to finish this thesis without their invaluable friendship and support.

I would like to express my gratitude towards to TÜBİTAK for their support of my M.Sc. study with their scholarship.

Last but not least, I would like to give my special thanks to my mother, my father and my sister for their endless trust, encouragement and patience in every moment of my life. Without the support of all these people that I mentioned, I would not be able to achieve this success.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Imaging technology is one of the most vivid fields in electronics. With the help of imaging equipments, people take pictures, record videos or even see what their eyes cannot. While performing those, the imaging sensor along with its camera electronics does the job. All parts together that have a role in this sequence called as an imaging system.

1.1 Imaging Systems

As stated above, an imaging system is the collection of all equipments that are used in image acquisition process. Commercially available digital cameras can be given as examples to such systems. Like these digital cameras, there are many electronic devices that can capture images or videos. Their structures, however, more or the less the same in terms of used equipments. These equipments can be electrical, optical or mechanical parts. Figure 1.1 shows an example imaging system block diagram.

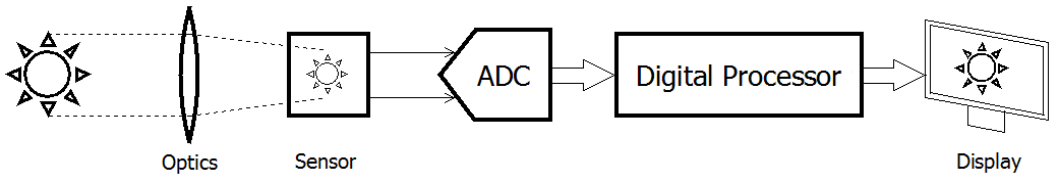

Figure 1.1: An example imaging system block diagram.

The electrical parts of the imaging systems mainly consist of an image sensor with its readout circuit and external electronics that reads the data from that sensor.

1.1.1 Sensor

Image sensors are the parts that convert incoming optical radiation into electrical signals. This can be explained as the part where the imaging process begins. There are mainly two categories of sensors according to their structures, which are monolithic sensors and hybrid sensors.

Monolithic sensors are single crystal devices with sensor part and readout circuit mostly reside on a single chip. The most common examples to these sensors are silicon based and used for visible wavelengths (0.4μm-0.7μm). CCD and CMOS sensors are two well-known examples for monolithic sensors.

On the other hand, hybrid sensors are formed by combining separately produced sensor and readout circuit parts together. These sensors are produced from other materials than silicon, to detect the radiation that silicon cannot. Infrared radiation can be given as an example. Although the wavelength interval 0.7μm to 1μm, which is classified as near infrared (NIR) region, can be visible with silicon; it is not possible to detect larger wavelengths than 1μm with it. Short-wave infrared (SWIR) wavelengths (1μm-3μm), mid-wave infrared (MWIR) wavelengths (3μm-5μm) and long-wave infrared (LWIR) wavelengths (8μm-12μm) are examples for such intervals.

The electrical outputs of the sensors are usually very small signals. Also pixel counts of image sensors are very high for reading them one-by-one. Even a VGA sensor, which has a resolution of 640x480, contains more than 300000 pixels. In order to handle those kind of difficulties, readout circuits are used. These circuits take the sensor output, amplify them and forward to the analog-to-digital converters.

1.1.2 External Electronics

One common point for all of these sensors is they all need some external electronics for proper operation. The requirements to the external electronics can be different between different types of sensors, however. External electronics here means all other electronic parts that are required to obtain meaningful data from the sensor. ADCs, controllers and memories are common parts that can be classified in this category.

1.1.2.1 Analog-to-Digital Converter

As stated before, sensor converts incoming radiation to analog electrical signals and readout circuit shapes those analog signals. However, in order to obtain images from those electrical signals, they should be digitized as pixel values. To achieve this, analog sensor outputs are converted to digital values via Analog-to-Digital Converters (ADCs). Different types of ADCs can be found in the market and in the literature. Resolution in terms of bits, speed, voltage levels and output types are some of the properties that affect ADC selection. According to the needs of imaging application, a proper ADC set should be selected in order to obtain correct image from the sensor.

1.1.2.2 Controller

Another important part of an imaging system is the controller. The necessary control signals and configurations for other electronics are generated and held by this unit. Since it is the closest digital part to the sensor and ADCs, the programming and driving tasks of these parts are performed by digital controller. Besides driving sensors and ADCs, the data transfer between a host and the sensor is also handled by this part. The controller may be designed as a simple circuit or it may be very complex depending on the application.

1.1.2.3 Memory

In an imaging system, an external memory can be necessary for many operations. Storing images, saving configurations or stacking reference frames for processing purposes can be given as examples. Today, there are many memory choices available on the market in terms of speed, capacity or for many other specifications.

Besides these common parts, some electronic devices that are designed for specific sensors are also available in the market. For instance, Texas Instruments' CCD signal processor chips have the capability of data processing as well as analog to digital converters[5]. These devices are used along with CCD cameras. Similarly, the processors with more complex operation capabilities are available for CMOS sensors.

On the other hand, for the hybrid sensors, the priority is to digitize the data in an efficient way rather than performing complex processing operations on it. This make the external electronics more important for such sensors.

In this section, the main parts of an imaging system are explained. Since the thesis is focused on the development of a digital integrated circuit, the common digital circuit design methodologies will be explained in the following section.

1.2 Digital Circuit Design

There are two basic design methodologies for digital design. These are using programmable logic devices and designing application specific integrated circuits.

1.2.1 Programmable Logic Devices (PLDs)

In imaging systems, like most of the electronic systems, analog design is usually the part where most of the effort is put. Therefore, analog part of the designs are tend to be more stable and closed to changes after sign-off. The digital part, on the other hand, need to be designed in relatively short time and it has more tendency to change or upgrade according to the system needs. This leads to search of easy-to-configure and reusable devices, which are called as Programmable Logic Devices (PLDs) in general.

1.2.1.1 Simple Programmable Logic Device (SPLD)

Simple Programmable Logic Devices (SPLDs) can be expressed as gate arrays that have low logic capacities. These are developed as first PLD examples. Their internal structures usually consist of simple AND-OR gates. There are mainly two types of SPLDs in the literature.

Programmable Logic Array (PLA) Programmable Logic Arrays (PLAs) are one of the first devices that can be counted as PLDs and developed in late 1970s[1]. They

include two gate array planes, one for AND gates and one for OR gates. The inputs of both gate arrays are programmable with a fuse structure. Figure 1.2 shows a simple PLA structure.

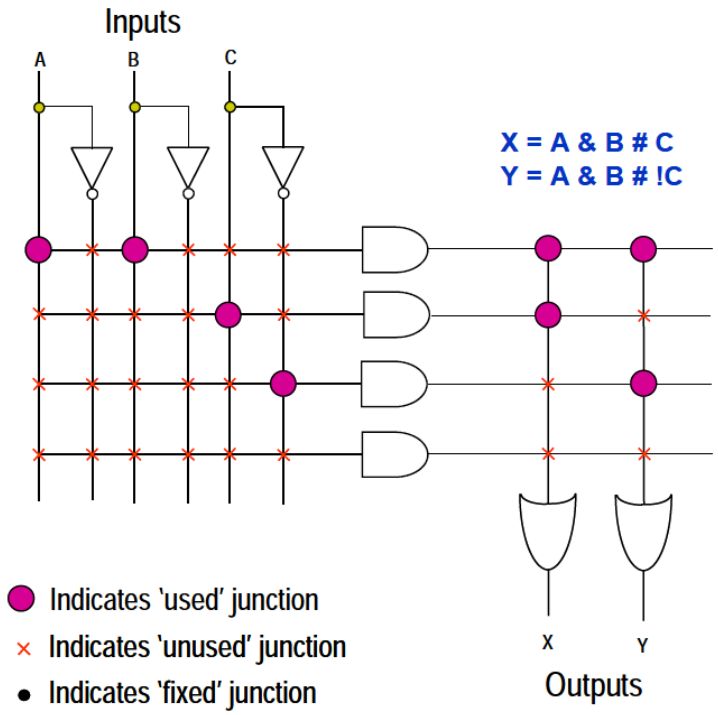

Figure 1.2: A simple PLA Structure[1]

Although obtaining all combinations of AND/OR gates is possible with this structure, high fuse count makes the device harder to program.

Programmable Array Logic (PAL) Since any combinational logic can be expressed as sum of products method[6], the sum (OR) parts can be fixed and all the programming options can be leaved to the products (AND) parts. Programmable Array Logic (PAL) device implements this idea and fixes the OR plane in a PLA. Figure 1.3 shows a simple PAL structure.

This structure lessens the programming complexity problem. However, the programmable combinations of AND/OR gates are less than a PLA device.

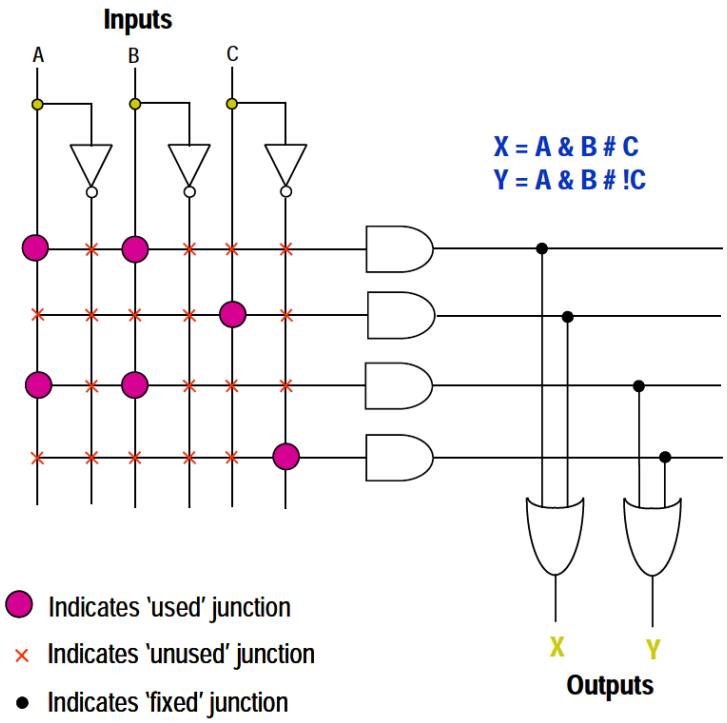

Figure 1.3: A simple PAL Structure[1]

1.2.1.2 Complex Programmable Logic Device (CPLD)

Complex Programmable Logic Devices (CPLDs) are developed after SPLDs. It provides multiple SPLD inside it. An interconnect array connects these SPLDs together to increase the logic interpretation capacity. Figure 1.4 shows a simple CPLD structure.

The CPLDs provides increased gate count with respect to SPLD structures and implementation of more sophisticated logic functions is possible.

1.2.1.3 Field Programmable Gate Array (FPGA)

Increased success of PLDs in the market leads to development of Field Programmable Gate Arrays (FPGAs) by Xilinx, in 1985[1]. The idea behind FPGAs is using configurable logic blocks (CLBs) with programmable interconnects between them, instead of simple logic gates. These CLBs are usually capable of performing more complex logic tasks than previous gate arrays. Figure 1.5 shows block diagram of an FPGA

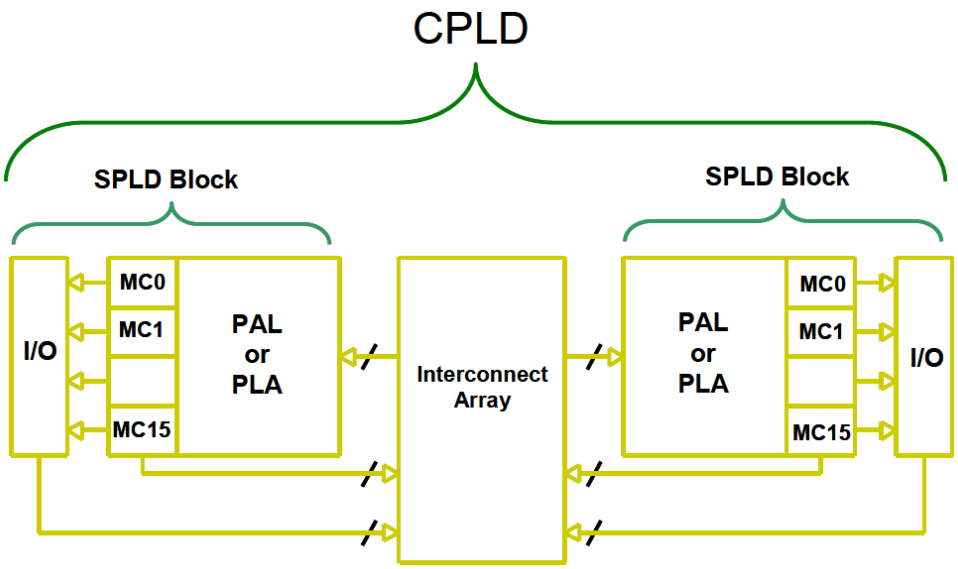

Figure 1.4: A simple CPLD Structure[1]

structure.

In today's FPGAs, a CLB can host up to 6 input look-up tables (LUTs), flip-flops, multiplexers and memory structures[7].

The configuration of these logic blocks is the basic design process of FPGAs. This configuration is performed using design tools that are developed by FPGA vendors. The high-level schematic or HDL designs are converted to the FPGA configuration files using these design tools. Figure 1.6 shows typical FPGA design flow.

1.2.2 Application Specific Integrated Circuit (ASIC)

The other way of designing digital integrated circuits, rather than using PLDs, is the design of an Application Specific Integrated Circuit (ASIC). As the name suggests, these are electronic devices that are designed and produced for specific applications. Therefore, ASICs are not general programmable devices like PLDs. Instead, they serve for specific purposes. The most common ASIC design methods can be summarized as full-custom and semi-custom based designs.

In the full-custom ASIC design method, the whole integrated circuit is designed man-

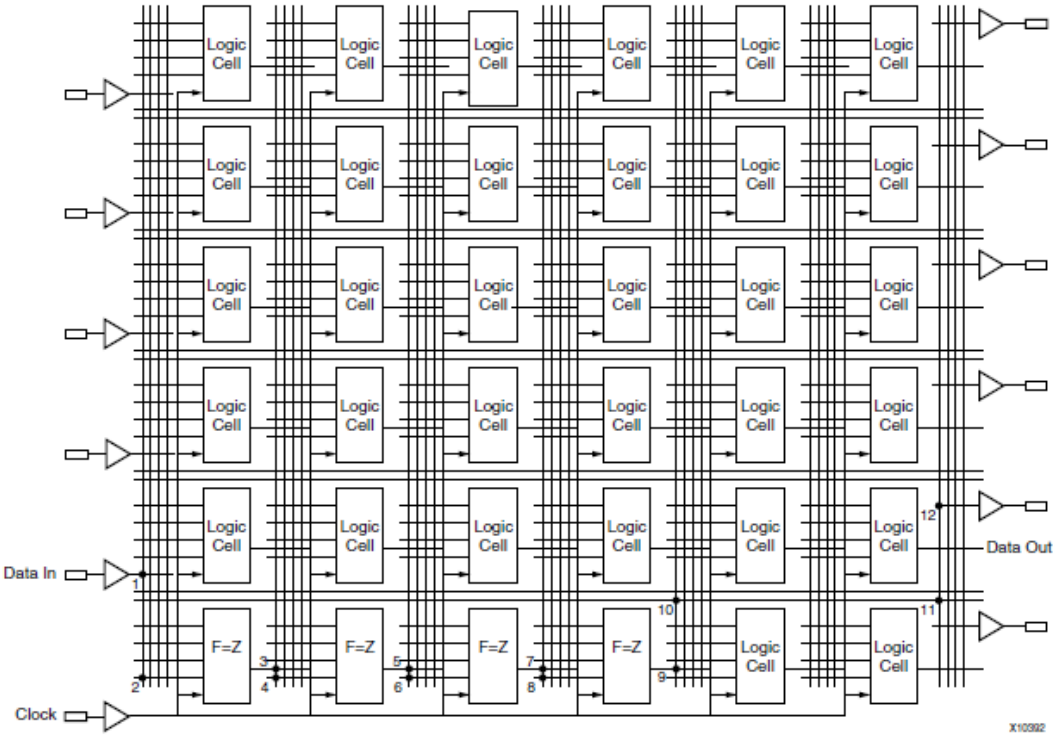

Figure 1.5: Block diagram of an FPGA structure[1]

ually, from top level to the smallest logic cell. This design method is important for crucial circuits since each step of the design is under the control of the designer. Also it is possible to obtain better power and performance results than other ASIC design methods [8] [9]. However, it should be noted that the time requirement of full-custom design is higher than the other design methods.

In the semi-custom design method, on the other hand, designers mostly use automated design tools to produce desired circuit. This brings the standard cell design term. At this point, one thing should be added to the classification of semi-custom ASIC design. In the early years of the gate array technologies, programmable arrays and FPGAs were also classified under the semi-custom ASIC design[10]. Since the gate array technologies, including FPGAs, are explained formerly in this chapter; the explanation will continue with the standard cell design method.

The standard cell design is one of the most popular methods for ASIC design. In this method, the circuit is build from a standard cell library, using automated design tools.

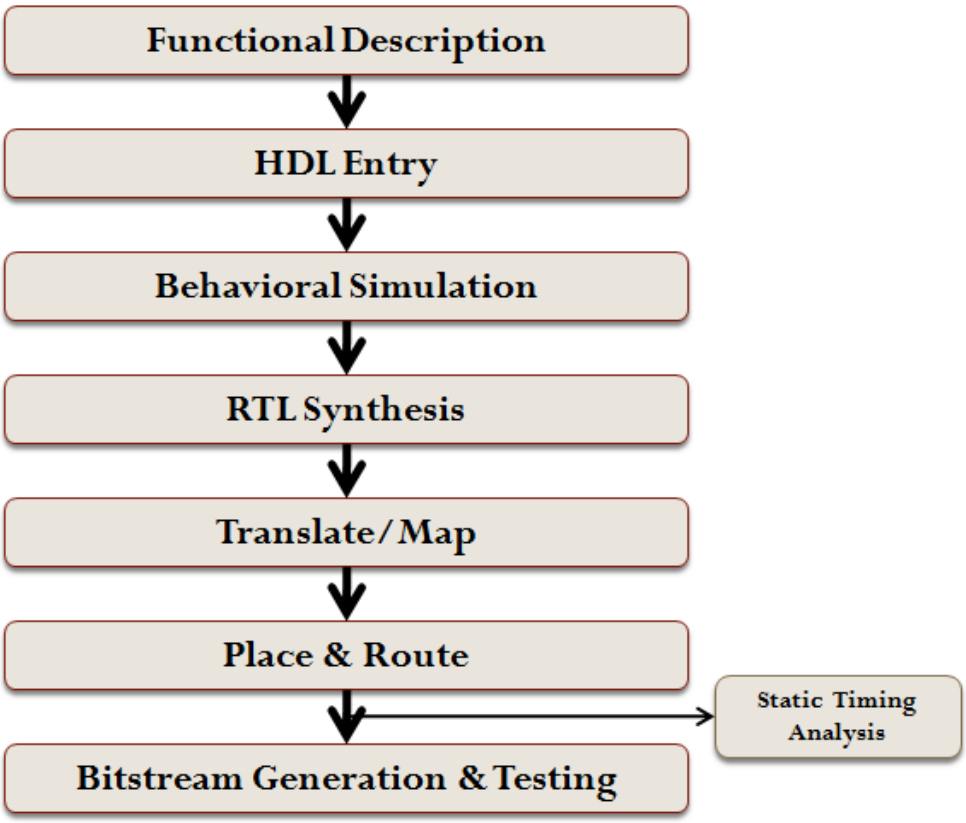

Figure 1.6: Typical FPGA design flow.

Designers generally use high-level HDL entry for their designs. Synthesis tools map these HDL designs into standard cell library. This mapping process is followed by the layout placement and metal routings of these cells, performed by place and route (PAR) tools. After the verification of the generated layout, the design can be sent to production [11]. Figure 1.7 shows typical standard cell ASIC design flow. As it can be seen, the design flow of the standard cell method is similar to the FPGA design flow, except for the final parts.

Today, FPGAs and ASICs are the two of the best-known and the most discussed methodologies of the integrated circuit design world. Both of them have some advantages and disadvantages over each other. Also, there are many reviews and comparisons between these two design methods. The common comparison points between these two are in terms of power, area, performance, non-recurring engineering (NRE) cost, unit cost and design time.

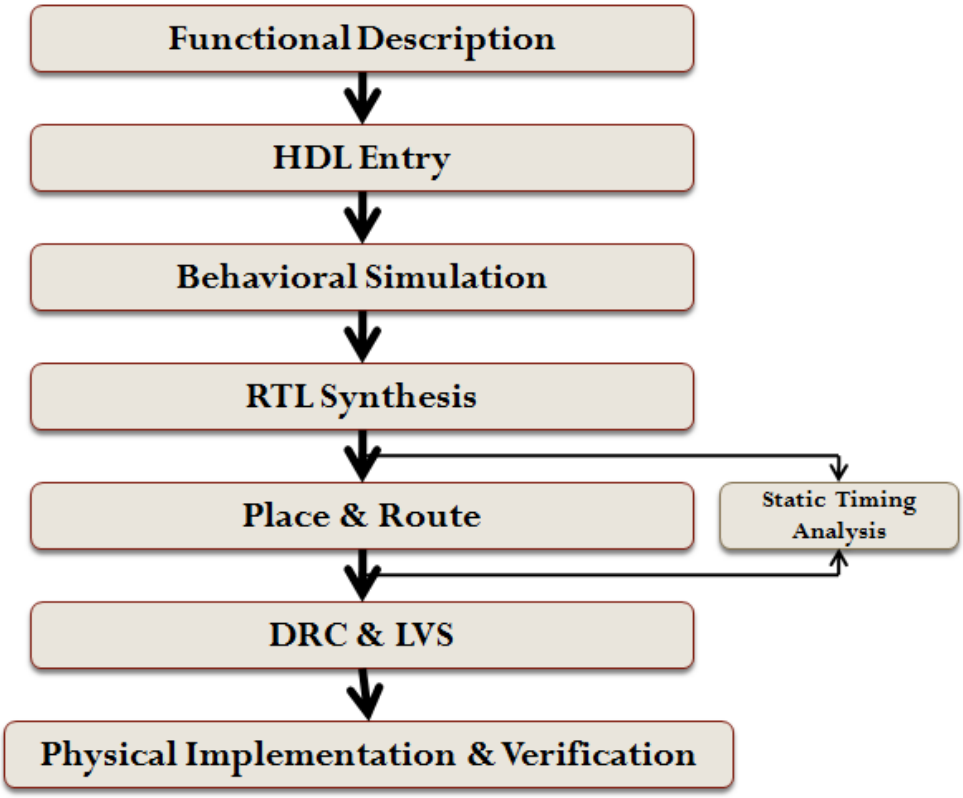

Figure 1.7: Typical standard cell ASIC design flow.

In terms of power consumption, ASICs are advantageous than the counterpart FPGA designs [12]. In the case of FPGA designs, extra gates and non-optimal usage of the design parts affect power dissipation in a negative way. On the other hand, while designing a low-power ASIC, power reduction techniques can be used. These techniques may include clock gating or power gating, which are impossible to apply or not wanted design methods in FPGAs.

Other comparison parameters between ASICs and FPGAs are area and performance characteristics. Benchmark tests conducted using multiple designs and both on standard cells and FPGAs show that ASIC designs are better in terms of area and performance [12].

One important thing should be added at this point. The power, area and performance parameters are susceptible to changes from design to design. The used technology nodes or FPGA devices also affect the results of these comparisons. In addition, the

optimization of the design, both on FPGA and ASIC, is an important factor. Usually, it is better to use specific IP cores in FPGA designs in terms of these parameters. Also it should be noted that FPGA vendors usually use the latest technology nodes, which is generally not the case for ASIC designs. This also closes the gap between ASICs and FPGAs [13].

Other than the discussed parameters, the cost of the design is also an important factor. The ASIC design process requires expensive computer-aided design (CAD) tools by its nature. The physical production of the designed parts is also an expensive process. These can be classified under the NRE costs, which is higher on ASIC designs. On the other hand, the NRE of the FPGA designs is relatively low. These cost calculations are meaningful with the volume of the part. As the production volume grows, the unit cost of the ASIC design will be smaller. However for low volume designs, using FPGAs will be a cheaper selection.

The last comparison point between the two main methods is the required design time. From the Figure 1.6 and Figure 1.7, it can be seen that the design processes for both methods are similar. However, the ASIC design steps require more attention and each step should be performed cautiously. On the other hand, most of the FPGA design steps are performed seamlessly by the tools. Although this difference creates a time gap, it is not the main reason of the different time requirements. The FPGA designs are ready to use after the last step, while the ASIC designs are only ready for the production in this stage. Time passed in the production line is the main reason for the design time differences between the two methods. Therefore, FPGA designs require significantly lower time to design. This also makes the small design changes possible in a short time.

The advantages and the disadvantages of both design methods are discussed in this section. Circuit designers should take these comparisons into account while designing their components. According to the design needs, the correct methods can be chosen.

1.3 Motivation for the Thesis

Up to now, general characteristics of imaging systems and design methods for digital integrated circuits are explained in general. As stated before, the thesis deals with the design of a digital ASIC for imaging applications. In this part, the reasons behind designing such a digital ASIC will be explained.

As explained before, image sensors require some external electronics for proper operation. Timing requirements, necessary programming operations and data acquisition can be given as examples to these requirements. In order to fulfill such requirements, digital integrated circuits are used. These circuits can be implemented using more than one ways. Software implementation, FPGA (or programmable logic device) implementation and custom ASIC implementation are well-known methods for these type of circuits.

Software implementation means that all the necessary communication and data acquisition are directly controlled from a host device. Besides the interface problems, it is difficult to implement fast and synchronous communication as well as lossless data acquisition. Therefore, it is better to use a device between the host and the sensor that handles those type of operations.

As an intermediate device, programmable logic devices can be used. Across the programmable devices, FPGA usage is a common solution. FPGAs are easy-to-use logic devices, however since the logic capacity and diversity are predefined by vendors, true optimization is not possible. Even if the functionality is ensured with FPGAs, unoptimized power and area parameters may be a problem for some designs. For image sensors, the best way of handling the necessary signals is placing the circuit as close as possible to the sensor. This minimizes wire delays and possible data loss. However, it is not desired to have high power and large circuits near the sensor due to the noise issues. This limits the FPGA usage in designs.

The unwanted cases can be eliminated using custom design methodology. With custom design, it is possible to obtain full functionality and circuit optimization. It is also the safest method for near-sensor operations in terms of correct programming and video data acquisition without any frame loss. This is one of the main reasons of

selecting ASIC implementation in this thesis.

In imaging systems, finding external electronics that are responsible for gathering image from the sensor is not an easy decision, since the performance of these parts directly affects the quality of the obtained image. Choosing proper devices from the market requires deep market knowledge as well as significant amount of time to decide whether the chosen part is appropriate for particular design. Also, it is not always possible to find devices whose properties meet the exact design criteria. Usually, designers are forced to choose parts with over-design features, or to sacrifice some features from their design to fit that part. These features can be the discussed points like area, power or performance. At the end, this dilemma can cause some unwanted consequences on the system level. Using custom ASIC implementation also eliminates such issues.

Custom ASIC implementation that supports sensor operations is a used method in the industry. Many different companies that work on the imaging sensors design such ASICs in order to handle sensor operations. FLIR, Teledyne and SRI can be given as examples to such companies [14] [15] [16]. The structure of these ASICs, however, can be different according to the requirements. The ASIC can be used to grab video, to digitize data or even fuse different image types.

Figure 1.8 shows MT6415CA imaging sensor with its external electronics designed by Mikro-Tasarım. The stack consists of a 640x512 imaging sensor coupled with an ADC card and an FPGA card. As it can be seen, the sensor itself occupies a small footprint compared with the other cards. The part that determines the size of this stack is the FPGA card. Although the complete system is small, it is possible to make it smaller by changing the peripherals.

In order to make the system smaller, the dimensions of the parts except the sensor should be decreased. Since the part that determines the size of the stack is the FPGA card, the best way of decreasing the dimensions is to use a custom made controller. This brings the design of an ASIC that includes all the digital parts like communication unit, data acquisition unit etc. This thesis explains the design of such a digital ASIC with smaller area and power demand than FPGA design. Following chapters explains the design progress with its verification, implementation and test phases.

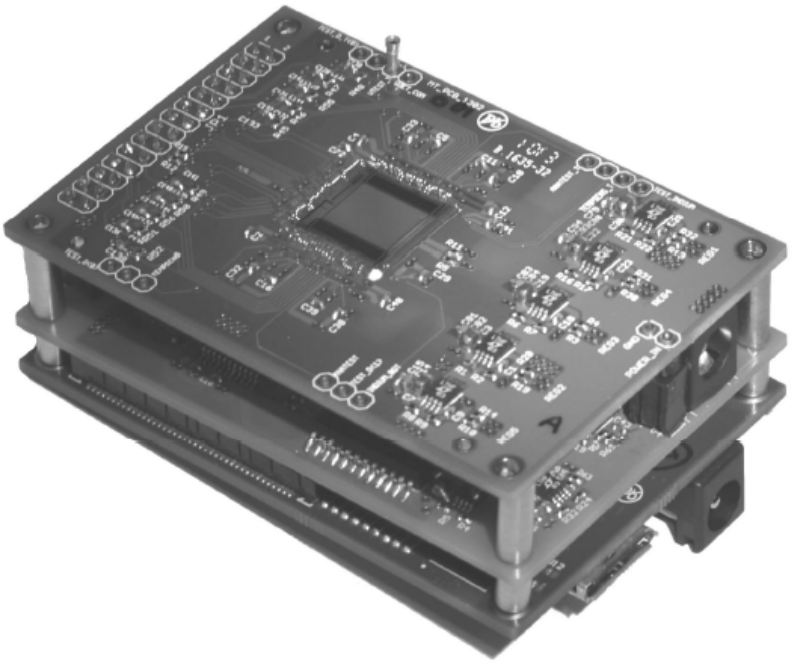

Figure 1.8: MT6415CA imaging sensor with its external electronics designed by Mikro-Tasarım[2].

CHAPTER 2

DESIGN OF THE DIGITAL ASIC

This chapter deals with the design of the digital ASIC that is used for imaging sensor drive and data acquisition applications. It explains the basic building blocks that are designed in the scope of this thesis. The chapter also provides some necessary background information on these blocks.

2.1 Design Requirements

As stated before, the digital ASIC is designed in order to control an imaging sensor with its peripherals. This section roughly explains the general building blocks of the digital ASIC with the requirements of the imaging system.

In this thesis, the imaging system refers to the combination of an image sensor, data serializers, and the designed digital ASIC. Figure 2.1 shows a simple block diagram of the imaging system studied in the thesis. As it can be seen from this figure, the parts except the sensor are enclosed with a dashed line and called as ASIC. Since the thesis deals with the digital parts of the complete ASIC design, the other parts like serializers are treated as external electronics according to the digital ASIC.

Before going deeper into the digital ASIC design, the requirements and specifications of peripherals should be examined. The following parts explains the peripherals that are used along with the designed digital ASIC. Note that for simplicity, the digital ASIC will be referred as ASIC from now on.

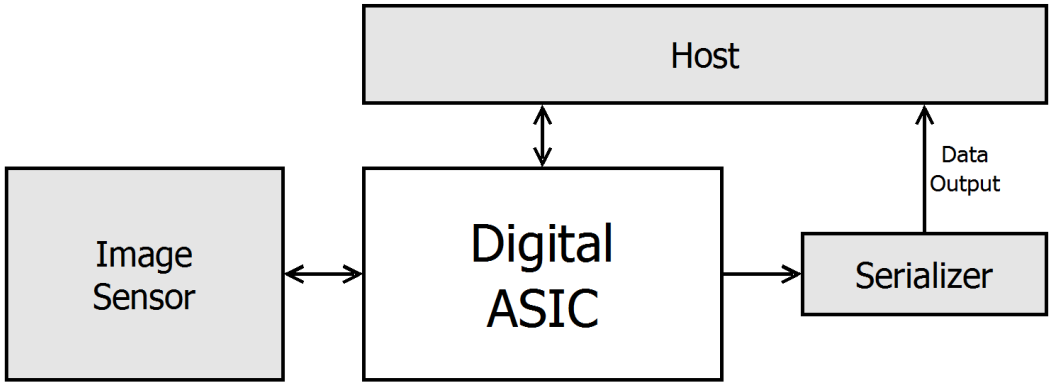

Figure 2.1: Simple block diagram of the imaging system studied in the thesis.

2.1.1 Image Sensor

The image sensor that is used with the ASIC is important since the interfaces between ASIC and this part are designed according to the selected sensor. For this design, the sensors with their readout circuits, which will form the imaging system, are chosen from Mikro-Tasarım's MT series products. Low number of pin requirement, easy programmability/controllability, data availability and ease of accessibility for testing and verification purposes are main reasons for choosing these products [2].

The ASIC is designed with the mentioned sensors in mind. However, the communication and video interfaces of these sensors are quite general. For communication, they use a standard 4-wire serial peripheral interface (SPI) bus in slave mode. For video output, they use Camera Link compatible synchronization signals to mark the valid pixel data. These signals are data valid (DVAL), line valid (LVAL) and frame valid (FVAL) signals as well as a pixel clock (PIXCLK) signal for sampling [17]. Figure 2.2 shows the video output of the imaging sensor with its synchronization signals.

Besides those video synchronization signals that are coming out from the sensor, one additional synchronization signal is also used between the ASIC and the sensor. The sensor synchronizes its frames according to the frame sync (FSYNC) signal. This signal is controlled from the ASIC and sent to the sensor. In order to follow and control the sensor operation, this signal is required. Figure 2.3 shows the video output of the imaging sensor with its synchronization signals and the frame sync signal.

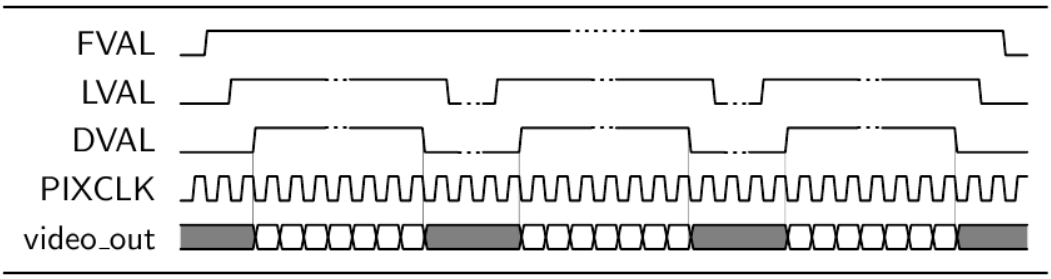

Figure 2.2: Video output of the imaging sensor with its synchronization signals.

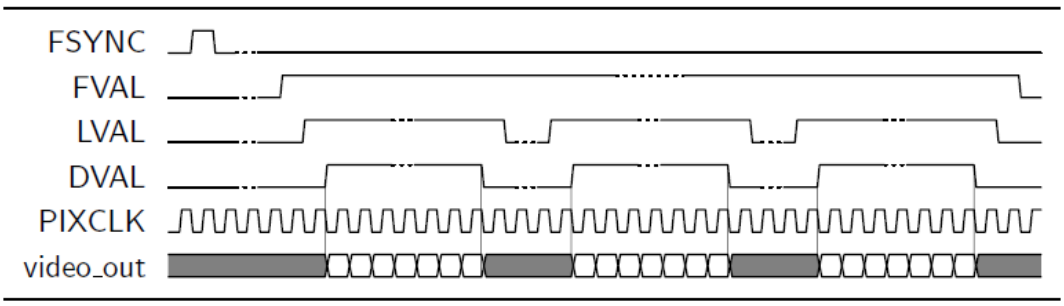

Figure 2.3: Video output of the imaging sensor with its synchronization signals and the frame sync signal.

In the Figure 2.3, the video output of the sensor is shown as one signal. However, in order to increase the video data rate without increasing the system clock frequency, more than one video channel can be used. The sensor that can be used along with the ASIC can have up to 4-channel video output, each has up to 10 MHz data rate. Therefore, the system clock of the sensor is taken as 10 MHz, which is also the serial communication clock frequency.

It is stated that the target products that can be used with the designed ASIC are Mikro-Tasarım's MT series products. As an example, the MT6415CA product can be given, which is mentioned in Chapter 1 as well. It offers all the features that are discussed in this section [2].

In the design of the digital ASIC, the image sensor outputs are assumed as digitized signals. In the case of analog image sensor outputs, they must be digitized using ADC circuits. This is the case for above-mentioned image sensor designed by Mikro-Tasarim.

2.1.2 Serializer

The ASIC outputs the data in a parallel way with a sampling clock. If the serial output is desired, an optional serializer can be used. There is not any strict specifications on this part. LVDS transmitters with serializers that are available in the market can be used according to the speed and data rate considerations.

2.1.3 Host

Host is the part which controls the designed ASIC. Control here means that the communication and programming requirements are fulfilled by the host controller. The output of the ASIC can also be collected by this host. The host controller may be a computer or another logic device as long as the interface between the ASIC and the host is securely formed.

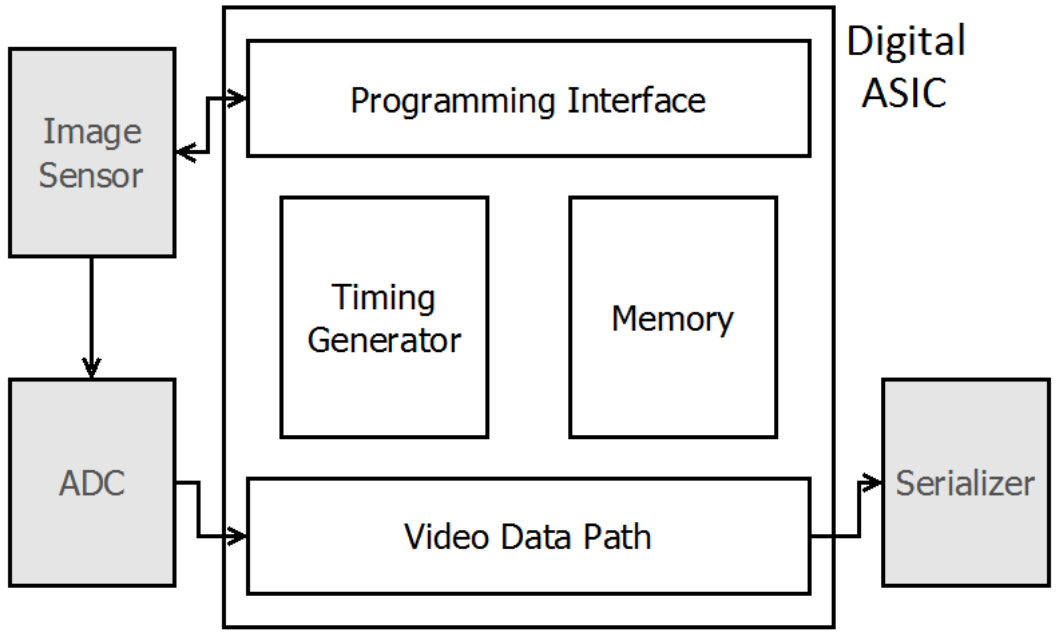

Figure 2.4: Block diagram of the ASIC with its peripherals.

2.1.4 Speed Requirements

The mention image sensors are designed to run at a native clock frequency of 10MHz. Therefore, the system clock of the sensor is chosen as 10MHz. This selection enforces the ASIC-sensor interface speed to the same clock frequency also. The video output of each sensor channel has also a 10MHz clock speed. Similarly, this speed enforces the speed of the data path of the ASIC.

Although the sensor runs at 10MHz system frequency, the ASIC system clock is chosen as 40MHz. However, not all the parts of the ASIC run at this frequency. While the communication (except the sensor interface), memory and programmable timing generator parts run at higher speeds; the sensor communication interface and the data path run with the sensor native frequency. The higher clock decreases the communication times and gives extra resolution in timing generation. Also the 4 channel video with each runs at 10MHz can be handled easily with the 40MHz clock. With those reasons in mind, the system clock frequency of the ASIC is chosen as 40MHz.

In this chapter, the imaging system parts are explained up to now. From now on, the modules that are designed for digital ASIC to control this imaging system parts will be explained.

2.2 ASIC Architecture

The ASIC mainly consists of 4 parts. These can be summarized as the serial communication module, memory, video data path and programmable timing generator. The video data path can be separated into 3 modules; which are FIFO, arithmetical and logical unit and 8B/10B encoder. Figure 2.4 shows the block diagram of the ASIC with its peripherals.

2.2.1 Programming Interface

The serial communication module in the ASIC is responsible from the communication between the digital ASIC, image sensor and host controller. The communication between those parts can be categorized as host to ASIC and ASIC to image sensor interfaces. Figure 2.5 shows the detailed block diagram of the ASIC programming interface with the ASIC memory block.

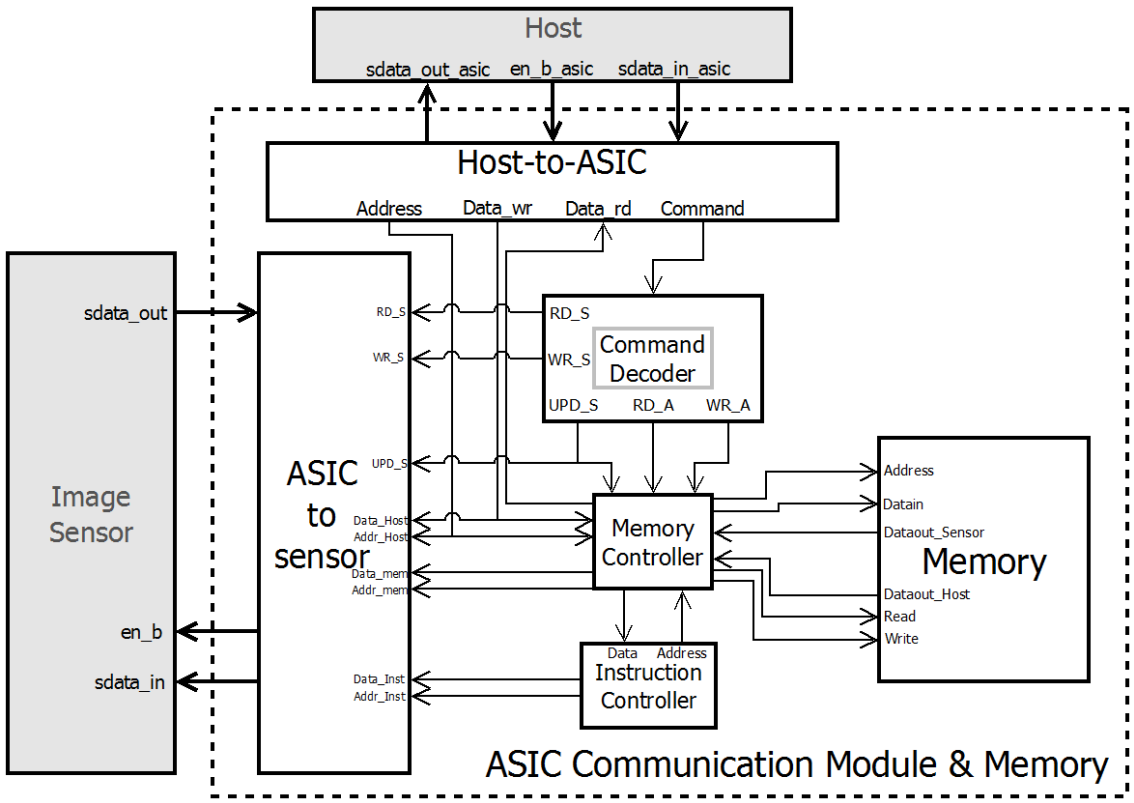

Figure 2.5: Detailed block diagram of the ASIC programming interface with the ASIC memory block.

ASIC includes a serial communication interface in order to provide an easy programming to the users. In order to achieve this, a total number of 3 pins are defined on the ASIC. Table 2.1 shows ASIC serial communication pins and their explanations.

The *"sdata_in"* pin is the main input for programming operations. The commands and necessary data are sent through this pin. Communication through this pin requires 8-

Table2.1: ASIC serial communication pins and their explanations

Pin	Definition	Direction
sdata_in	Serial data input	Input
sdata_out	Serial data output	Output
en_b	Serial communication enable	Input

bit command followed by 8-bit address and 16-bit data if necessary. Figure 2.6 shows the timing diagram of the serial interface signals with respect to the system clock.

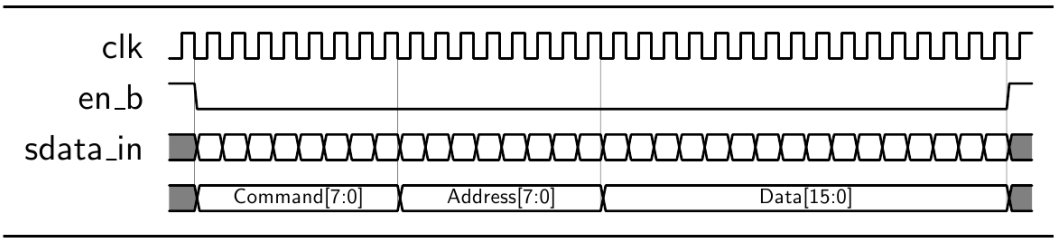

Figure 2.6: Timing diagram of the serial interface signals with respect to the system clock.

The one byte wide command part stands for the programming commands that the ASIC demands in order to operate properly. Table 2.2 gives the defined commands for the ASIC and their descriptions.

Table2.2: Defined commands for the ASIC and their descriptions.

Code(Hex)	Abbreviation	Explanation	Operands
0x81	WR_A	Write to ASIC memory	Address | Data
0x18	RD_A	Read from ASIC memory	Address
0xC3	WR_S	Write to sensor memory	Address | Data
0x3C	RD_S	Read from sensor memory	Address
0xCC	UPD_S	Update sensor memory	

WR_A: This operation represents writing data to the ASIC memory block. ASIC memory block consists of the programmable bits and timing configurations on its lower (lower addressed) half and image sensor configuration memory on its upper (higher addressed) half. Any write operation intended to program these parts should use this command. The "WR_A" command requires an 8-bit address afterwards,

as well as an 16-bit data followed by this address. This one byte address points the location on the memory that will be edited. The remaining two bytes data is written to this memory location at the end of this operation. With these operands, the execution of "WR_A" command lasts 32 clock cycles in total. Figure 2.6 can be given as the timing diagram.

RD_A: This command is applied by the host when a data in the ASIC memory is intended to be read. The operation is similar to the write operation except the data operand. In order to point the place from where the data will be read, one-byte long address is enough. After applying this address, the data can be read from the *"sdata_out"* pin in the next communication cycle, i.e. driving *"en_b"* pin high and again low. Figure 2.7 shows the ASIC serial interface timing diagram for the RD_A command.

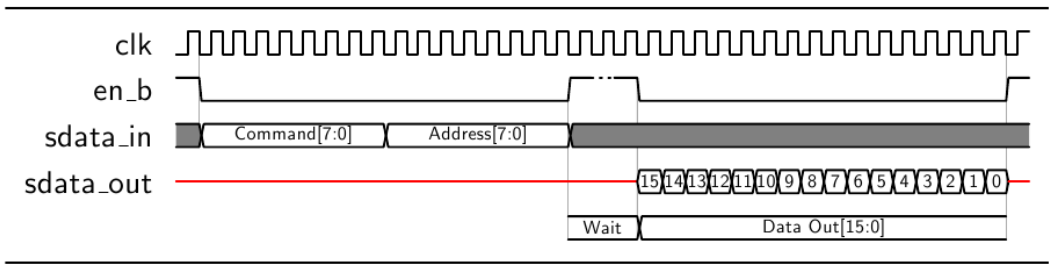

Figure 2.7: ASIC serial interface timing diagram for the RD_A command.

The wait time between the last bit of the address and the first bit of the outgoing data needs an explanation. Since reading from memory is a time consuming operation, the design implementation should be safe. In this design, the wait time should be at least 8 clock cycles for safe operation.

WR_S: The sensor write command is similar to the ASIC write command (WR_A). After this command is sent, address and data bits should be transferred sequentially. Figure 2.6 can be given as the timing diagram for this operation.

One more thing to know about this operation is that, since a mirror of the sensor configuration memory is located inside the ASIC memory, any direct write operation to sensor will cause a mismatch between sensor memory and sensor configuration memory located in the ASIC. If users want to keep track of the changes in the sensor

memory, this should be taken into account.

RD_S: This command is used when the user wants to read a word from the sensor memory. "RD_S" operation is similar to the "RD_A" operation. However the wait time that is discussed above is larger since a sensor communication operation takes place. For a safe communication, this delay should be at least 280 clock cycles. The large wait time is due to the requirement of two communication operations for reading and lower clock frequency at the sensor side. Figure 2.7 shows the timing diagram for this operation, which is the same with the "RD_A" command timing diagram.

UPD_S: The "UPD_S" command is used when user wants to update a large memory block in the sensor. This large block can include the whole sensor memory. The programming interval is configurable through ASIC memory with start and end addresses. The main reason for designing this programming option is to configure the sensor memory with single command, if the sensor configuration memory is already programmed in a desired way. For execution of "UPD_S", only the hex command is enough.

With this command, the user can use the sensor configuration memory instead of using direct sensor memory access (WR_S). With this way, it becomes easy to keep changes of the sensor memory since the sensor memory content is the same with the sensor configuration memory located in ASIC.

The valid commands for the serial communication unit of the ASIC are explained. As it can be seen from the timing diagrams and figures, the required time for one communication cycle can change from 8 clock cycles (i.e. "UPD_S") to 32 clock cycles (i.e. "WR_S") according to the command needs. If uniform time requirement is desired, 32 clock cycle wide communication can be used for all commands. The unused parts like address and data together (in the case of "UPD_S") or data alone (in the case of "RD_A and "RD_S") will be ignored in these cases. Although communication lasts a bit longer, it is easier to handle all commands with this way.

Like the ASIC itself, in order to configure image sensor that is connected to the ASIC, dedicated serial ports are added to the design. Table 2.3 shows the ASIC-to-sensor serial communication pins and their explanations.

Table2.3: ASIC to sensor serial communication pins and their explanations.

Pin	Definition	Direction
sdata_in_sensor	Sensor serial data input	Output
sdata_out_sensor	Sensor serial data output	Input
en_b_sensor	Sensor serial communication enable	Output

The communication between the image sensor and the ASIC is designed as similar to the ASIC serial interface. However in this mode, the length of the communication is 24 bits instead of 32 bits. Figure 2.8 shows the programming timing diagram of the imaging sensor.

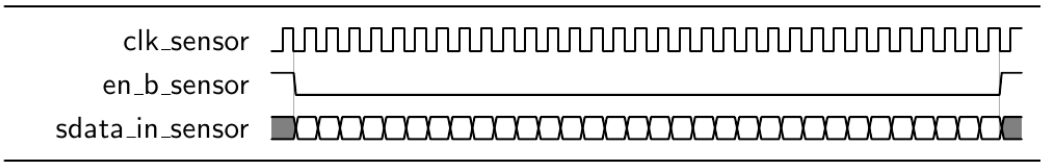

Figure 2.8: Programming timing diagram of the imaging sensor.

2.2.1.1 Instruction Controller

The programming interface includes an instruction controller, which can be seen in the Figure 2.5. This part is designed to control the sensor with periodical commands and it has its own instruction set. The first quarter of the ASIC memory is reserved for such instructions. These instructions are executed sequentially, with the help of an address counter and a state machine. Figure 2.9 shows the state machine of the instruction controller included in the programming interface.

The instructions that are defined for this special block is selected for periodical sensor operations. There are 3 instructions and the Table 2.4 gives the defined instructions for instruction controller and their descriptions.

"Write to sensor memory" instruction occupies 2 words in the ASIC memory. Fig-

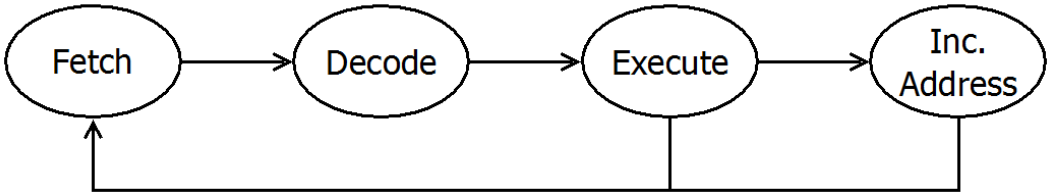

Figure 2.9: State machine of the instruction controller included in the programming interface.

Table2.4: Defined instructions for instruction controller and their descriptions.

Code(Hex)	**Explanation**
0xC3	Write to sensor memory
0x1F	Run the image sensor for 1 frame
0xFF	Go to address

ure 2.10 shows the memory map for the "Write to sensor memory" instruction. In this figure, "Address[7:0]" corresponds to the sensor memory address that will be programmed and the "Data[15:0]" corresponds to the data that will be written to this address.

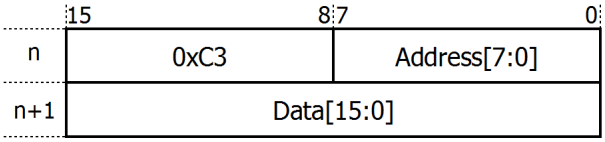

Figure 2.10: Memory map for the "write to sensor memory" instruction.

"Run the image sensor for 1 frame" instruction occupies 1 word in the ASIC memory. When this command is executed, the ASIC runs the sensor for one frame time. Figure 2.11 shows the memory map for the "run the image sensor for 1 frame" instruction.

"Go to address" instruction occupies 1 word in the ASIC memory. When this instruction is executed, the address counter is reset to the "Address[7:0]" value. With this way, the periodicity of whole instruction memory is ensured. Figure 2.12 shows the memory map for the "go to address" instruction.

Since this part is a bit confusing, it is better to explain it with an example. Figure 2.13

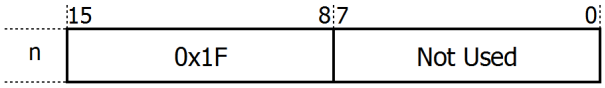

Figure 2.11: Memory map for the "run the image sensor for 1 frame" instruction.

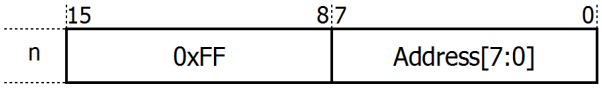

Figure 2.12: Memory map for the "go to address" instruction.

shows the memory map for an example usage of the instruction controller. In this example the two sensor memory locations "Address0" and "Address1" are programmed with different data inputs in sequential frames. This operation is repeated as long as the instruction controller is enabled.

	15 — 8	7 — 0
0	Write to Sensor	Add0[7:0]
1	Data0-0[15:0]	
2	Write to Sensor	Add1[7:0]
3	Data1-0[15:0]	
4	Run 1 Frame	N/A
5	Write to Sensor	Add0[7:0]
6	Data0-1[15:0]	
7	Write to Sensor	Add1[7:0]
8	Data1-1[15:0]	
9	Run 1 Frame	N/A
10	Go to	0

Figure 2.13: Memory map for an example usage of the instruction controller. In this example the two sensor memory locations "Address0" and "Address1" are programmed with different data inputs in sequential frames.

2.2.2 Memory

Like many other digital programmable electronics, the ASIC contains memory blocks in order to write and store the desired settings. This memory has a capacity of 256 words, with each word has 16 bits. The configuration settings about the whole ASIC as well as the image sensor are stored in this block. Programmable features are also written to and stored in this memory. Figure 2.14 shows the map of the complete ASIC memory module.

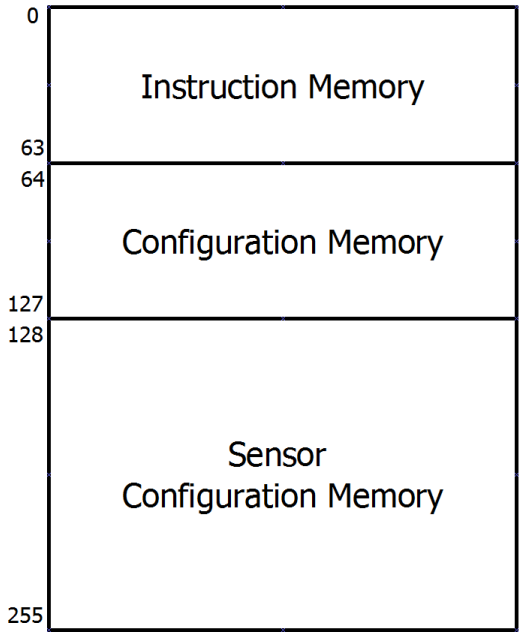

Figure 2.14: Map of the complete ASIC memory module.

The first quarter of the memory is reserved for the instructions, which are explained in the instruction controller part. The second quarter is used for the ASIC configuration; in which the arithmetical unit, encoder and programmable timing generator settings are included. The last half of the ASIC memory is reserved for the imaging sensor. This 128x16 bit part can be used to store the desired sensor settings.

The memory is controlled by a memory controller. The main purpose to use such a controller is to organize the communication between multiple modules and the ASIC memory. This communication consists of two main operations which are write and read. The write operation is controlled from only one port which is accessible from

the host interface. Using one port for writing ensures safe write operation. For the read operation, however, two independent ports are used. One of these ports is reserved for the host read operations while the other one is reserved for the sensor read operations. Choosing two ports for read operation prevents possible conflict between host and sensor read operations that are executed at the same time. With this way, safe read operation is also ensured.

2.2.3 Data Path

The data path of the digital ASIC is the part where the image sensor output acquisition is performed. The main reason of this path is to construct a bridge between the sensor and the host. While constructing such a bridge, some arithmetical and logical operations as well as error reduction techniques on the data are taken into account. For those reasons, a FIFO, an arithmetical processing unit and an encoder is placed through this path. Following sub-sections explain these units in detail.

2.2.3.1 FIFO

In order to separate different clock domains in the ASIC, asynchronous FIFO structure are used. Asynchronous FIFO refers to a FIFO type whose write and read clocks are different from each other. In the applications that require data transmission between different clock domains, these FIFO structures are used as buffers between these domains. In ASIC, this structure is used to achieve a safe data acquisition from the sensor.

The imaging sensor works with a clock that is generated from the ASIC, however, returns its video data with its own pixel clock. The ASIC, on the other hand, uses its own clock while performing data path operations. although the speeds of these clocks are identical, their phase relation may differ. In order to solve the uncertainty between the pixel clock of the sensor and the clock of the ASIC data path, an asynchronous FIFO is used in the beginning of the data path.

There are many ways to design an asynchronous FIFO. The most important point

while designing one is to assure that data loss does not occur while transferring data from one clock domain to another. In order to achieve this, a structure with write and read pointers are used along with the empty and full indicators since it is safe and easy to implement in both FPGA and ASIC designs[3]. Figure 2.15 shows the block diagram of the FIFO structure. Since data storage is not an aim of this FIFO, the memory used in this structure is very small.

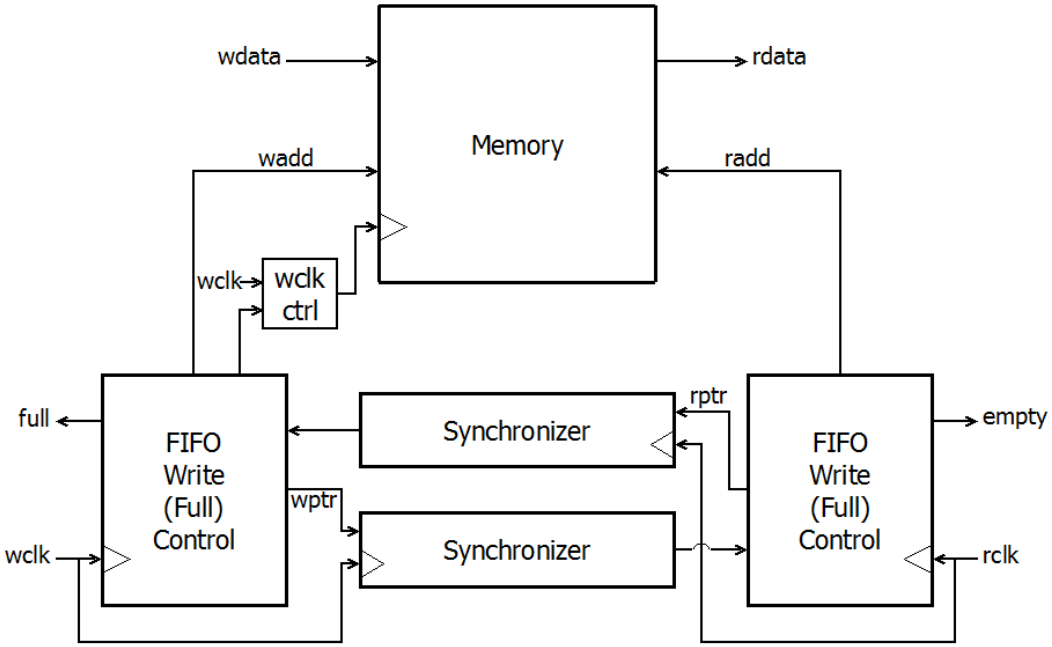

Figure 2.15: Block diagram of the FIFO structure.[3].

2.2.3.2 Arithmetical Processing Unit

The ASIC data path includes an arithmetical processing unit to apply some arithmetical and logical operations on the incoming data. The incoming data represents the digitized video outputs of the imaging sensor. The unit has both arithmetical and logical parts. The arithmetical part can perform simple arithmetical operations like addition or subtraction as well as simple multiplication and division. The logical part, on the other hand, can apply basic logical operations like AND, OR, XOR and taking the complement of the incoming data. Table 2.5 gives the defined function list of the arithmetical processing unit.

Table2.5: Defined function list of the arithmetical processing unit

4-Bit Control Input				**Operation**
0	0	0	0	G=A
0	0	0	1	G=A+B
0	0	1	0	G=A+B'+1
0	0	1	1	G=A*C
0	1	0	0	G=A*C+B
0	1	0	1	G=A»1
0	1	1	0	G=A«1
0	1	1	1	G=B
1	0	0	X	G=A&B
1	0	1	X	G=A|B
1	1	0	X	G=A$\oplus$B
1	1	1	X	G=A'

In the Table 2.5, the control input 'S' as well as the operand inputs 'B' and 'C' are configured from the device memory. The input 'A' refers to the incoming video data. Finally, 'G' refers to the output of the module[18].

Here, A and B operands are designed as 16-bit digital buses. 16-bit arithmetical and logical operations that are listed in the Table 2.5 can be conducted using those inputs. C operand, on the other hand, has 12-bit data width which controls the multiplication and division operations with a shift-and-add structure. Table 2.6 shows the explanation of the bits of the multiplication operation defined in the arithmetical processing unit. The final result is calculated by adding the results of the active shift operations. The more accurate representation of the 'C' operand can be stated as a fixed-point unsigned binary number with a scaling factor of $1/2^8$.

As an example, if the user wants obtain "1.25*A" with this multiplication operation, the 'C' operand should be programmed as "0x140".

2.2.3.3 8B/10B Encoder

In order to achieve an error-free communication at the LVDS drivers end, an 8B/10B encoder is placed at the end of the data path.

Table2.6: Explanation of the bits of the multiplication operation defined in the arithmetical processing unit .

C[11:0]	Explanation	Result (A*C)
100000000000	Multiply with 8	A*8
010000000000	Multiply with 4	A*4
001000000000	Multiply with 2	A*2
000100000000	Multiply with 1	A*1
000010000000	Divide with 2	A/2
000001000000	Divide with 4	A/4
000000100000	Divide with 8	A/8
000000010000	Divide with 16	A/16
000000001000	Divide with 32	A/32
000000000100	Divide with 64	A/64
000000000010	Divide with 128	A/128
000000000001	Divide with 256	A/256

8B/10B encoding is proposed in 1983, by A. X. Widmer and P. A. Franaszek[19]. In this encoding, 8 bit data is converted into 10 bit data by adding some redundancy. The reason behind that is to achieve a DC balanced communication and characterize maximum run-length. DC balanced communication is important in high speed transfers to maintain maximum noise margin at differential terminals[20]. This reduces the bit error rate. Figure 2.16 shows a possible error situation in the case of unbalanced DC communication. On the other hand, the maximum run-length is important for clock recovery at the receiver end[19]. In the applications that do not require a transfer of a separate clock signal, the receiver uses incoming data to estimate data rate. Therefore, upper-bounded run-length is significant in such communications.

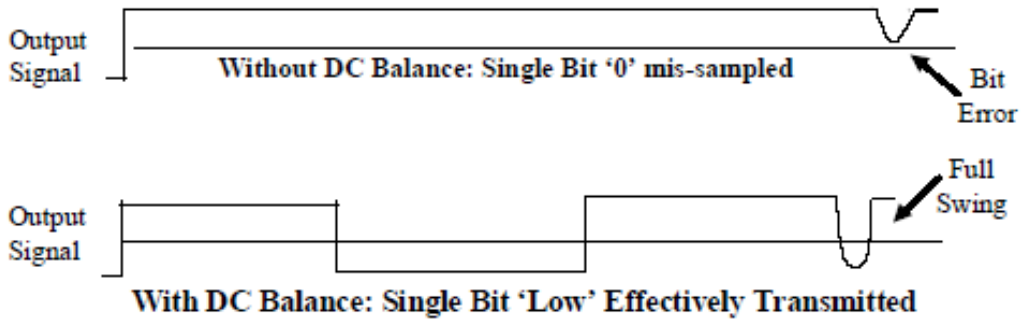

Figure 2.16: A possible error situation in the case of unbalanced DC communication [4].

The 8B/10B encoder consists of one 5B/6B encoder and one 3B/4B encoder that are connected together. Suppose that the 8 bit code that will be encoded is *"ABCDE-FGH"*. Than the 5B/6B encoder part converts *"ABCDE"* to *"abcdei"* which is a 6-bit wide symbol. Similarly, the 3B/4B part converts *"FGH"* to *"fghj"* which is a 4-bit wide symbol. The final 10-bit wide encoded symbol will be *"abcdeifghj"*.

The encoder provides DC balance by constantly checking the disparity of send data. Disparity means the difference between the number of 1's and 0's in a given block of binary data. If these numbers are equal to each other, the disparity is 0. However, if the number of 1's are greater, the disparity is positive. Similarly, if the number of 0's are greater, the disparity is negative.

In this encoding algorithm, each 6B and 4B blocks are confined to have disparities -2, 0 and +2. Additionally, the ones with nonzero disparities have inverted pairs which can be decoded to the same 5B or 3B symbols. According to the state of running disparity, control unit selects the correct code to secure DC balance at the line. Table 2.7 and Table 2.8 shows the 5B/6B encoding scheme and 3B/4B encoding scheme, respectively.

In Table 2.7 and Table 2.8, *"ABCDE"* or *"FGH"* represents the code that will be encoded by this module, from the least significant bit to the most significant bit. Similarly, *"abcdei"* or *"fghj"* represents the encoded code. The *"D-1"* column stands for the previous value of the running disparity and the *"D"* shows the disparity of the current encoded stream. Finally, *"Alternate"* field shows the alternative code for the inverse running disparity value. According to this running disparity value, one of the alternate codes is selected for encoding.

One more advantage of the 8B/10B encoding is that it allows to use special characters other than ordinary codes. These special characters can be used for some user defined tasks, like marking the start or end of a data packet or defining commands like reset. In order to define those special characters, one more input is added to the 8B *"ABCDEFGH"* input, which is *"K"*. Table 2.9 shows the special characters that are defined in 8B/10B encoding.

There are multiple implementation methods for 8B/10B encoding/decoding. Logic

Table2.7: 5B/6B encoding scheme.

Name	***ABCDE***	***K***	**D-1**	***abcdei***	**D0**	**Alternate**
D.0	00000	0	+	0**11**000	-	100111
D.1	10000	0	+	1000**1**0	-	011101
D.2	01000	0	+	0100**1**0	-	101101
D.3	11000	0	x	11000**1**	0	
D.4	00100	0	+	0010**1**0	-	110101
D.5	10100	0	x	10100**1**	0	
D.6	01100	0	x	01100**1**	0	
D.7	11100	0	-	111000	0	000111
D.8	00010	0	+	0001**1**0	-	111001
D.9	10010	0	x	10010**1**	0	
D.10	01010	0	x	01010**1**	0	
D.11	11010	0	x	110100	0	
D.12	00110	0	x	00110**1**	0	
D.13	10110	0	x	101100	0	
D.14	01110	0	x	011100	0	
D.15	11110	0	+	1**0**1**0**00	-	010111
D.16	00001	0	-	0**11**01**1**	+	100100
D.17	10001	0	x	10001**1**	0	
D.18	01001	0	x	01001**1**	0	
D.19	11001	0	x	110010	0	
D.20	00101	0	x	00101**1**	0	
D.21	10101	0	x	101010	0	
D.22	01101	0	x	011010	0	
D.23	11101	0	-	111010	+	000101
D.24	00011	0	+	00**11****0**0	-	110011
D.25	10011	0	x	100110	0	
D.26	01011	0	x	010110	0	
D.27	11011	0	-	110110	+	001001
D.28	00111	0	x	001110	0	
D.29	10111	0	-	101110	+	010001
D.30	01111	0	-	011110	+	100001
D.31	11111	0	-	1**0**1**0****1****1**	+	010100

implementation and look up table implementation are the two most preferred methods. Logic implementation is chosen in this design because of its low area demand and HDL friendly structure. In the original paper, the 8B/10B encoding is performed so that the bit change occurs from the input to the output is minimum[19]. This also

Table2.8: 3B/4B encoding scheme.

Name	***FGH***	***K***	**D-1**	***fghj***	**D0**	**Alternate**
D.x.0	000	0	+	0**1**00	-	1011
D.x.1	100	0	x	100**1**	0	
D.x.2	010	0	x	010**1**	0	
D.x.3	110	0	-	1100	0	0011
D.x.4	001	0	+	0010	-	1101
D.x.5	101	0	x	1010	0	
D.x.6	011	0	x	0110	0	
D.x.P7	111	0	-	1110	+	0001
D.x.A7	111	0	-	**0**11**1**	+	1000

Table2.9: Special characters that are defined in 8B/10B encoding.

Name	***ABCDE***	***FGH***	***K***	**D-1**	***abcdei***	***fghj***	**D0**
K.28.0	00111	000	1	-	001111	0100	0
				+	110000	1011	0
K.28.1	00111	100	1	-	001111	1001	+
				+	110000	0110	-
K.28.2	00111	010	1	-	001111	0101	+
				+	110000	1010	-
K.28.3	00111	110	1	-	001111	0011	+
				+	110000	1100	-
K.28.4	00111	001	1	-	001111	0010	0
				+	110000	1101	0
K.28.5	00111	101	1	-	001111	1010	+
				+	110000	0101	-
K.28.6	00111	011	1	-	001111	0110	+
				+	110000	1001	-
K.28.7	00111	111	1	-	001111	1000	0
				+	110000	0111	0
K.23.7	11101	111	1	-	111010	1000	0
				+	000101	0111	0
K.27.7	11011	111	1	-	110110	1000	0
				+	001001	0111	0
K.29.7	10111	111	1	-	101110	1000	0
				+	010001	0111	0
K.30.7	01111	111	1	-	011110	1000	0
				+	100001	0111	0

minimizes the logic that is required for implementation.

The basic idea behind this logic implementation is to determine changed bits in encoding operation and fit those changes in a logic function.

2.2.4 Programmable Timing Generator

The ASIC includes general purpose timing generators for user needs. The timing generators are designed for image sensors that require external signals to run. These modules have programmability options in terms of pulse widths, repetition and polarity. There are a total number of 8 such blocks and their configurations come from internal ASIC memory. If user wants to produce a desired timing, the related memory locations should be programmed. Also note that this module tracks the timing of the connected sensor with internal counters. Synchronization of these timing signals is handled with this way.

Each timing generator programmability options occupies 4 words in the ASIC memory. The 32 location whose addresses lie between 78 and 109 can be used to program these timings. Table 2.10 shows the memory map for one timing generator block. In this table, *"toggle_pt0"* and *"toggle_pt1"* represents the toggle points of the signal in one line time. For example, if 100 clock cycle pulse width is desired which starts at the 250th clock of the line, then these memory locations should be programmed as 250 and 350, respectively.

All the programmability options of the timing signals lie in the *"configuration[15:0]"* bits. Table 2.11 shows the explanation of the configuration bits for one timing generator block. These options are selected according to the needs of the imaging sensors.

In this chapter, the basic building blocks of the digital ASIC are explained with the requirements of the imaging system. In the next chapter, the simulation results of the

Table2.10: Memory map for one timing generator block.

Address*	Bits	Explanation
0	toggle_pt0[15:0]	First toggle point in line
1	toggle_pt1[15:0]	Second toggle point in line
2	active_line[11:0]	Line number that the signal is active**
3	configuration[15:0]	Timing configuration bits
*Indicates the address in the 4-word memory block **If the signal is one-at-a-frame		

Table2.11: Explanation of the configuration bits for one timing generator block.

Bit No.	Value	Explanation
0	0	Output invert disabled
	1	Output invert enabled
1	0	Output is not tied to static value
	1	Output is tied to static 0 (1 if invert is enabled)
2	0	One-at-a-line signal
	1	One-at-a-frame signal
3	0	Active at every line (or frame, see Bit 2)
	1	Active at once in every two line (or frame, see Bit 2)
4	0	If Bit 3 is 1, active at even lines
	1	If Bit 3 is 1, active at odd lines
5	0	Disabled
	1	Enabled
6-15	-	Not used, reserved for future use

designed blocks will be explained.

CHAPTER 3

SIMULATIONS

This chapter explains the simulations and design verifications of the ASIC. For simulations, Xilinx ISE Design Suite is used along with its simulator ISIM [21]. The building blocks of the ASIC which are explained in the previous chapters are designed using Verilog HDL and simulated using above-mentioned tools.

3.1 Serial Communication Unit and Memory

The serial communication unit and memory together form the programming interface of the ASIC. The communication module also creates a bridge structure between the host and the image sensor. In order to verify the operation of those modules properly, they have simulated with the attached models of the image sensor interface and sensor memory. Figure 3.1 shows the simulation setup for the serial communication unit and the memory.

The flow of this block can be summarized as follows: First, the host to ASIC interface samples incoming data that is coming from the host controller. Then, this data is shaped to meaningful command, address and data parts. The command part is fed to the command decoder and if it is a valid selection, related code flag is asserted. With this code flag, the module associated with the given code performs its operation. When operation is finished, command flags are cleared to be ready for the next operation.

The host to ASIC communication part is responsible from obtaining necessary data

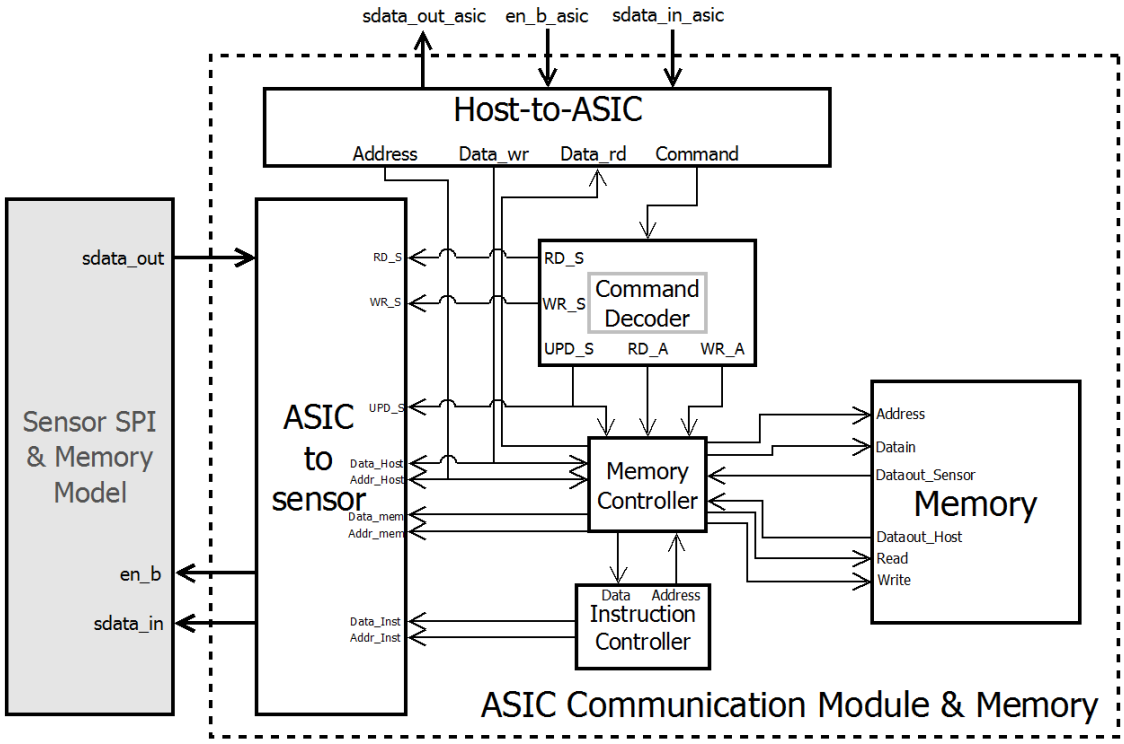

Figure 3.1: Simulation setup block diagram for the serial communication unit and the memory

at the input pins. The data at the *"sdata_in"* pin is sampled through this block. After that; command, address and data codes are sent to the related blocks. Figure 3.2 shows the simulation results of the host-to-ASIC interface for one communication cycle. In this short simulation, the "WR_S" command is sent to the ASIC whose corresponding hexadecimal value is "0xC3". After this command, a random address and data are sent in order to show the order and timing of this communication. As it can be seen, after each partition (command, address or data) of the incoming data, associated *"ready"* signals are formed to inform related blocks.

The valid communication commands of the ASIC are discussed in Chapter 2. Table 2.2 gives the available commands and their explanations. Figure 3.3 shows the simulation results of the command decoder with valid commands. Note that the flags are cleared manually in this figure to show command decoding operation properly. Normally these flags are cleared by the related block, at the end of the decoded operation.

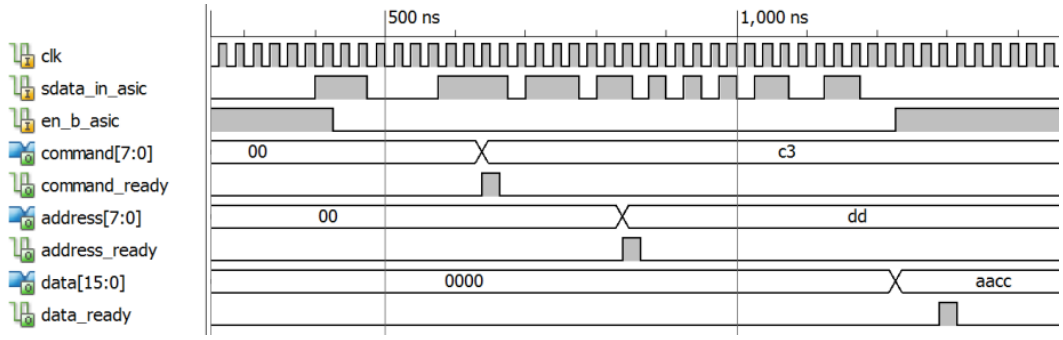

Figure 3.2: Simulation results of the host-to-ASIC interface for one communication cycle.

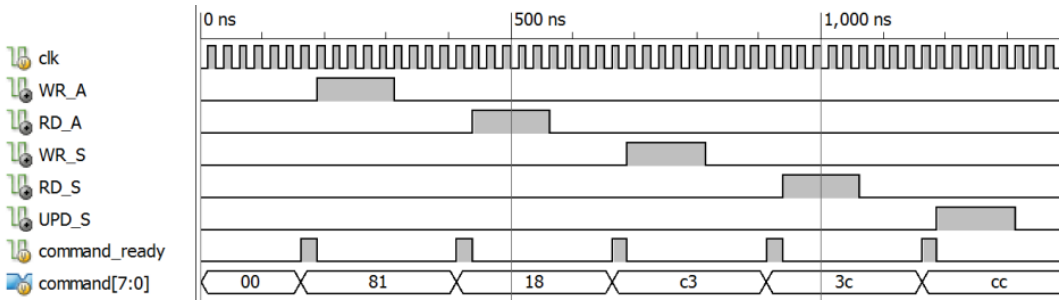

Figure 3.3: Simulation results of the command decoder with valid commands.

The "WR_A" and "RD_A" commands are related with the ASIC memory. These commands are used to write to and read from this memory, respectively. Figure 3.4 and Figure 3.5 shows the simulation results of the host-to-ASIC interface with the "WR_A" and "RD_A" commands, respectively.

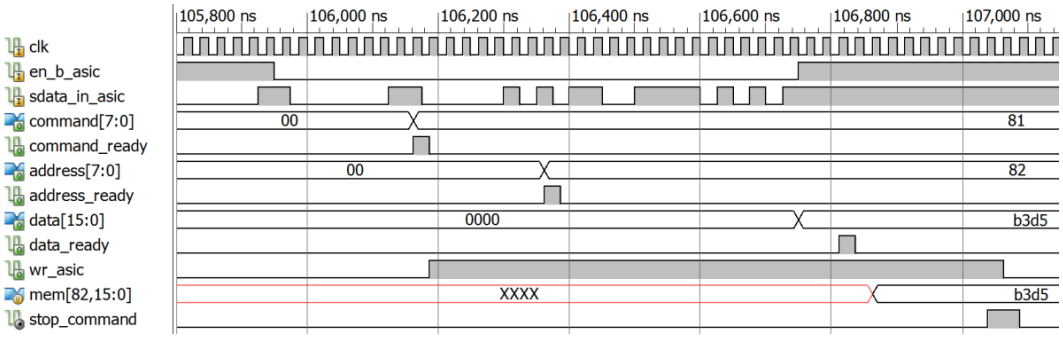

Figure 3.4: Simulation results of the host-to-ASIC interface with the "WR_A" command.

In Figure 3.4, the "WR_A" command is executed. As it can be seen, the address

"0x82" is programmed with the data "0xB3D5". Here, the "wr_asic" signal shows the decoded command and the "stop_command" shows the end of the command, thus it represents the reset of the flag. Finally, the 16-bit wide "mem" data shows the data of the desired memory location.

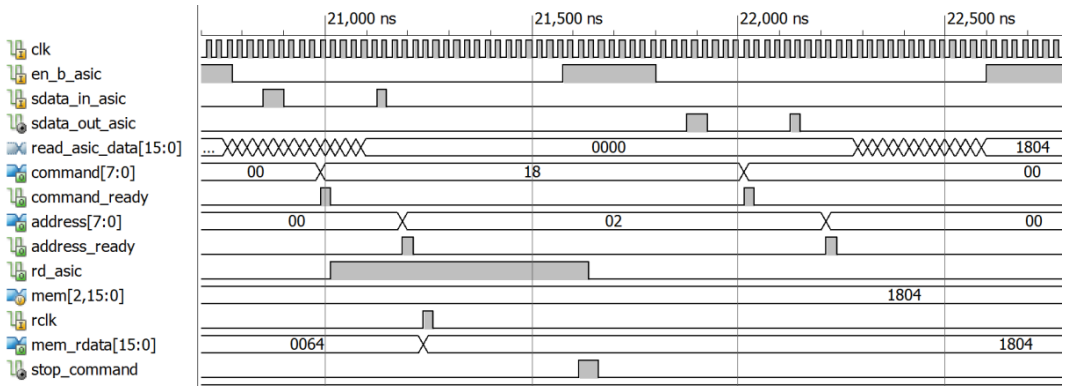

Figure 3.5: Simulation results of the host-to-ASIC interface with the "RD_A" command.

In Figure 3.5, the "RD_A" command is executed. As it can be seen, the address "0x02" is desired to be read. Here, the "rd_asic" signal shows the decoded command and the "stop_command" shows the end of the command, thus it represents the reset of the flag. The 16-bit wide "mem" data shows the data of the desired memory location. In this command, the important point is that the result of the read operation is output at the second communication cycle, from the "sdata_out_asic" pin. As it can be seen from the parallelized output "read_asic_data[15:0]", which is the result of the operation, the content of the memory location "0x02" is transferred to the output as "0x1804".

The next block is the ASIC to sensor communication module, which organizes the interface between ASIC and the image sensor. When the operation is a sensor read/write operation, this block is activated. There are 3 commands in this category, which are "WR_S", "RD_S" and "UPD_S". Simulation results for these three commands are given in the following parts.

Figure 3.6 shows the simulation results of the ASIC-to-sensor interface with "WR_S" command. When this command is sent to the ASIC, the address and data bits are transferred to the sensor. In this example, it can be seen that these bits are sent to the

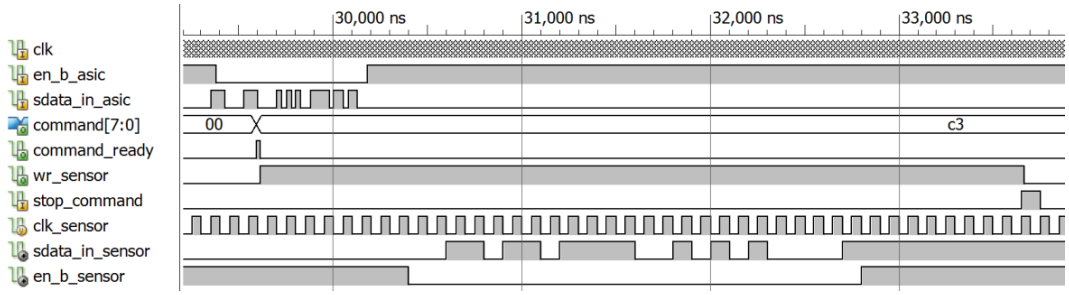

Figure 3.6: Simulation results of the ASIC-to-sensor interface with "WR_S" command.

sensor by driving the "sdata_in_sensor" and active-low "en_b_sensor" pins.

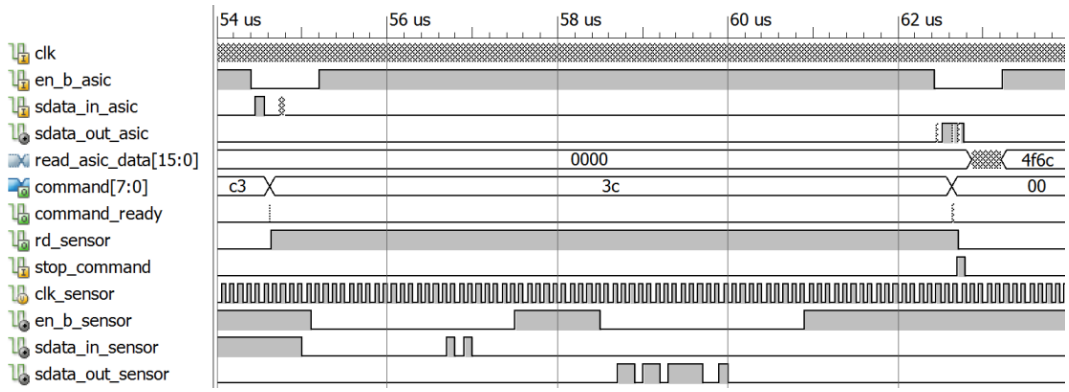

Figure 3.7: Simulation results of the ASIC-to-sensor interface with "RD_S" command.

Figure 3.7 shows the simulation results of the ASIC-to-sensor interface with "RD_S" command. The image sensor memory read operation is similar to the ASIC memory read operation. That means in order to gather data from sensor, two communication cycles are needed. Therefore, this operation sends related address in the first cycle and samples data at the next cycle. In the first "en_b_sensor" cycle, the address of the memory location that will be read is sent. In the second one, the content of the read sensor memory location is sampled from the "sdata_out_sensor" pin. This sampled data is then sent to the host via the "sdata_out_asic" pin.

Figure 3.8 shows the simulation results of the ASIC-to-sensor interface with "UPD_S" command. The image sensor memory update operation is simply the multiple "WR_S" operations with automatic address incremental. The data of the write operations are read from the mirror sensor memory which is located in the upper (higher address)

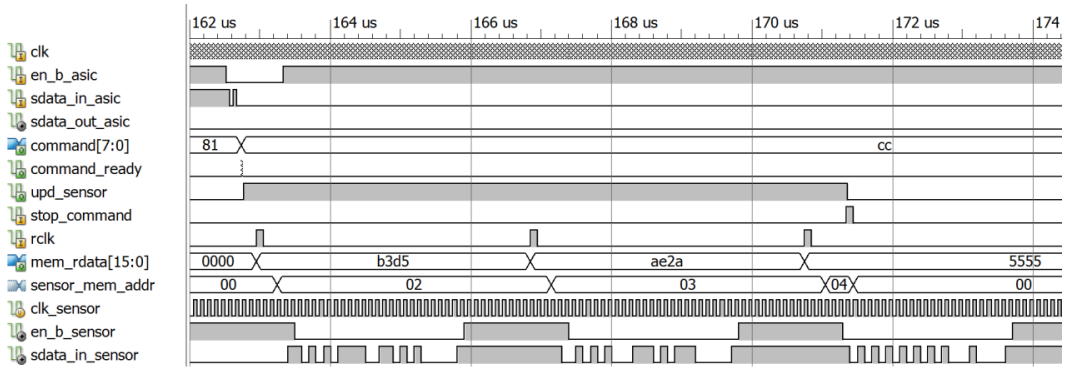

Figure 3.8: Simulation results of the ASIC-to-sensor interface with "UPD_S" command.

part of the ASIC memory. In the example, the sensor memory addresses from "0x02" to "0x04" are updated with data "0xB3D5", "0xAE2A" and "0x5555", respectively. The specialty of this command is that the user can program the whole sensor memory with just one command, which is easy to handle.

3.1.1 Instruction Controller

The instruction controller is simulated with the instructions that are given in the Table 2.4. The simulation setup for this block is similar to the communication tests setup. However in order to get the frame time of the sensor, the timing generator block is also added. Figure 3.9 shows the simulation setup for the instruction controller.

In the instruction controller tests, the instruction memory is programmed such that the sensor runs periodically with 2 different frame settings. This is an example usage of this block and Figure 3.10 shows the instruction memory contents for the instruction controller simulations. As it can be seen from the figure, the sensor is wanted to be programmed between consecutive frames with the help of instruction controller. Such usage is desired by the user, if the sensor needs to be run with different settings in successive frames.

Simulation for the instruction controller is run with the mentioned settings and Figures 3.11 and 3.12 shows the simulation results for the instruction controller. Note

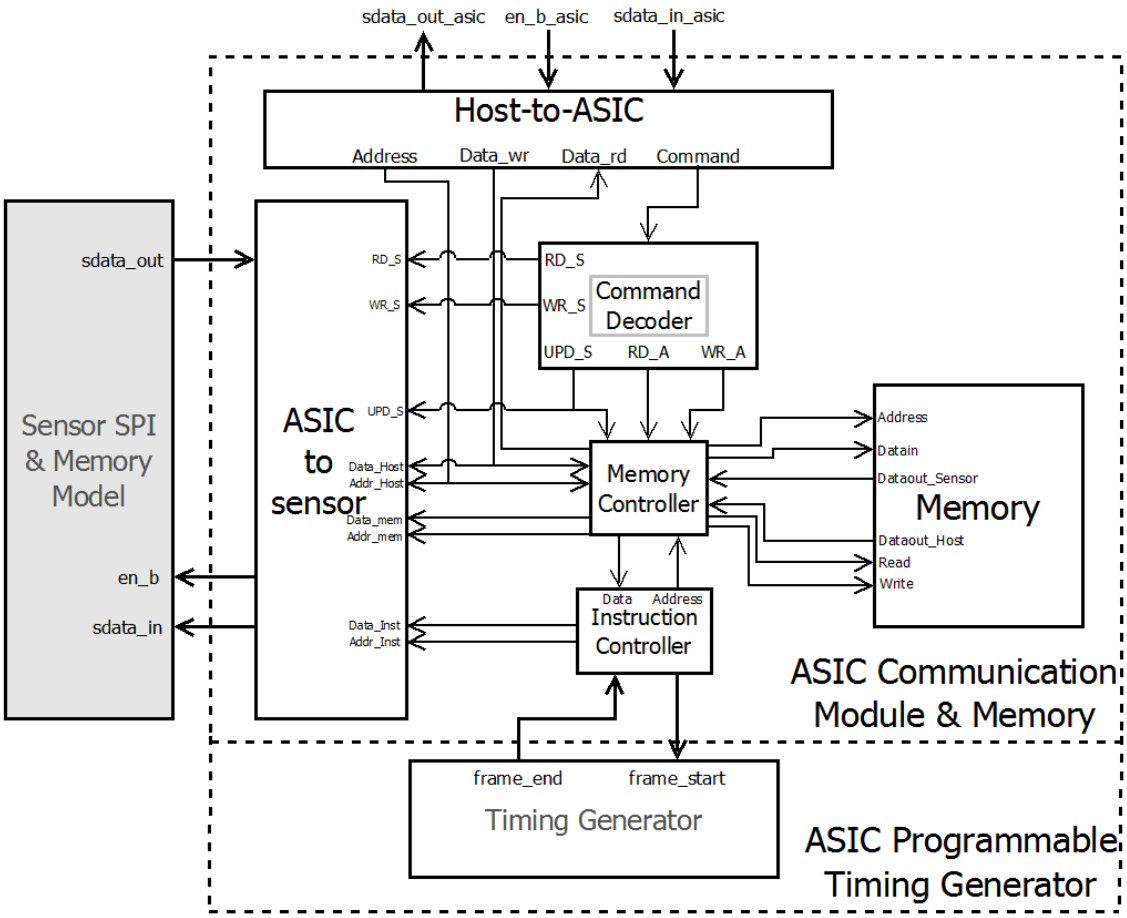

Figure 3.9: Simulation setup block diagram for the instruction controller.

	15 — 8	7 — 0	
0	0xC3	0xAA	Write to Sensor Address: 0xAA Data: 0xCCCC
1	0xCCCC		
2	0x1F	N/A	Run 1 Frame
3	0xC3	0xAA	Write to Sensor Address: 0xAA Data: 0x3333
4	0x3333		
5	0x1F	N/A	Run 1 Frame
6	0xFF	0x00	Go to Address 0

Figure 3.10: Instruction memory contents for the instruction controller simulations.

that in order to the simulations be understandable, the frame time is chosen as a very small time which lies between "frame_start" and "frame_end" signals.

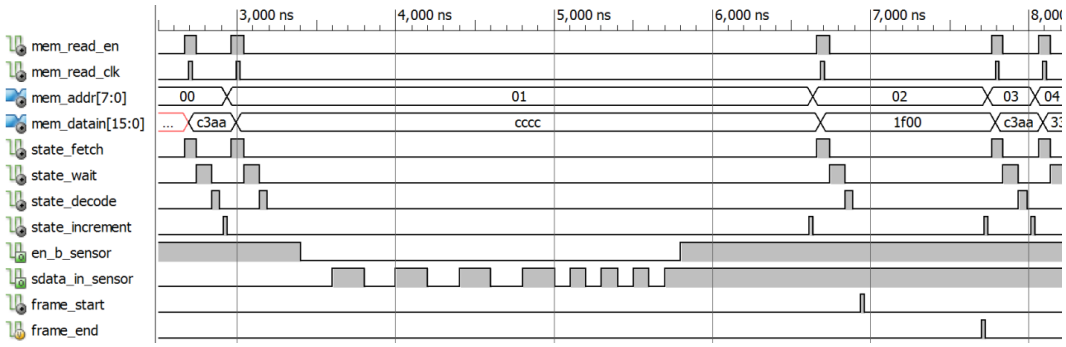

Figure 3.11: Simulation results for the instruction controller, part 1.

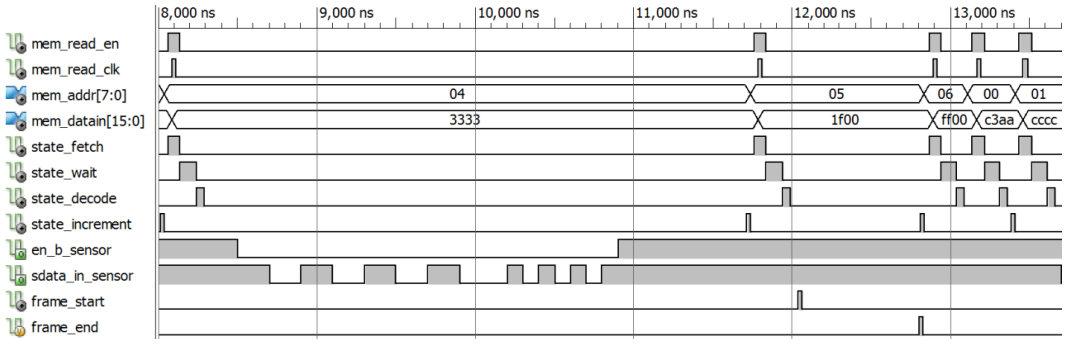

Figure 3.12: Simulation results for the instruction controller, part 2.

In the simulation, the "mem_addr[7:0]" shows the instruction memory address which is controlled by an address counter. This address starts from 0 and increments as the instructions are executed one by one. When address is 6, "Go to address 0" instruction is executed and counter is reset. This cycle is repeated as long as the instruction controller is enabled.

"state_fetch" signal shows the fetch state which means reading the next instruction from memory. In this state, the content of the memory location that is pointed by the address counter is read. After a small "wait" state, the read instruction is decoded. According to the decoded command, the execution is performed and the state machine continues with the next address.

The sensor write command triggers the sensor serial communication ports. When this

instruction is executed, the address and data that are fetched from memory are sent to the sensor via "sdata_in_sensor" and "en_b_sensor" pins. The content of these signals can be verified with the memory map which is given in the Figure 3.10.

3.2 Data Path

The data path of the ASIC forms a bridge between the digitized sensor data and host device. The video data pass through the sub-modules of this path and reach the output of the ASIC. As explained in the previous chapters, there are four channel included in this path and each channel consists of a small FIFO, an arithmetical unit and an 8b/10b encoder.

For simulations of this block, the data path is placed after the video data provider module which acts as image sensor. Figure 3.13 shows the simulation setup block diagram for the data path.

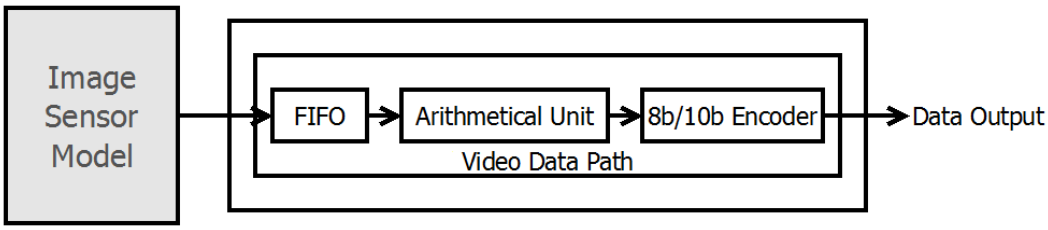

Figure 3.13: Simulation setup block diagram for the data path.

3.2.1 Arithmetical Processing Unit

The arithmetical unit is simulated with different input configurations. All the functions that are explained before are tested sequentially. Figure 3.14 shows the simulation results of the arithmetical processing unit with different input configurations.

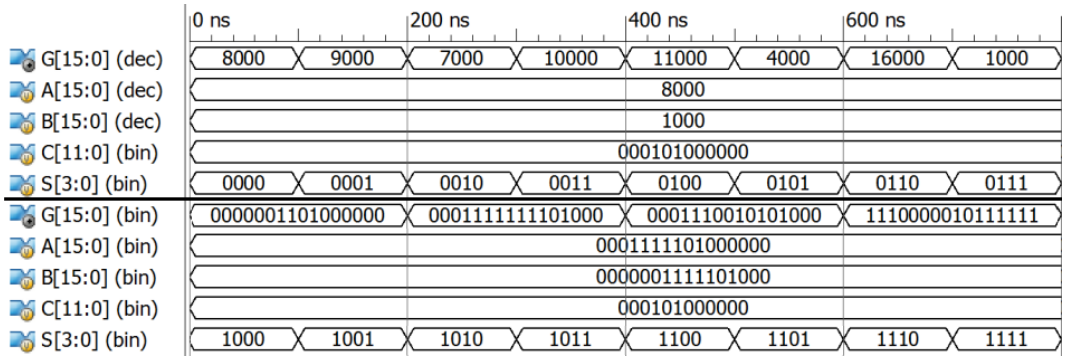

Figure 3.14: Simulation results of the arithmetical processing unit with different input configurations.

3.2.2 8B/10B Encoder

This part deals with the simulation results of the encoder part. First, the 3B/4B and 5B/6B encoding sub-blocks functional simulation results are given. Then, the whole 8B/10B encoder block is verified using an 8B/10B decoder connected together.

Figures 3.15, 3.16 and 3.17 give the simulation results for this block for different input configurations.

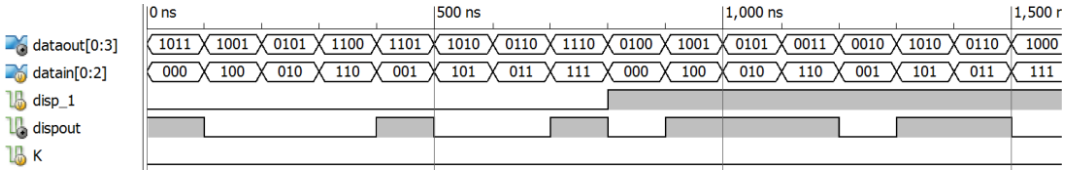

Figure 3.15: 3B/4B encoder simulation results, K=0, e=0, i=0

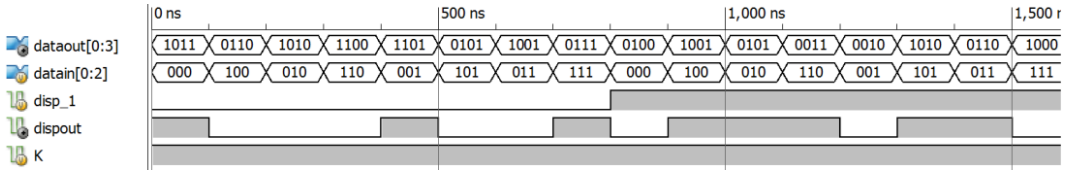

Figure 3.16: 3B/4B encoder simulation results, K=1, e=0, i=0

Besides the 3B/4B part of the encoder, the 5B/6B part is also simulated in a similar manner. Figure 3.18 shows a sample simulation for this part.

After the verification of smaller parts, both encoder modules are connected together

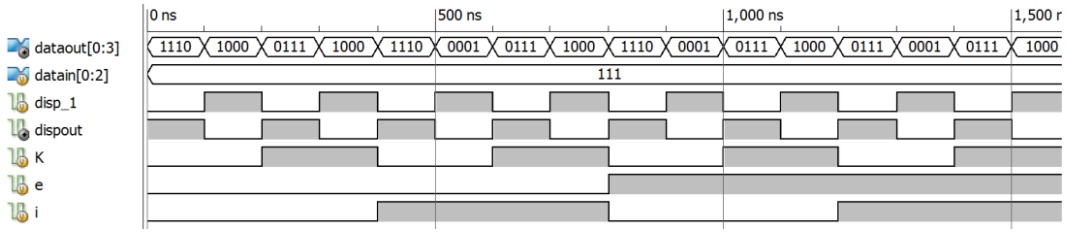

Figure 3.17: 3B/4B encoder simulation results, Data=111

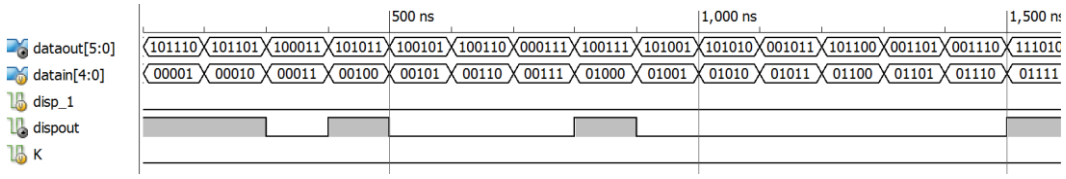

Figure 3.18: 5B/6B encoder simulation results, K=0

to form an 8B/10B encoder. Figure 3.19 shows a sample simulation results of the 8B/10B encoder.

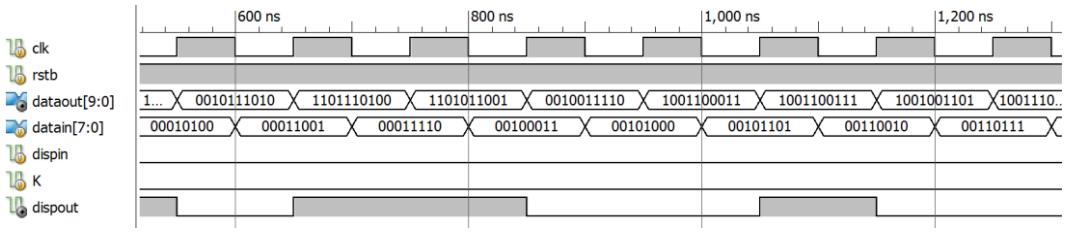

Figure 3.19: Sample simulation results of the 8B/10B encoder, K=0

The final simulation on this block is performed with a commercially available Xilinx 8B/10B decoder IP. The two blocks are connected together to verify the designed 8B/10B encoder. With these simulations the functional verification of this block is performed.

3.2.3 Data Path - Top Level

This part gives simulation results of whole data path, with the setup explained in Figure 3.13. As video data input, a counter that counts in data valid window is used. Figure 3.20 shows the simulation results of the imaging sensor model.

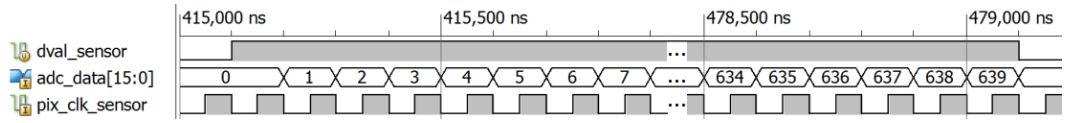

Figure 3.20: Simulation results of the imaging sensor model.

In the top level data path simulation, one scenario is selected as an example. In this scenario, the arithmetical unit is programmed to apply 1.5 gain and 1000 count offset. The encoder at the end of the path is disabled for better understanding of the output data. Figure 3.21 shows the simulation results of the data path with 1.5 gain and 1000 count offset applied on the incoming data.

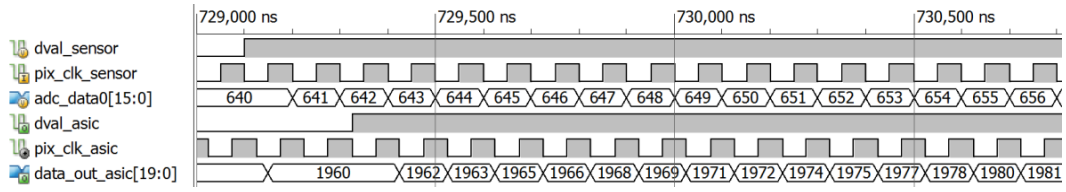

Figure 3.21: Simulation results of the data path with 1.5 gain and 1000 count offset applied on the incoming data.

3.3 Programmable Timing Generator

The final part that will be explained using simulations is the programmable timing generator block. This block can generate up to 8 independent signals whose pulse widths, repetitions and polarities are programmable. The programming options of these signals can be found in Chapter 2, under the "Programmable Timing Generator" section. For simulations, all of the 8 signals are programmed with different behaviors in order to check all the programmability options. Figure 3.22 shows the simulation results of the programmable timing generator block.

The bottom 8 signals in the figure represent the generated timings in this block. For the sake of simplicity, one frame is programmed as 32 lines and one line is programmed as 50 clock cycles.

As it can be seen, timings 0,1,3 and 4 are programmed as line based signals, which

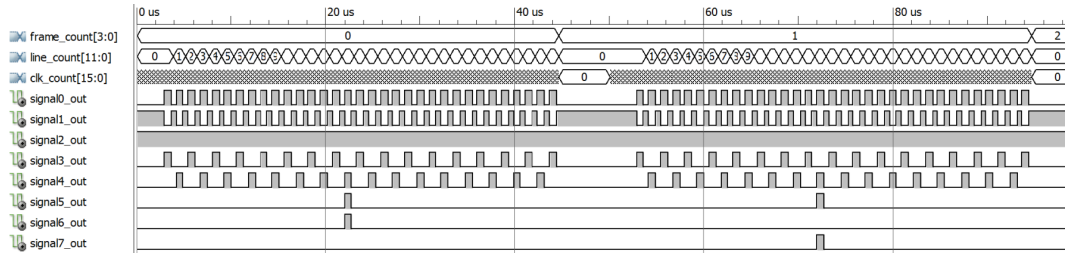

Figure 3.22: Simulation results of the programmable timing generator block.

means they are activated in every line or two. The "signal0_out" is programmed as an active high signal produced in every line. The "signal1_out", on the other hand, chosen as the same signal with different polarity which is active low. Timings 3 and 4 are even and odd versions of the timing 0.

One other property of this block is that it can also produce frame based signals. Timings 5,6 and 7 can be shown as examples. The "signal5_out" is programmed such that it is only active in particular line, which is line number 15 in this example. The signals 6 and 7 are even and odd versions of the timing 5. However in this case, even and odd calculated from the frame count instead of line count.

Finally the timing "signal2_out" is tied to a static value, which is logic 1 in this case. Similarly, it is also possible to tie the timings down to logic 0, which is not shown in this example due to the number of outputs.

In this chapter, the verification of the ASIC is explained in block level. Each block is simulated with different scenario that is easy to handle. Some simulations are conducted with multiple blocks tied together for better understanding. In some simulations, the model of the external electronics, such as sensor and host, are also included for correct verification. Since the verification of the blocks are completed, the implementation can be performed starting with FPGA tools. Following section explains the different implementation methods of the designed ASIC.

CHAPTER 4

DESIGN IMPLEMENTATION

In the previous chapters, the design of the ASIC with its features are discussed. The simulation results of the modules that are designed for the ASIC are also justified. In this chapter, the physical implementation of the digital ASIC will be explained. First, the implementation of the complete design on an FPGA device will be given. Then, the silicon implementation of the verified design will be explained.

4.1 FPGA Implementation

The simulation verifications of the designed systems are essential in order to be sure about the behavior of the circuit. However, it may not be enough for some designs. For example, if the circuit is designed to be a part of a system, it is better to test the design with the whole system. For such applications, FPGA implementation plays a crucial role. This process can also be called as emulation or prototyping, and it offers simpler verification while speeding up this verification process [22].

For the FPGA implementation, the system that can be seen in Figure 1.8 is used. This system includes a 640x512 image sensor coupled with MT6415CA readout circuit, a 2-channel 14-bit ADC card and a USB accessible FPGA card with Xilinx Spartan 6 FPGA. This system is chosen because it includes the actual sensor that is planned to be used along with the designed digital ASIC. Also, it is possible to test the essential ASIC features with this hardware.

4.1.1 Communication Tests

In order to communicate with the ASIC in an easy way, an ASIC test software is developed. With this software, the functionality tests, including communication, can be handled in an efficient way. Therefore, the test results of the ASIC is captured from this software.

The communication tests are performed to verify the simulations of the ASIC communication module and the memory. The communication commands are tested one-by-one, started from ASIC memory commands.

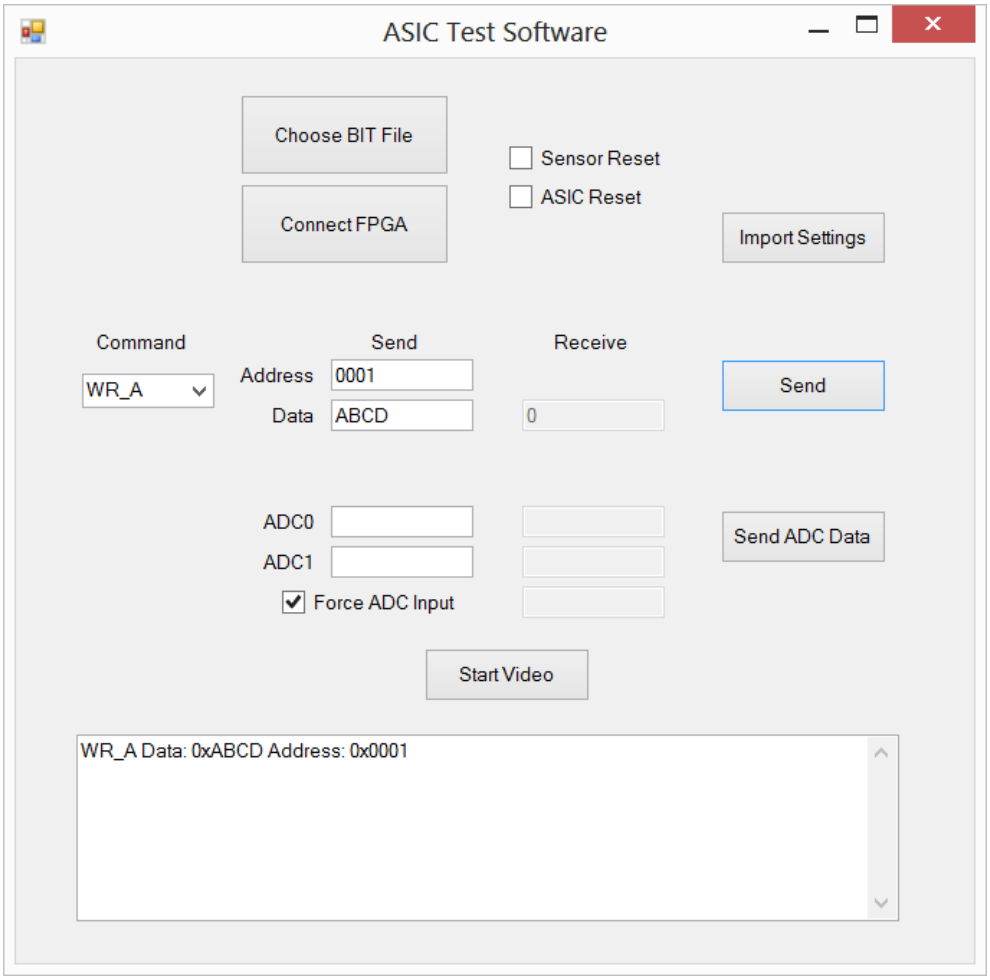

Figure 4.1: Screen capture of the "WR_A" command test. For tests, the location with the address "0x01" is programmed with the data "0xABCD".

The first command is "WR_A", which corresponds to "Write to ASIC memory" function. Figure 4.1 shows the screen capture of the "WR_A" command test. For tests,

the location with the address "0x01" is programmed with the data "0xABCD". As it can be seen from the log textbox, which is located at the bottom of the GUI, the "WR_A" command is executed.

In order to verify the "WR_A" command test, the programmed memory location should be read back. This brings the "RD_A" command test, which stands for the "Read from the ASIC memory" function. For this test, the pre-programmed location, whose address is "0x01", is read and the result is printed to the textbox under "Receive" part. Figure 4.2 shows the screen capture of the "RD_A" command test with a received data of "0xABCD" from the address "0x01". The verification can be made from the log window by observing the sent and the received data are identical.

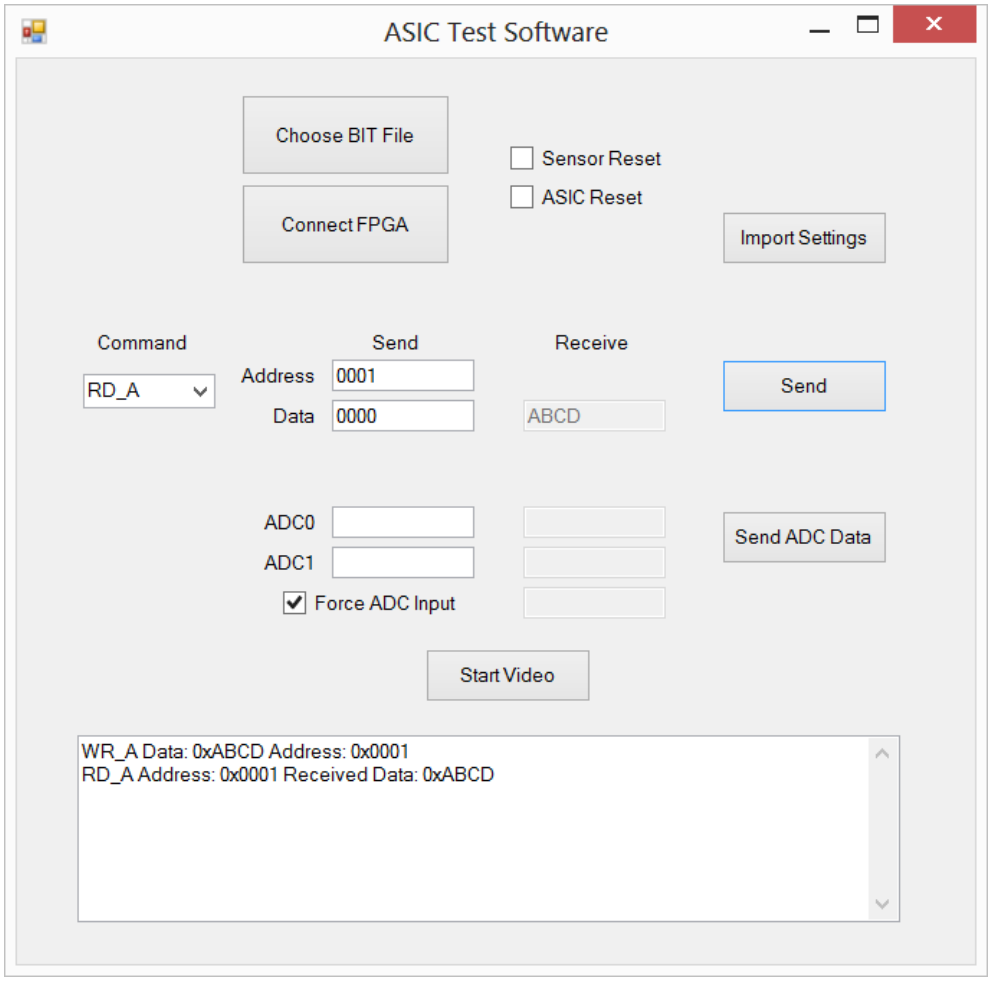

Figure 4.2: Screen capture of the "RD_A" command test with a received data of "0xABCD" from the address "0x01".

After the ASIC memory read and write tests, the sensor memory operations should

be verified. For this verification, the randomly selected sensor memory location is read to see the default content. Figure 4.3 shows the screen capture of the "RD_S" command test, part 1. The "Address" textbox shows the current read address, which is "0x04". The result is printed to the "Receive" textbox, which is "0x1249".

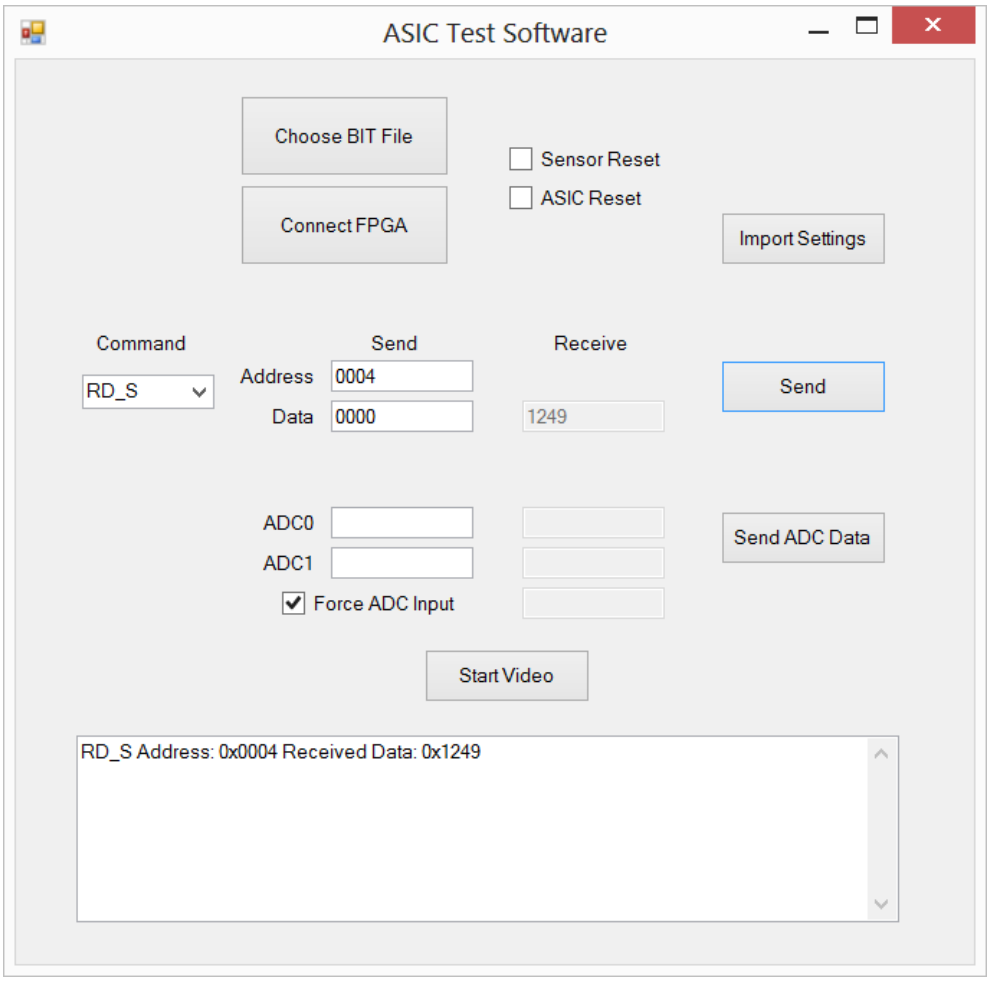

Figure 4.3: Screen capture of the "RD_S" command test, part 1. The "Address" textbox shows the current read address, which is "0x04". The result is printed to the "Receive" textbox, which is "0x1249".

The next test is the verification of the "Write to sensor memory" command, which is abbreviated as "WR_S". For the test, the previously read sensor memory location is programmed with a random value. Figure 4.4 shows the screen capture of the "WR_S" command test. The address is assigned as "0x04" and the data is given as "0xF0F0". The log window shows the sequence of these events.

In order to check the sensor memory write function, the written address should read

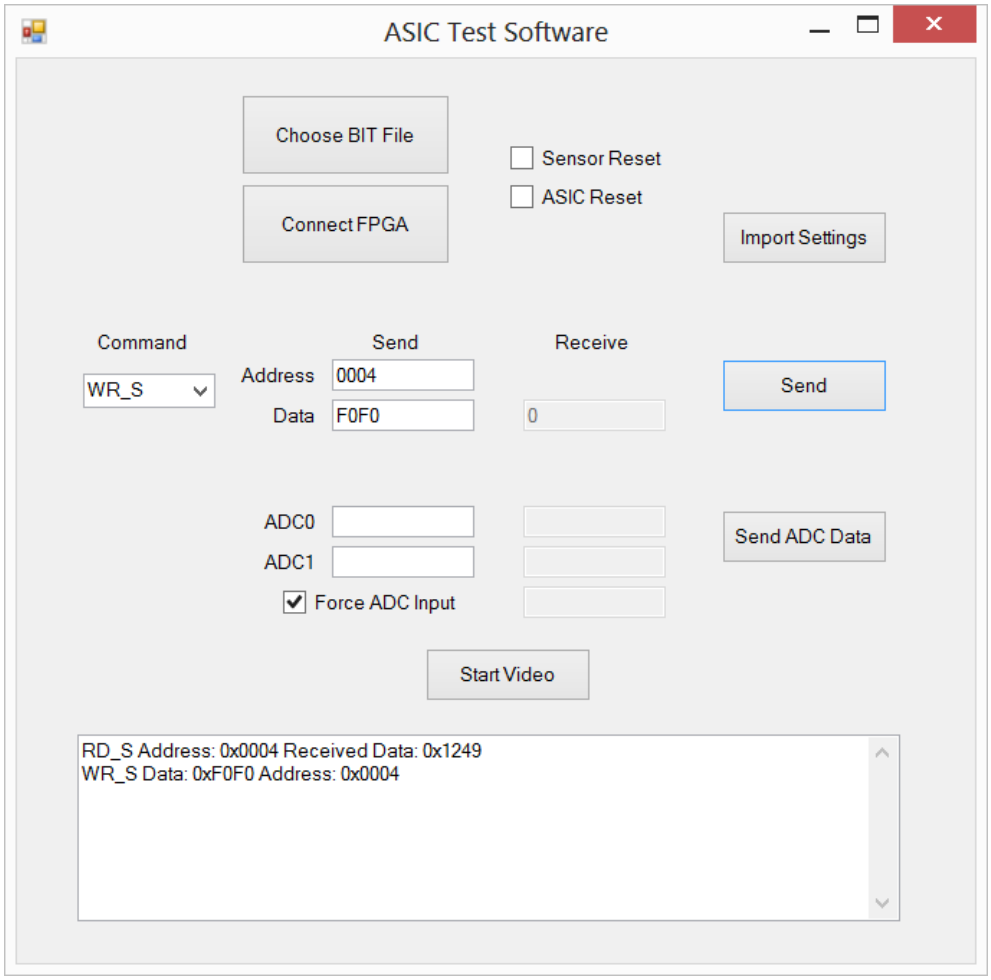

Figure 4.4: Screen capture of the "WR_S" command test. The address is assigned as "0x04" and the data is given as "0xF0F0".

back. Figure 4.5 shows the screen capture of the "RD_S" command test, part 2. The "Address" textbox shows the current read address, which is "0x04". The result is printed to the "Receive" textbox, which is "0xF0F0".

The final communication command that should be tested is the "Update sensor memory" command, which is "UPD_S". For this command, the related ASIC memory location that includes the sensor memory update interval is programmed first. Then, the corresponding sensor mirror memory, which resides in the higher half of the ASIC memory, is programmed. After those operations, the "UPD_S" command is executed. After this execution, the updated sensor memory locations are read back and compared with the corresponding sensor mirror memory. Note that since it is difficult to show each step separately, the results are not shown with a screen capture. However,

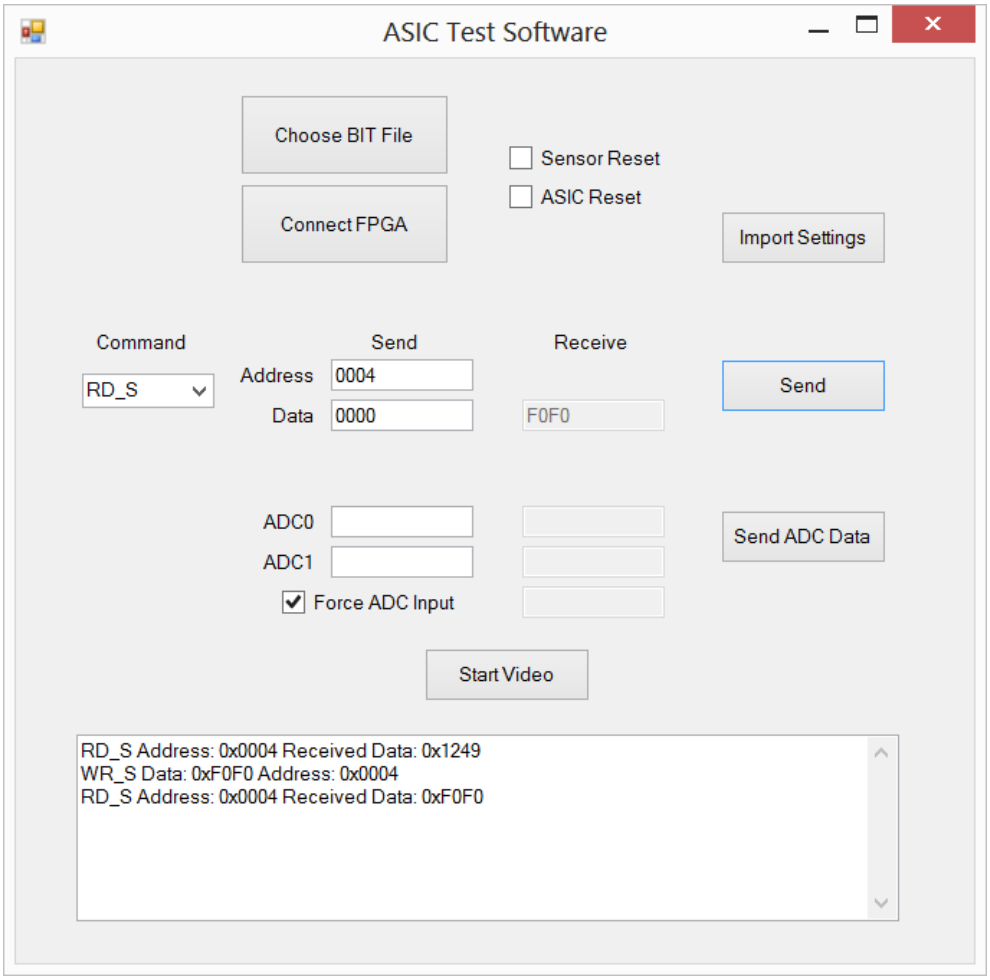

Figure 4.5: Screen capture of the "RD_S" command test, part 2. The "Address" textbox shows the current read address, which is "0x04". The result is printed to the "Receive" textbox, which is "0xF0F0".

this operation can be thought as the multiple occurrence of the "WR_S" event.

4.1.1.1 Instruction Controller Tests

The tests for the instruction controller are held on the same camera setup. For this test, the commands that are explained in the Table 2.4 are used. As the test vector, the instruction memory is programmed similarly to the simulation setup input used in the Chapter 3.

It can be remembered that the simulations are performed with different sensor settings take place in consecutive frames. This method is used when multiple different

frames of same scene are necessary. These differences between frames can be expressed in many way; like wavelength, focus, exposure etc. Those multiple frames can be used for "image fusion", which means combining multiple images by extracting meaningful data from each of the single images[23]. One common method of this can be stated as the multiple exposure imaging. In this method, it is possible to obtain an image from differently exposed images, leaving under-exposed and over-exposed areas behind. This is used to increase dynamic range and it forms the basics of High Dynamic Range (HDR) imaging[24]. For the instruction controller tests that in the ASIC, the dual exposure in consecutive frames is implemented.

For the dual exposure test, the instruction memory is programmed according to the two different exposure levels. Figure 4.6 shows the instruction memory map of the dual exposure programming event. As exposure times, 5ms and 20ms are selected with the current test conditions.

Program Sensor for 5ms Exposure
Run Sensor for 1 Frame Time
Program Sensor for 20ms Exposure
Run Sensor for 1 Frame Time
Repeat this process

Figure 4.6: Instruction memory map of the dual exposure programming event.

In addition to the instruction memory, the programmable timing generator block is also programmed according to the sensor needs. One of these timings is used to trigger the frame synchronization input of the sensor. Also the arithmetical unit is programmed for the normalization of the image, which will be explained in the data path tests.

The whole camera setup is used with the mentioned settings. The video output is collected with the developed ASIC test software. Figure 4.7 shows the video screen capture with two different exposure settings. The upper picture represents the image with 20 ms exposure time and the bottom picture represents the image with 5 ms exposure time. Note that the software shows the upper 8 bit part of the incoming 14

bit ADC data. Any other processing or manipulation techniques are not performed on the raw image.

Figure 4.7: Video screen capture with two different exposure settings. The upper picture represents the image with 20 ms exposure time and the bottom picture represents the image with 5 ms exposure time.

The two differently exposed images includes different data. The under-exposed parts in less exposed image can be seen clearly in the other. On the contrary, the saturated parts in more exposed image can be seen in the other. The two images can be combined in many ways. The simplest averaging is performed on the image using the software to see the effects of double exposure. Figure 4.8 shows the video screen capture with two different exposure settings and average of the two images. The bottom picture is constructed from the top two images which are 5 ms exposed image and 20 ms exposed image, respectively.

Figure 4.8: Video screen capture with two different exposure settings and average of the two images. The bottom picture is constructed from the top two images which are 5 ms exposed image and 20 ms exposed image, respectively.

4.1.2 Data Path Tests

In order to verify the data path implementation, the previously explained camera setup is used. In this test, the communication tests are also included since it is necessary to program the ASIC and the sensor, properly. The main reason to perform this test is to verify the data acquisition of the ASIC. The sensor is programmed to give two channel video output to the two channel 14-bit ADC card. The ADC outputs are given as inputs to the two of the ASIC input channels. The output is observed at the output channels of the ASIC.

There are some other programming options for the data path also. The arithmetical unit and the encoder options are also important to get true values from the outputs. The encoder part is held disabled to obtain meaningful data at the video outputs. With the different settings of the arithmetical unit, multiple tests are conducted.

For the first test, the arithmetical unit is programmed to give static values at the output. This test is performed to verify the data path after the arithmetical unit. In order to see the difference between the two channels, they are programmed as distinct values. Figure 4.9 shows the video screen capture of the event for which the data is supplied statically from the arithmetical unit. For this test, channel 0 and channel 1 are programmed with the minimum and maximum count values, respectively.

The second test is conducted with the similar thought, but different static data providers. This time, the arithmetical unit programmed as transparent, i.e. the input is transferred to the output without any change in the value. However, the data input channels of the ASIC are connected to static values instead of the sensor video outputs. The values of the static inputs are programmable through the developed software. Figure 4.10 shows the video screen capture of the event for which the data is forced statically from the ASIC data inputs. For this test, channel 0 and channel 1 are forced to the minimum and maximum count values, respectively. Note that while the previous test verifies the data path after the arithmetical unit, this test ensures the functions of the whole data path in terms of inputs and outputs.

The third test about the verification of the data path is the acquisition of the sensor video outputs. In order to follow the verification process easily, the setup mentioned

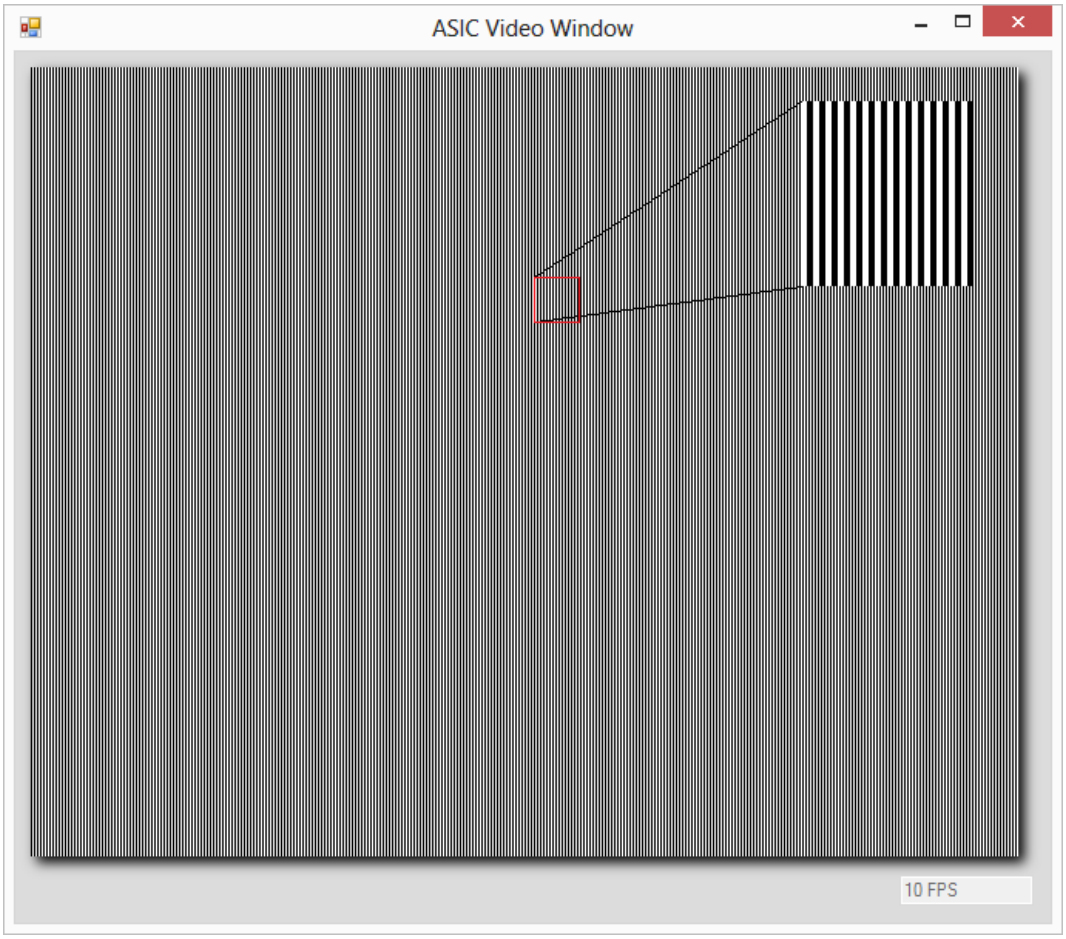

Figure 4.9: Video screen capture of the event for which the data is supplied statically from the arithmetical unit. For this test, channel 0 and channel 1 are programmed with the minimum and maximum count values, respectively.

in the second test is edited and the ASIC video data inputs are connected to the sensor video outputs. The data path is held in the transparent settings to see the raw video outputs at the end of the data path. This raw video data is collected with the developed software and displayed on the host computer. Figure 4.11 shows the video screen capture of the event for which the raw data is collected from the sensor.

The results of the third test shows that it is hard to see a proper image from the raw video data due to the small signal levels. For example for an 14-bit system, looking to the full scale counts from 0 to 16383 when the image information lies between 8000 and 12000 may not be a good idea. To overcome this problem, the image should be normalized or the histogram of the image should be equalized[25]. The forth test is about this issue and it tries to verify the arithmetical unit by implementing a linear

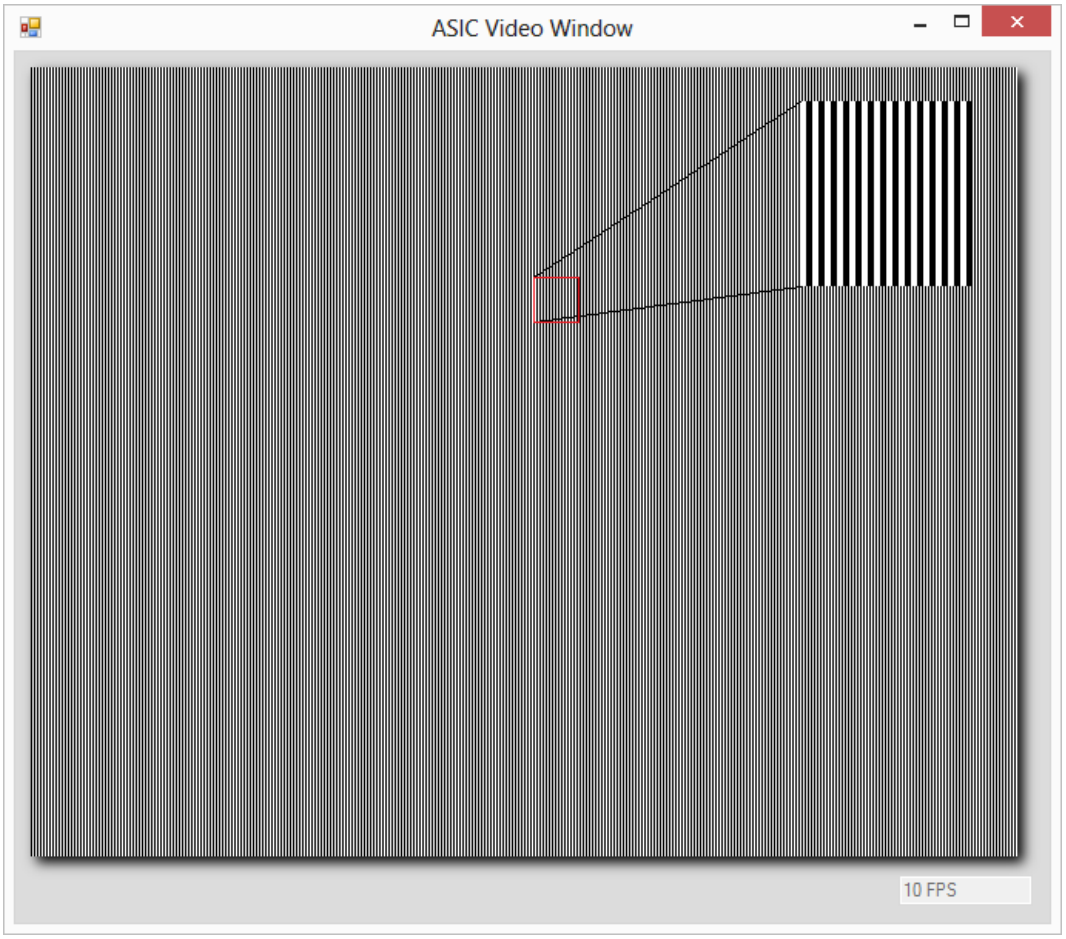

Figure 4.10: Video screen capture of the event for which the data is forced statically from the ASIC data inputs. For this test, channel 0 and channel 1 are forced to the minimum and maximum count values, respectively.

histogram equalization technique.

The range of the pixel values in an image can be expressed as,

$$N_{Range} = N_{Max} - N_{Min} + 1 \tag{4.1}$$

where N_{Max} is the maximum pixel value and N_{Min} is the minimum pixel value. On the other hand in the case of small video signal levels, the meaningful range of the pixel values in an image can be expressed as,

$$N_{MeanRange} = N_{MeanMax} - N_{MeanMin} + 1 \tag{4.2}$$

where $N_{MeanMax}$ is the maximum meaningful pixel value and $N_{MeanMin}$ is the minimum meaningful pixel value. In other words, the $N_{MeanRange}$ can be expressed as the

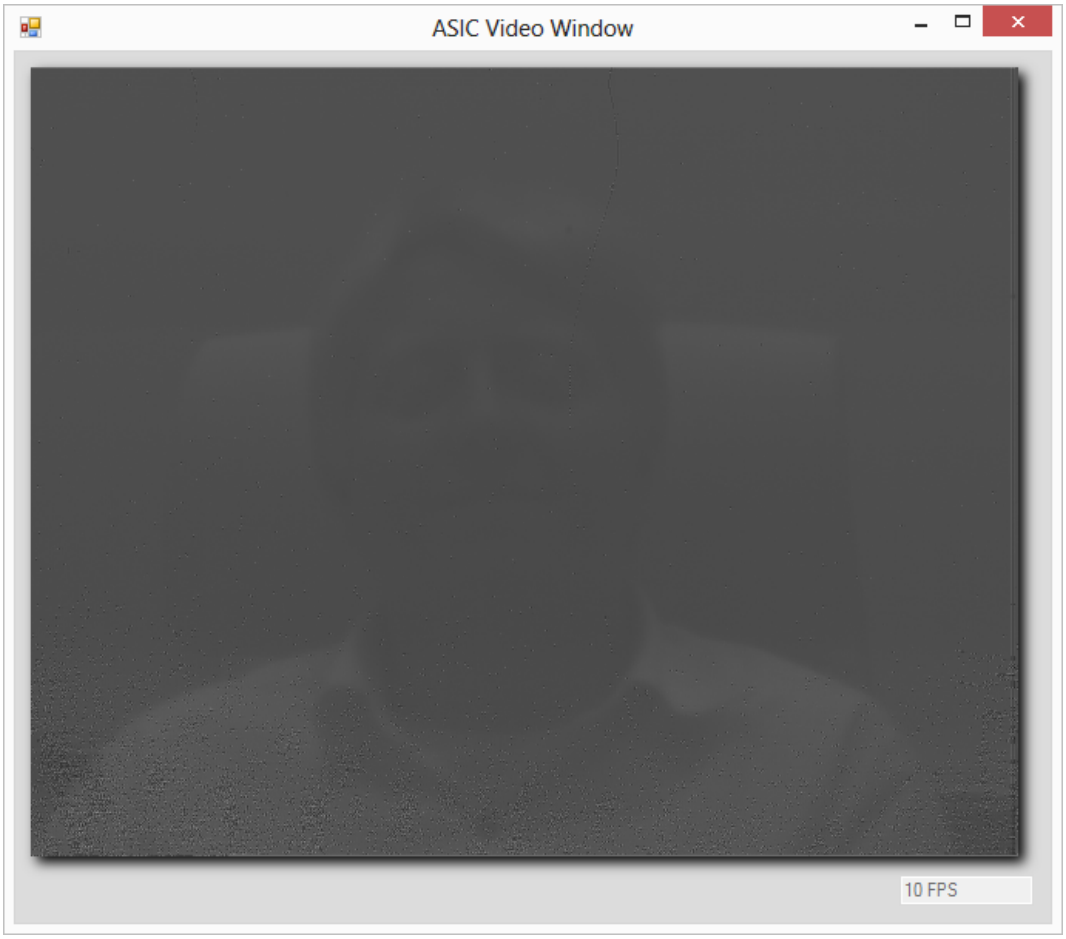

Figure 4.11: Video screen capture of the event for which the raw data is collected from the sensor.

range where the normalization is desired. With these variables are in hand, the new value of a given pixel can be expressed as,

$$N = (N_{Current} - N_{MeanMin}) * (N_{Range}/N_{MeanRange}) + N_{Min} \tag{4.3}$$

where N is the normalized pixel value and $N_{Current}$ is the current pixel value. For 14-bit pixel values, since N_{Max} and N_{Min} is equal to 16383 and 0, respectively; Equation 4.3 becomes,

$$N = (N_{Current} - N_{MeanMin}) * (16384/N_{MeanRange}) \tag{4.4}$$

Equation 4.4 shows that, with the proper selection of $N_{MeanRange}$ and $N_{MeanMin}$ values, the normalized pixel value, N, can be calculated with the "A*C+B" operation. It can be remembered that this operation is in the list of the supported arithmetical operations of the ASIC arithmetical unit. Therefore with the proper programming,

normalization can be applied on ASIC. Figure 4.12 shows the video screen capture of the event for which the normalization is implemented on the video data. For this test, $N_{MeanRange}$ is programmed as 4096 and $N_{MeanMin}$ is programmed as 9000.

Figure 4.12: Video screen capture of the event for which the normalization is implemented on the video data. For this test, $N_{MeanRange}$ is programmed as 4096 and $N_{MeanMin}$ is programmed as 9000.

4.1.3 FPGA Implementation Results

One of the important points in the design is the power consumption. In order to estimate the power consumption of the design, Xilinx XPower Analyzer tool is used along with the Xilinx ISE. The target architecture is selected as Spartan 6 since it is used in the implementation and tests. In these estimations, constraints are selected according to the different clock domains of 40MHz and 10 MHz. Table 4.1 shows the estimated power summary of the FPGA implementation.

Table4.1: Estimated power summary of the FPGA implementation.

	Dynamic	**Static**	**Total**
Supply Power (mW)	23.50	36.40	59.90

Besides the power consumption, the instance count of the synthesized design is an important factor for the area estimation. Table 4.2 shows the synthesis results of the FPGA implementation.

Table4.2: Synthesis results of the FPGA implementation.

Instance	**Count**
RAM (16x16 bit)	4
Adders/Subtractors	32
Adder Trees	4
Counters	2
Accumulators	8
Registers	4831
Comparators	37
Multiplexers	713
Xors	52

The synthesized implementation is then mapped to the FPGA. Table 4.3 shows the map results of the FPGA implementation.

Table4.3: Map results of the FPGA implementation.

Instance	**Count**
Slice Registers	5160
Slice LUTs	6914

The mapped results are then given to the place and route tool. On the final result, the timing analysis is performed and a maximum frequency of approximately 50MHz is

obtained. Note that this timing result is obtained with 40MHz timing constraints on the clock lines.

4.2 ASIC Implementation

The other method of implementing the designed ASIC is the physical implementation. As explained in the Chapter 1, there are multiple methods of this implementation. Here, the standard cell based implementation method is selected for the ASIC.

The main reason behind this selection is to be able to use automatic standard cell implementation tools, which is the way of converting RTL designs into physical layout. Since the ASIC design is completed with the Verilog HDL, such tools are necessary for ASIC implementation. There are multiple tools in the industry for such designs. In this work, the Cadence Encounter software is used. This software covers all the steps of implementation from RTL to layout[26].

In Chapter 1, the implementation steps of the ASIC are given in the Figure 1.7 as it can be remembered. Mentioned figure shows the implementation steps roughly. Figure 4.13 shows the detailed ASIC design flow.

In the ASIC implementation method, the steps that are given in the Figure 4.13 are followed. For the design and synthesis, a 90nm standard cell library is used with a supply voltage of 0.9V. The timing requirements are given as the same as in the simulations and FPGA implementation. In this way, the physical implementation of the proposed design is completed.

As it can be seen, the implementation starts with the RTL design entry, which is performed using Verilog HDL in this design. After this process, the synthesis tool converts the design into a netlist which is formed by standard cells. Then, this netlist is mapped to an initial layout placement. Figure 4.14 shows the initial layout placement of the physically implemented digital ASIC core with a square floor plan.

The placement operation is followed by the clock tree placement operation. The connections between the placed cells are made in the routing stage and the final design is verified in terms of timing, design rules and equivalence with the netlist. Note that

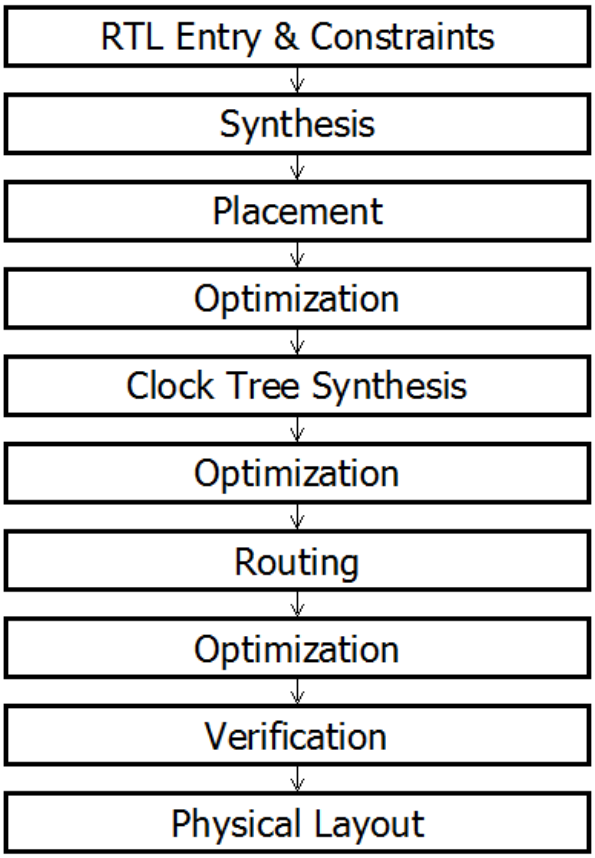

Figure 4.13: Detailed ASIC design flow.

between the important steps, one or more optimization stages are added to obtain a better final product.

Figure 4.15 shows the top level layout of the physically implemented digital ASIC core with a square floor plan. The layout measures approximately 550um both in height and width.

4.2.1 ASIC Implementation Results

For the power estimation of the ASIC implementation, the Power Analysis tool of the Cadence Encounter software is used. In the estimations, different switching ratios are applied with the dominant frequency of 40MHz. Table 4.4 shows the estimated power summary of the ASIC implementation for different switching modes. Note that the frequency is selected as 40MHz although some parts of the design (e.g. data path)

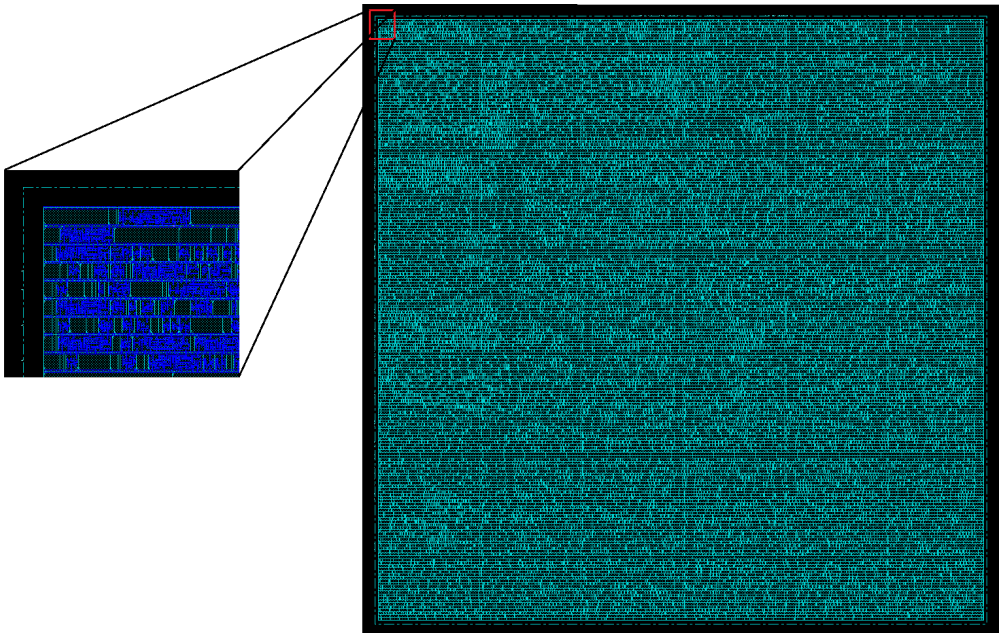

Figure 4.14: Initial layout placement of the physically implemented digital ASIC core with a square floor plan.

run at 10MHz. These results can be inferred as the worst case dissipations obtained from the power analysis tools.

Table4.4: Estimated power summary of the ASIC implementation for different switching modes.

	Internal	**Switching**	**Leakage**	**Total**
Power @ 20% switching (mW)	2.265	0.93	1.084	4.279
Power @ 50% switching (mW)	5.263	2.21	1.084	8.557
Power @ 100% switching (mW)	10.03	4.213	1.084	15.32

In order to obtain some information on the area coverage of the ASIC implementation, instance counts should be examined. Table 4.5 shows the synthesis results of the ASIC implementation.

One of the most important points in the ASIC implementation process is the timing results. In this implementation, 40MHz timing constraints are applied to the clock lines. With these settings, approximately 50MHz maximum working frequency is obtained.

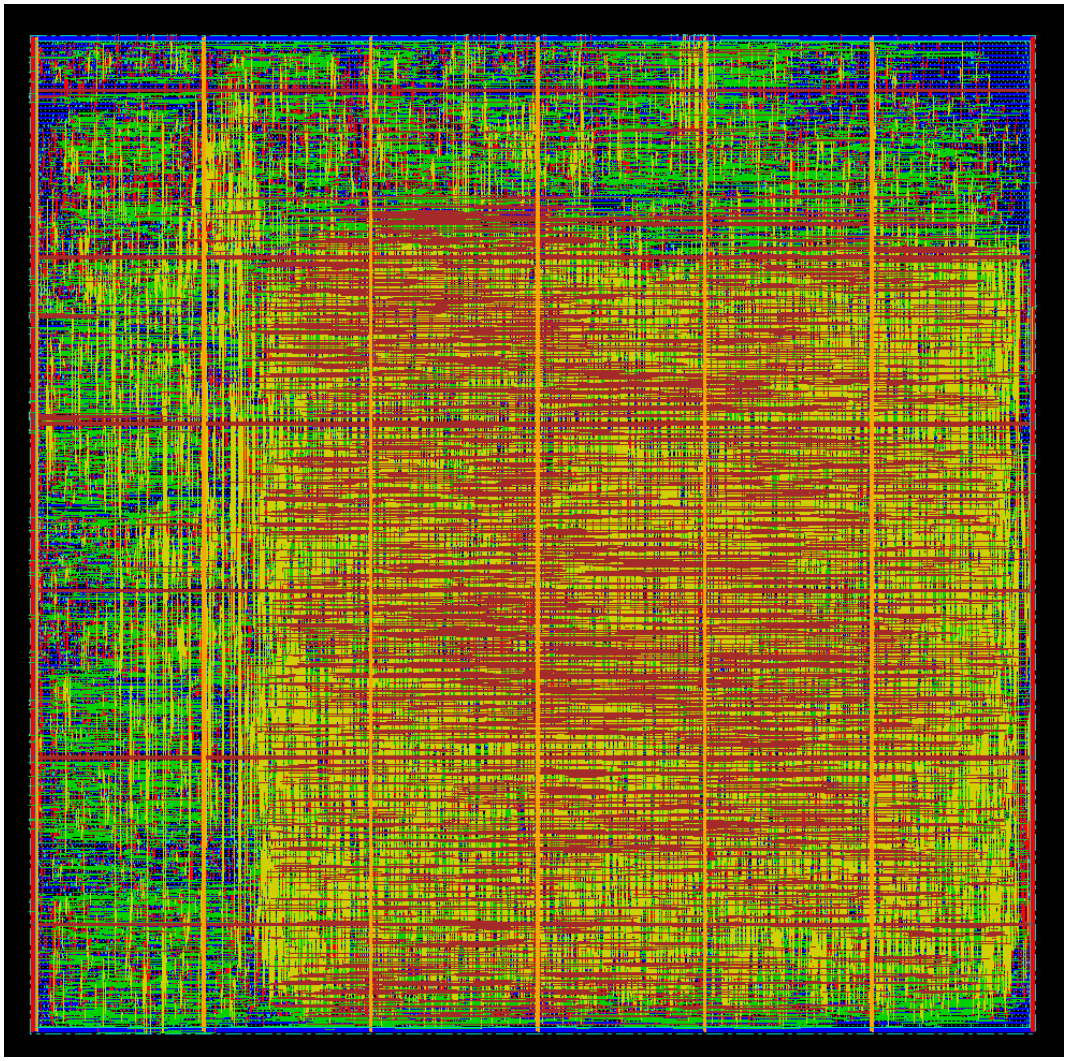

Figure 4.15: Top level layout of the physically implemented digital ASIC core with a square floor plan. The layout measures approximately 550um both in height and width in a 90nm CMOS process.

Table4.5: Synthesis results of the ASIC implementation.

Instance	**Count**
Sequential	5880
Inverter	627
Logic	11430

The obtained power estimations of the FPGA and ASIC designs can be compared. While comparing those results, it should be noted that the technology node of the ASIC implementation is 90nm while the node of the Xilinx Spartan 6 family is

45nm[27]. The reason to use a 90nm process in the ASIC implementation is to be as close as possible to the FPGA implementation in terms of technology node. If any other technology is desired, the design can be easily migrated to the other nodes since the used design flow is straightforward. With these specifications in mind, it can be seen that the ASIC implementation is advantageous over the FPGA design in terms of power.

In the implementation results, the final instance counts of both designs are examined. However, it is not possible to deduce the direct comparison results from these numbers. The reason behind this problem is the difference between the mapping libraries of the two implementation methods. While FPGA implementation uses slices and LUTs to map the given design, ASIC implementation method uses a pre-defined standard cell library. However if a comparison is desired, the similar numbers of sequential elements in both designs can be observed.

Finally, the timing analysis results of the two designs can be observed. With the similar timing constraints, up to 50MHz maximum frequency is obtained on the both designs. Note that on the ASIC implementation, similar timing performance is achieved with the FPGA implementation; with less power consumption.

In this section, the different implementation methods of the designed circuit is given with the tests and analysis. The given design parts in previous chapters are verified using sensor-in-the loop tests and the FPGA tested circuit is implemented using high-level synthesis tools. In the following chapter, the conclusion of the thesis will be explained.

CHAPTER 5

SUMMARY AND CONCLUSION

The thesis deals with the design of a digital ASIC that is used for for drive and data acquisition of image sensors. After the determination of the design specifications, the ASIC is designed using Verilog HDL. This design then simulated using Xilinx design and simulation tools. After the simulation verifications, the ASIC is tested on a Xilinx Spartan 6 FPGA board with the sensor-in-the-loop tests. The final design is converted to the physical layout using automated design tools. The achievements that are obtained while designing the ASIC can be summarized in the following way:

1) The design can handle image sensor communication and data acquisition while generating necessary periodical signals that are needed by the sensor. During those operations, it offers a wide range of programmability options as well.

2) The programming options of the ASIC can be controlled through a standard 3-wire SPI interface in slave mode. The programming is simplified with the help of the pre-defined communication commands. These communication commands are handled by the communication module which handles all the communication between ASIC-sensor and host-ASIC interfaces.

3) The ASIC includes 256x16 bit register based memory. This 256 word memory is used to save the sensor settings, ASIC configurations and some instructions. The sensor settings are saved in the upper half part of the memory and all the sensor memory is mapped to this part. The instruction memory saves sequential instructions that are used for periodical sensor operations. Finally, the ASIC configuration memory is used to store the necessary ASIC settings.

4) The instructions that are written in the instruction memory can be used for periodical sensor operations. With it own instruction set, the frame based periodicity of the sensor can be controlled through this memory. This part of the memory is controlled with the instruction controller module which includes an address counter and a simple state machine for this operation.

5) The sensor timings can be controlled with the programmable timing generator block of the ASIC. This block can generate up to 8 timing signals with a wide range of programmability options. Generation of line based, frame based, static, active low and active high signals are some of these programmability options. These settings are selected while considering general image sensor operations.

6) The data path of the ASIC can acquire the video data output of the sensor. Up to 4 channels of video are supported. Each channel consists of an arithmetical unit and an encoder. Mentioned parts bring some manipulation capabilities over the video data. The arithmetical unit resides in this path can apply some arithmetical operations on the data like addition, subtraction or multiplications as well as some logical operations like and, or, xor. The encoder, on the other hand, ensures an error-free communication if the output serialization is the case.

7) The whole ASIC is designed using Verilog HDL in a modular way. The design is simulated using Xilinx design tools. In the simulations, the ASIC is connected with the sensor device models as well.

8) After the simulations, the ASIC is tested on a complete image system. The system includes a MT6415CA image sensor, a Spartan 6 FPGA and a 2-channel 14-bit ADC card. The ASIC is placed in the FPGA and tested accordingly. With this way, the sensor-in-the-loop tests are conducted and the features of the ASIC are verified with the real hardware.

9) The verified ASIC is given to the synthesis tools to obtain physical layout of the design. For this operation, Cadence Encounter software is used with a 90nm standard cell library. With this way, the layout of the ASIC is obtained with the design automation tools. The final layout measures approximately 550um both in height and width.

10) Finally, the two implementation methods are compared according to the obtained power, area and timing results. From these comparisons, it is seen that the ASIC implementation is advantageous over the FPGA implementation in general.

Although the ASIC provides wide range of programmability options, some of the features can be improved. The future work for the ASIC can be expressed in the following way:

1) The arithmetical unit resides in the data path can be improved to obtain advanced operations. For example, the frame based operations like dark-image subtraction or filtering can be given as examples.

2) For frame based operations the device memory should be increased. For this reason, a custom designed memory can be used in the ASIC. Another solution can be the addition of the supports for external memory modules.

REFERENCES

[1] Programmable Logic Design, Quick Start Guide. Technical report, Xilinx, 2008. also available at http://www.xilinx.com/support/documentation/boards_and_kits/ug500.pdf [Online; accessed 04-Feb-2014].

[2] Selim Eminoglu, Murat Isikhan, Nusret Bayhan, M. Ali Gulden, O. Samet Incedere, S. Tuncer Soyer, Serhat Kocak, Gokhan S. Yilmaz, and Tayfun Akin. MT6415CA: a 640x512-15um CTIA ROIC for SWIR InGaAs detector arrays, 2013.

[3] Clifford E. Cummings. Simulation and Synthesis Techniques for Asynchronous FIFO Design. In *SNUG (Synopsys User Group Conference)*, San Jose, CA, 2002. also available at http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf [Online; accessed 04-Feb-2014].

[4] J. Goldie and S. Poniatowski. Lvds goes the distance? *SID Symposium Digest of Technical Papers*, 30(1):126–129, 1999.

[5] CCD Analog Front-End with Timing Generator for Digital Cameras. Technical report, Texas Instruments, 2013. also available at http://www.ti.com/lit/ds/symlink/vsp8133.pdf [Online; accessed 04-Feb-2014].

[6] E.A. Bender and S.G. Williamson. *A short course in discrete mathematics*. Dover books on mathematics. Dover Publications, 2005.

[7] 7 Series FPGAs Configurable Logic Block User Guide. Technical report, Xilinx, 2013. also available at http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf [Online; accessed 04-Feb-2014].

[8] W.J. Daily and A. Chang. The role of custom design in ASIC chips. In *Design Automation Conference, 2000. Proceedings 2000*, pages 643–647, 2000.

[9] D.G. Chinnery and K. Keutzer. Closing the power gap between ASIC and custom: an ASIC perspective. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 275–280, 2005.

[10] M.J.S. Smith. *Application specific integrated circuits*. VLSI systems series. Addison-Wesley, 1997.

[11] David G. Chinnery and Kurt Keutzer. *Closing the Power Gap between ASIC and Custom - Tools and Techniques for Low Power Design.* Springer, 2007.

[12] Ian Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203–215, 2007.

[13] Andreas Ehliar. Performance driven FPGA design with an ASIC perspective, Linköping University, Ph.D. dissertation, 2009.

[14] FLIR Lepton. also available at http://www.flir.com/cvs/cores/view/?id=62648 [Online; accessed 19-Mar-2014].

[15] SIDECAR™ ASIC. also available at http://www.teledyne-si.com/imaging/sidecar.html [Online; accessed 19-Mar-2014].

[16] Acadia® II Embedded Video Processors. also available at http://www.sri.com/engage/products-solutions/acadia-ii-embedded-video-processors [Online; accessed 19-Mar-2014].

[17] Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers. Technical report, 2000. also available at http://www.imagelabs.com/wp-content/uploads/2010/10/CameraLink5.pdf [Online; accessed 04-Feb-2014].

[18] M. Mano and C.R. Kime. *Logic and computer design fundamentals*. Prentice Hall, 3rd edition, 2004.

[19] A. X. Widmer and P. A. Franaszek. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. *IBM Journal of Research and Development*, 27(5):440–451, 1983.

[20] LVDS Owner's Manual, Including High-Speed CML and Signal Conditioning. Technical report, Texas Instruments, 2008. also available at http://www.ti.com/lit/ml/snla187/snla187.pdf [Online; accessed 04-Feb-2014].

[21] Xilinx Design Tools - ISE Design Suite 14. also available at http://www.xilinx.com/publications/matrix/Software_matrix.pdf [Online; accessed 24-Mar-2014].

[22] M. Courtoy. Emulation: prototyping without the hassles of fpga-to-asic conversion. In *WESCON/'95. Conference record. 'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*, pages 283–, 1995.

[23] P.J. Burt and R.J. Kolczynski. Enhanced image capture through fusion. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 173–182, May 1993.

[24] S. Mann and R.W. Picard. Being 'undigital' with digital cameras: Extending Dynamic Range by Combining Differently Exposed Pictures. Technical Report 323, M.I.T. Media Lab Perceptual Computing Section, 1994. also appears IS&T's 48th annual conference, Cambridge, Massachusetts, May 1995.

[25] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[26] Cadence Encounter Digital Implementation System: The fastest, most deterministic path to power, performance, and area realization. Technical report, Cadence, 2013. Data sheet, also available at http://www.cadence.com/rl/resources/datasheets/edi_system_ds.pdf [Online; accessed 04-Feb-2014].

[27] High-Volume Spartan-6 FPGAs: Performance and Power Leadership by Design. Technical report, Xilinx, 2011. also available at http://www.xilinx.com/support/documentation/white_papers/wp396_S6_HV_Perf_Power.pdf [Online; accessed 22-Mar-2014].