USB 3.0 Machine Vision Camera Hardware Design and FPGA Implementation

by

Qianqi Zhuang

B.A.Sc., China University of Mining and Technology, 1993

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2017

Approval

Name:	Qianqi Zhuang
Degree:	Master of Engineering
Title:	**USB 3.0 Machine Vision Camera Hardware Design and FPGA Implementation**
Examining Committee:	**Chair:** Michael Adachi Assistant Professor

Jie Liang
Senior Supervisor
Professor

Marinko Sarunic
Supervisor
Professor

Jianbing Wu
Supervisor
Pn.D., P.Eng.
CEO, AltumView Systems Inc.

Date Defended/Approved: May 31, 2017

Abstract

Machine vision camera is becoming more popular in the industry. USB 3.0 interface support 5Gbps transmission, and is a low-cost and fast way to transmit video signals from sensors to computers. However, most image sensors are incompatible with USB 3.0 protocol, and cannot directly connect to USB 3.0. In this report, we present the development of the hardware design of a USB 3.0 multi-purpose camera using the Cypress microprocessor. Our camera also has a FPGA module as an adaptive part to provide protocol translation, voltage level conversion, serial to parallel conversion, multi-channel data collection, and clock synthesis. We also discuss some important principles of system and schematic design, and emphasize a few critical PCB routing rules for assurance of PCB integrity. Some testing outcomes are provided to illustrate the hardware functionality and algorithm running results, which verify the success of the design and the performance of the camera.

Keywords: USB 3.0; Machine Vision Camera; FPGA; Image Processing.

Acknowledgements

I would like to express my gratitude towards my senior supervisor, Dr. Jie Liang, for his valuable guidance, motivation and support for this project. I am sincerely grateful to him for sharing his truthful and illuminating opinions on various aspects of this project.

I would also like to thank Dr. John Wu for showing me how to design the mechanical and optical parts of the camera. I also thank my colleagues Dr. Minghua Chen and Mr. Si Ke, who have given me tremendous helps and unselfish and constructive suggestions during the integration of hardware, firmware and software, especially when I met difficulties in fixing bugs.

I would also like to express my gratitude to Dr. Marinko Sarunic and Dr. Michael Adachi for serving on my committee and providing me valuable advices and feedback.

Table of Contents

List of Tables

List of Figures

List of Acronyms

A/D	Analog to Digital converter
CMOS	Complementary Metal-oxide-semiconductor
DDR	Double data rate
EAV	End Active Video
FPGA	field-programmable Gate Array
FIFO	First In, First Out
GE	Gigabit Ethernet
HSTL	High Speed Transceiver Logic
IP	Internet Protocol
LVCMOS	Low-voltage CMOS
LVDS	Low-voltage differential signaling
LVTTL	Low Voltage TTL
MAC	Media Access Control
MV	Machine Vision
MVC	Machine Vision Camera
PCB	Printed Circuit Board
PLL	Phase Locked Loop
POE	Power Over Ethernet
SAV	Start Active Video
SerDes	Serializer/Deserializer
SSO	Simultaneously Switching Output
SSTL	Stub Series Terminated Logic
TCP	Transmit Control Protocol
TTL	Transistor-transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
USB 3.0	Supper speed USB Interface, 5.0Gbps Bit Rate

Chapter 1.

USB 3.0 Multi-Purpose Camera Introduction

1.1. Overview of machine vision camera

Machine vision (MV) is the technology and methods used to provide imaging-based automatic inspection and analysis for applications such as automatic inspection, process control, and robot guidance. Machine vision encompasses many technologies, software and hardware products, integrated systems, actions, methods and expertise.

Currently the most popular machine vision cameras are USB 3.0 (or simply USB3) cameras and GigE cameras. The main difference between USB 3.0 camera and GigE camera is the interface between camera and computer. GigE camera uses Gigabit Ethernet interface to connect to computer, and the communication between them is based on TCP/IP protocol. USB 3.0 camera uses USB 3.0 interface to connect to computer, which follows the USB 3.0 protocol for communication.

Comparing to GigE camera, USB 3.0 camera has a few advantages, as listed below:

- USB 3.0 camera has much higher communication bandwidth than GigE camera.
- USB 3.0 camera connects to computer through USB 3.0 interface which can provide up to 5Gbps bandwidth, while GigE camera uses Gigabit Ethernet to transmit video or images to computer with 1Gbps bandwidth limitation.
- Since USB 3.0 camera can provide more bandwidth for communication, USB 3.0 camera can provide higher resolution and higher frame rate than GigE camera.
- USB 3.0 host interface can provide 5V, 0.9A power budget. Therefore, as a device to computer, USB 3.0 camera does not need external power supply, which simplifies design and assembly, making the entire system more compact.

On the other hand, the GigE camera also has some advantages:

- A single USB 3.0 cable can only be up to 6 feet long，whereas GigE camera can be 100m away from the computer through cat5e cable.
- GigE camera can also be powered by POE (Power Over Ethernet) through cat5e twisted cable.

In this project, we focus on the development of a USB 3.0 camera, and present the system structure, hardware design, FPGA programming for interface and protocol translation, and test results.

1.2. USB3 camera hardware architecture

Most image sensors are incompatible with USB 3.0 protocol, and cannot directly connect to USB 3.0. Therefore in this project, the Cypress EZ-USB FX3 SuperSpeed USB 3.0 peripheral controller is used as the bridge between the image sensor and the PC. Our USB3 camera system consists of five main parts plus power supply, as shown in Figure 1.1.

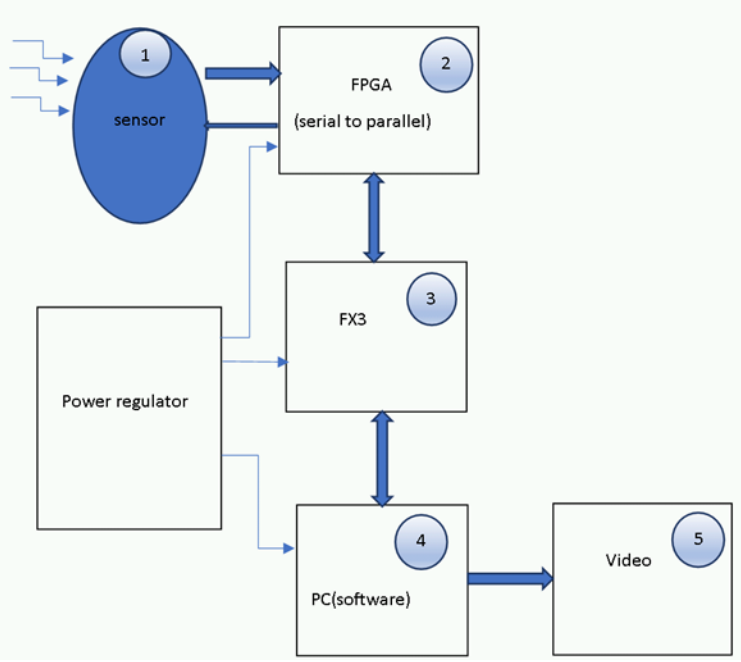


Figure 1.1. USB 3.0 Camera system block diagram.

Parts 1: Video sensor; In this part, video sensor captures pictures when incident light illuminates on the video or image sensor, which has been reflected from objects. Sensor will then send out the picture data pixel by pixel in raster order, where the pixels are read row by row until the entire frame of pixel is transmitted. The procedure can be controlled by the FPGA component (part 2).

Part2: FPGA; FPGA helps to generate the video clock signal, read data from sensor, convert it to proper parallel format, and then feed to the FX3 module (FX3).

Part3: FX3; This is the Cypress EZ-USB FX3 SuperSpeed USB 3.0 peripheral controller, which serves as the bridge between the image sensor and the PC. It picks up parallel video data stream from FPGA, schedules the data stream to USB buffer (endpoint), and sends the data to PC when the host requests data through USB protocol.

Part 4: PC software. This part is responsible for receiving video data from USB interface, reconstruct video or image according the control information, and perform other image and video processing. Another important function of this part is to convert the raw Bayer sampling format of the sensor to the full RGB format.

The hardware made up of two boards, one is sensor board which contains part 1, and the second board is main board which consist of part 2, part 3 and power regulator. The sensor board and the main board are connected through a 60 pins connector. More details of each of part will be discussed later.

1.3. USB 3.0 host/slave interface and corresponding firmware structure

1.3.1. USB 3.0 physical layer

In Figure 1.2, the USB 3.0 SuperSpeed cable employs a dual-simplex channel which makes bidirectional communication possible. SSTX+, SSTX- represent transmission differential pair; SSRX+, SSRX- represent receive differential pair. The SuperSpeed channel signal rate can reach 5Gbps, so in many aspects the SuperSpeed

channel is different from USB 2.0 high speed channel. The major difference is listed in Table 1.1.

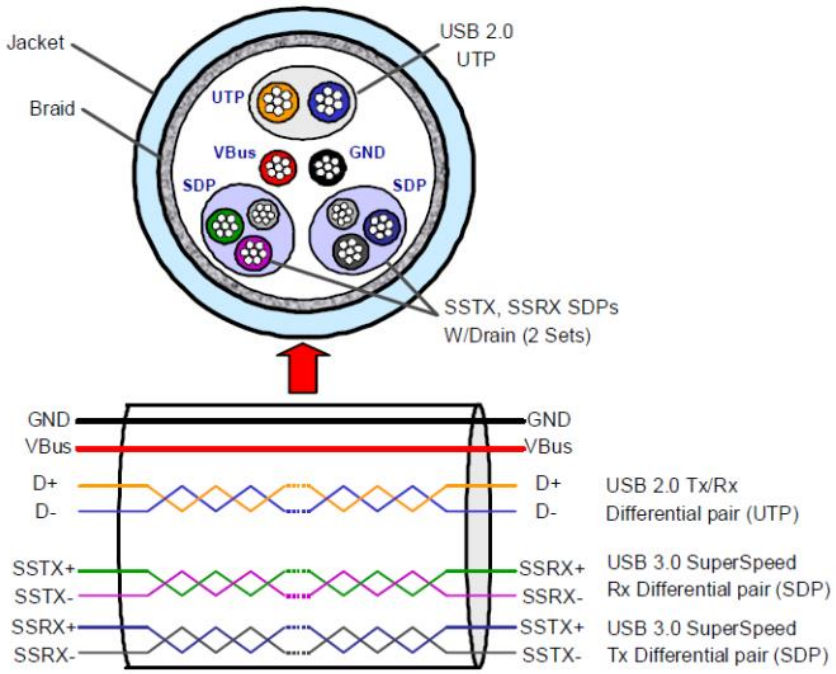


Figure 1.2. Cross section of USB 3.0 cable [16].

Table 1.1. USB 3.0 features compare to USB 2.0.

Feature	USB 3.0 supper speed	USB 2.0 high speed
Data transfer	Dual simplex	Half duplex
Data rate	5Gbps	480Mbps
Device detection by host	Receiver side termination	Pull-up resistor on D+ or D- line

1.3.2. USB 3.0 communication layer

Link layer

The Link layer sets up link between host and slave and maintains link reliability. The Link layer establishes link through link training and status state machine (LTSSM). LTSSM consists of 12 states as listed below.

- U0, fully powered
- U1, standby with fast recovery
- U2, standby with slow recovery
- U3, suspended greatest power savings and longest recovery back to U0 (milliseconds)
- Four link initialization and training states (Rx-Detect, Polling, Recovery, Hot Reset)
- Two link test states (Loopback and Compliance mode)
- SS Inactive: (link error state where USB 3.0 is non-operable)
- SS Disabled: (SuperSpeed bus is disabled and operates as USB 2.0 only) [16]

Protocol layer

The protocol layer performs the following tasks:

1. Unicast: USB 3.0 can deliver packets to the target with the help of routing string.
2. Token/Data/Handshake Sequences: In USB 3.0, the token is incorporated into the data packet for OUT transactions; it is replaced by the handshake for IN transactions. An ACK packet acknowledges the previous data packet sent and requests the next data packet.

The differences IN and OUT transaction sequence between USB2.0 and USB 3.0 are shown in Fig. 1.3 and Fig. 1.4 [16].

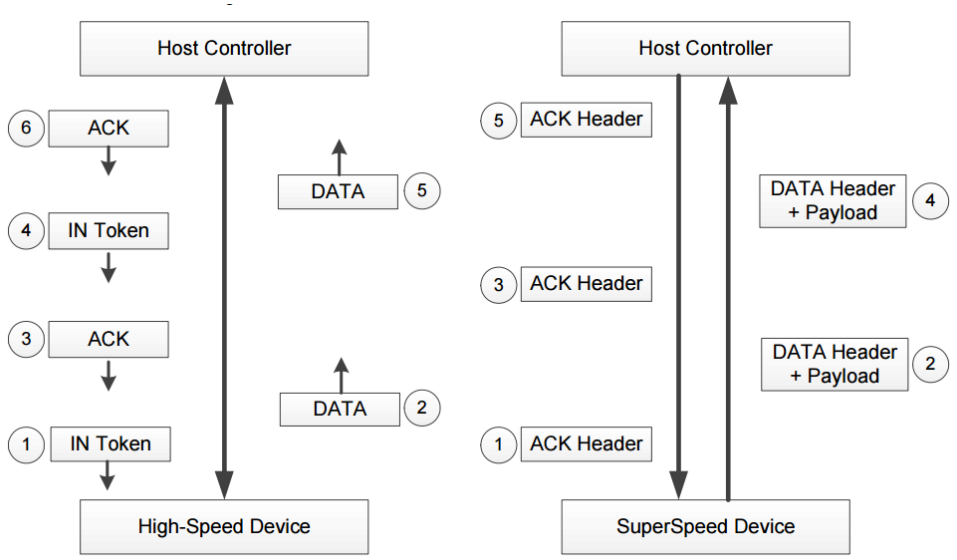


Figure 1.3. Difference IN transaction sequence between USB2.0 and USB 3.0 [16].

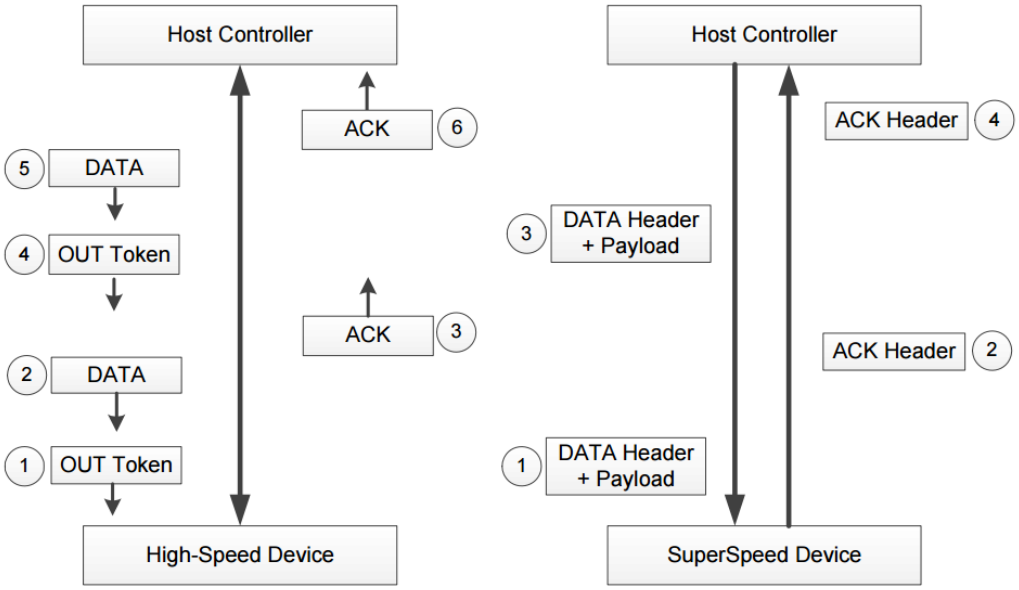


Figure 1.4. Difference OUT transaction sequence between USB2.0 and USB 3.0 [16].

Data bursting

The SuperSpeed end-to-end protocol supports transmitting data in burst to improve performance. The maximum burst size is 16, and the actual number to be used

represents the number of data packets that can be sent without receiving an acknowledgement, as showed in Figure 1.5.

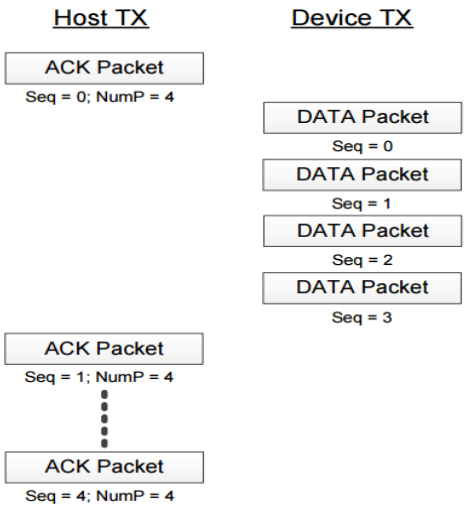


Figure 1.5. Data bursting [16].

End-to-End Flow Control

SuperSpeed flow control uses a poll-once approach coupled with an asynchronous ready notification, as shown in Fig. 1.6.

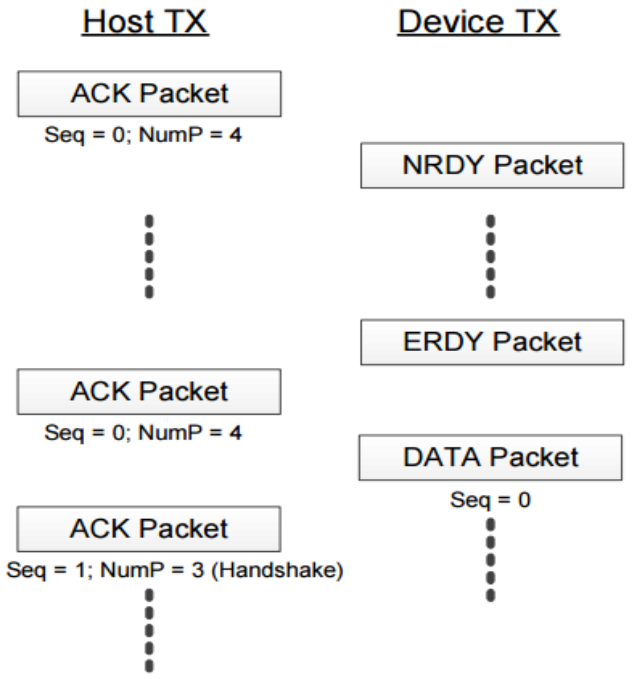


Figure 1.6. End-to-End Flow Cotrol [16].

1.4. USB 3.0 Camera Applications

As showed in Figure 1.7, FX3 sends command to Image Sensor and pick up video signal from Image sensor through video data interface. Video data interface consist of clock, Sync, and data. Sync consists of H-sync and V-sync. Data can be 8-bit,16 bit or 32-bit interface structure. In this design, we use FPGA between Image sensor and FX3 for high speed LVDS translate to parallel data interface.

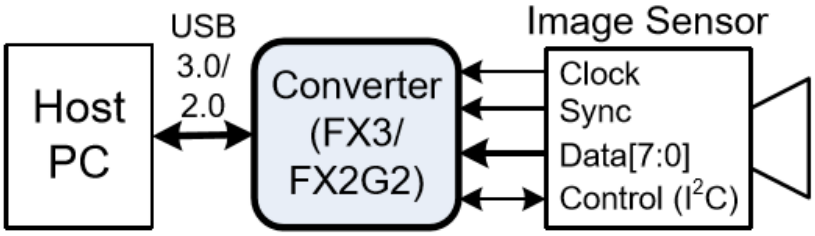


Figure 1.7. USB 3.0 camera block diagram and main interface signals [9].

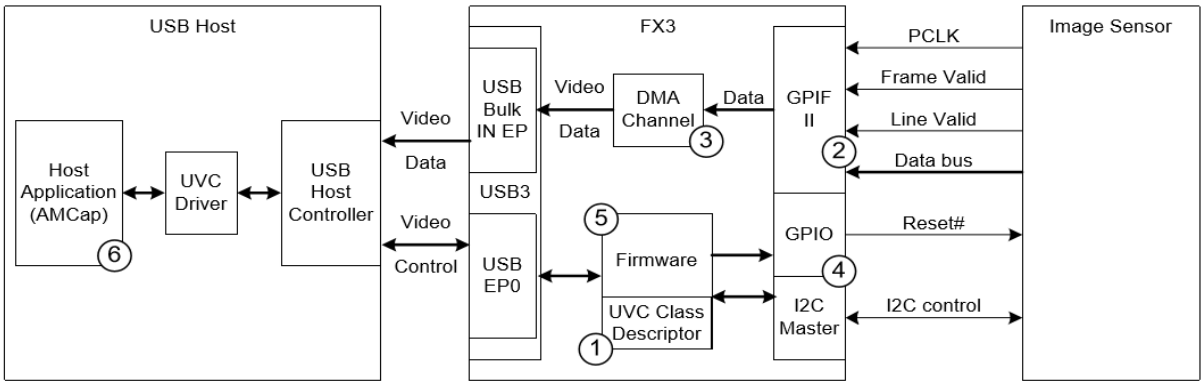


Figure 1.8. Camera application system block diagram [9].

Figure 1.8 shows more details of the USB host and FX3 module, which include a UVC (USB Video Class) descriptor and driver respectively. However, in this design, we did not fully follow the standard UVC process, the reason is listed as following.

1) UVC uses Y, U, V as color video format before transmit to PC USB host. With the limitation of USB 3.0 transmission speed (5Gbps), it only supports 1280x720 @30 FPS.

2) Our design uses Bayer format as color pixel before transmitting to PC USB host. The Bayer to R, G, B conversion is performed by the PC software. Therefore, the maximum video frame rate can reach to 40FPS with the resolution 4096x2160.

Although we did not fully follow UVC standard, the process for data transmission is quite similar to UVC as described in the following.

1.4.1. USB 3.0 control path for features setting

The USB 3.0 control path works as follows. The host will enumerate the device in the initial stage, and will be able to get and set features for specific applications. The process is showed in the diagram below.

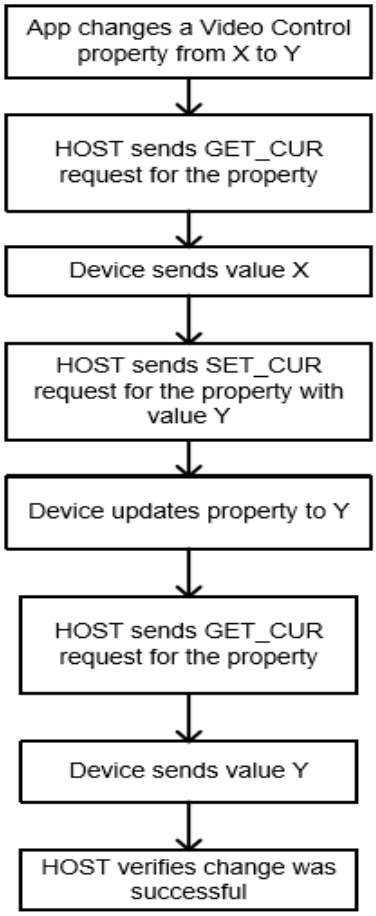


Figure 1.9. Camera features reading and setting [16].

After successfully enumerating the device, the PC software will use USB control transfer PIPE to communicate with FX3 through GET_CUR and SET_CUR. FX3 translates the command sent from PC software to FPGA, and FPGA re-translates the command and data to control the PLL and video sensor, or sends status information to the PC.

1.4.2. EZ-USB FX3 payload data transfer architecture

To successfully transfer data from the GPIF II interface at the sensor to the USB interface at the PC, four major steps are needed, as shown in Fig. 1.10.

1. Set up two GPIF threads, GPIF Thread0 and GPIF Thread1. The two GPIF threads are scheduled by GPIF II state machine, and connected to 2 GPIF Sockets respectively.

2. Set up DMA channel for GPIF sockets and USB3 sockets.

3. Set up DMA Descriptor: a DMA Descriptor is a set of registers allocated in the FX3 RAM. It holds information about the address and size of a DMA buffer as well as pointers to the next DMA Descriptor. These pointers create DMA Descriptor chains.

4. Set up DMA buffer: a DMA buffer is a section of RAM used for intermediate storage of data transferred through the FX3 device. DMA buffers are allocated from the RAM by the FX3 firmware, and their addresses are stored as part of DMA Descriptors.

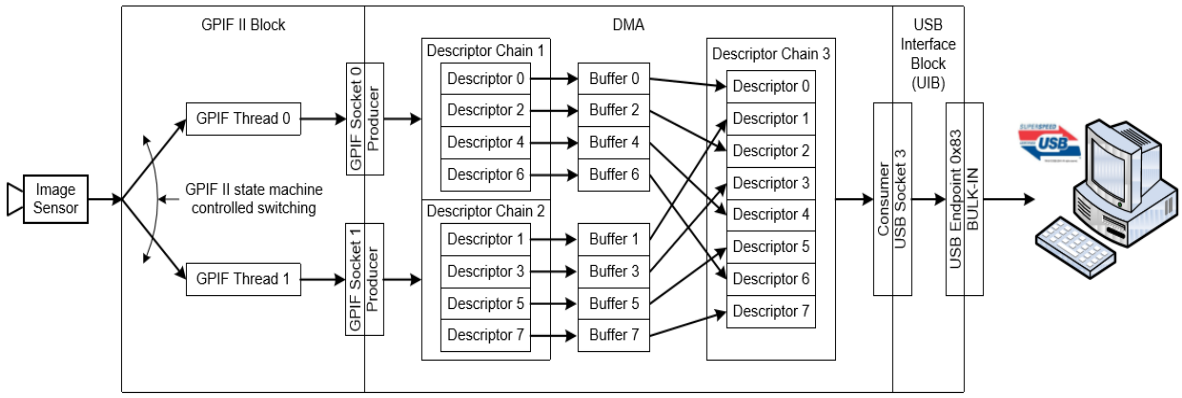


Figure 1.10. FX3 Data transfer architecture [9].

1.5. PC software

The PC software is one of the main parts of machine vision camera, but it is not the focus in this report. Here we just give a brief description of its role in the integrated system.

- Set up USB 3.0 driver and SDK for PC, the driver and SDK can be downloaded from Cypress website [17].

- USB data transfer: read raw data from USB driver and save it to USB data buffer.

- Image conversion: read raw data from the USB data buffer and call various image processing algorithms, include de-mosaicing, white balance, and Gamma color correction.

- Image drawing: get the converted RGB data and call PC API to draw image on the screen.

In the next few chapters, we discuss detailed design of the USB 3.0 camera.

Chapter 2.

Image Sensor and Sensor Board Design

2.1. Image Sensors

In this report (or the related hardware design), two types of image sensor are involved, SONY IMX226 and IMX273.

2.1.1. IMX226 Introduction

Description

The IMX226CQJ_C is a diagonal 9.33 mm CMOS color image sensor. There are 2 main readout modes: one is 10-bit output with approximately 9.03M effective pixels at 59.94 fps, and the static picture of the effective pixels can be 12.4M. Another main readout mode is 12-bit output with resolution 2048x1080 and maximum frame rate of 60fps.

Key Features

- Input clock frequency: The maximum input frequency can be 72MHz
- Readout mode
  - All-pixel scan mode with 10-bit output
  - All-pixel scan mode with 12-bit output
  - Horizontal/vertical 2/2-line binning mode
- Image quality and control:
  - High-sensitivity, low dark current, no smear, excellent anti-blooming characteristic
  - Electronic shutter function with variable storage time
  - CDS (correlated double sampler) on chip
  - PGA on chip. Gain +27dB （step <0.1 dB）
  - R, G, B primary color mosaic filter on chip
- Control interface:
  - I2C or SPI serial communication circuit on chip

These features determine the clock tree, the control interface, and ISP (in system processing) design with implementation on sensor hardware board or FPGA code design.

Recommended operation condition

Supply voltage:	VADD	2.9±0.1V.*1
	VDDD1	1.2±0.1V.*2
	VDDD2	1.8±0.1V.*3
Input voltage:	VI	-0.1 to VDDD2+0.1V
Output voltage:	Vo	-0.1 to VDDD2+0.1V

*1 VADD: VDDSUB, VDDHCM, VDDHVS, VDDHPX, VDDHDA, VDDHCP (2.9V power supply)
*2 VDDD1: VDDLCN1, VDDLCN2, VDDLSC1 and VDDLSC2, VDDLPL (1.2V power supply)
*3 VDDD2: VDDMIO, VDDMLV1 and VDDMLV2(1.8V power supply) [1]

2.1.2. IMX273 Introduction

The IMX273LLR-C is a diagonal 6.3 mm CMOS sensor with 1.58M effective pixels, global shutter, variable charge-integration time. The power supply for this chip set are analog 3.3V, digital 1.2V and interface 1.8V respectively, and has low power consumption.

Key Features

- Built-in timing adjustment circuit which can be configured through I2C or SPI communication interface.
- Master or slave mode
  IMX 273 can work on master mode or slave mode which can be configured through I2C or SPI interface. When working on master mode, H-sync and V-sync signals are generated by sensor (output from sensor). When working on slave mode, H-sync and V-sync are input signals from external master such as FPGA.

- Input frequency:

  Three options are available: 37.125 MHz, 74.25 MHz or 54MHz. In this design 37.125 MHz is selected. Combined with PLL inside FPGA, variable clock rate is provided to adapt the different frame rate requirements.

- Readout mode

- All-pixel scan mode with different frame rate
- Vertical 2-pixel FD Binning mode
- 2 X 2 Vertical FD binning mode
- ROI mode
- Vertical/Horizontal –Normal / Inverted readout mode

- Frame rate

In all-pixel scan mode, the maximum frame rate depends on data output structure. Different ADC sampling widths or pixel output widths will result in different frame rates. This speed reflects how fast the sensor disposes video pixels. Combined with FPGA PLL to adjust input clock, variable frame rate will be generated. The maximum frame rate is listed below:

- 8-bit: 276.0 frame/s;
- 10-bit: 226.5 frame/s;
- 12-bit 165.9 frame/s

- shutter function
  - Variable-speed shutter function (resolution 1 H units)

- ADC sampling bit width

  - 8-bit ADC mode
  - 10-bit ADC mode
  - 12-bit A/D converter

- Output bit width

  - Also has three opyions: 8-bit / 10-bit /12-bit

- I / O interface

  - Low voltage LVDS (150 m V_{p-p})
  - serial (2 ch / 4 ch / 8 ch switching) DDR output

Recommended operating conditions

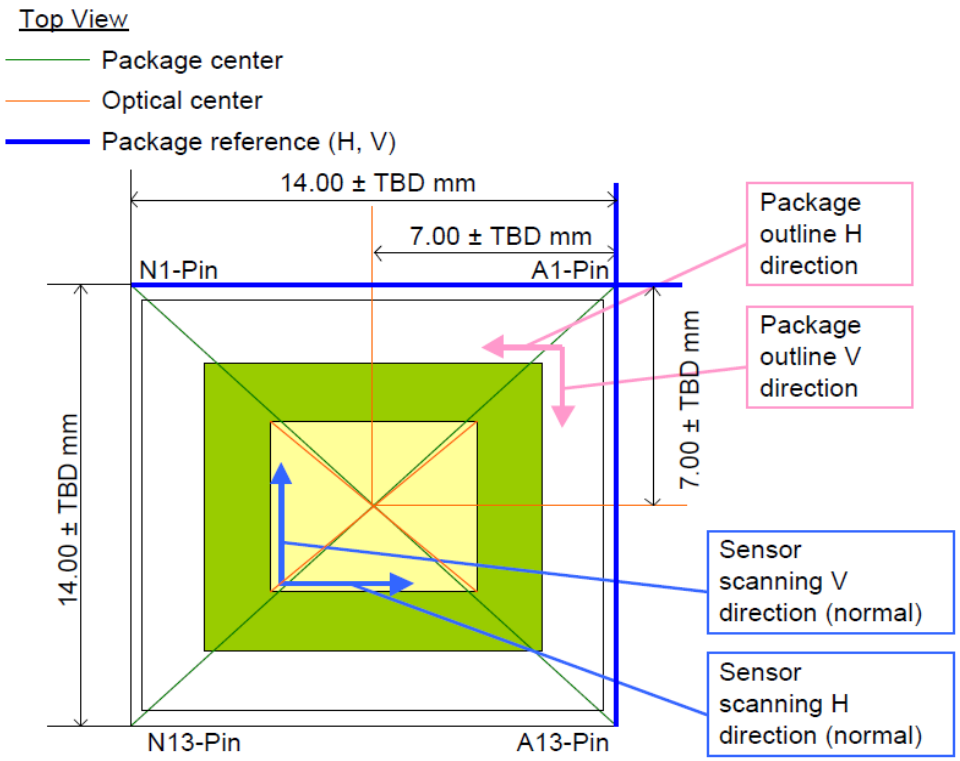
The recommended operating conditions are shown in Table 2.1.

Table 2.1. Supply voltage for IMX273.

Item	Symbol	Min.	Typ.	Max.	Unit
Supply voltage (Analog 3.3V)	AV_{DD}	3.15	3.30	3.45	V

Supply voltage (Interface 3.3V)	OV_{DD}	1.70	1.80	1.90	V
Supply voltage (Digital 3.3V)	DV_{DD}	1.10	1.20	1.30	V

Chip Center and Optical Center



Optical Center

Each sensor will provide the positions of chip center and optical center. The chip center is the center of external mechanical size of the sensor. It is determined by chip scale, footprint etc. The optical center is the effective optical area center, which will affect image region if it is improperly aligned with lens. Normal chip center is coincided with optical center, but there are some cases that the chip center is different from optical center. In these cases when we design PCB board and camera case, we need to make sure that the optical center is fully aligned with lens center. Figure 2.1 show the relationships in IMX273.

Pixel Arrangement

Figure 2.2 list different areas to designate different number of pixels such as total number of pixels, effective pixels, active pixels and recommended recording pixels. Considering OB and effective margin for color processing, the real image pixel number will be 1440x1080.

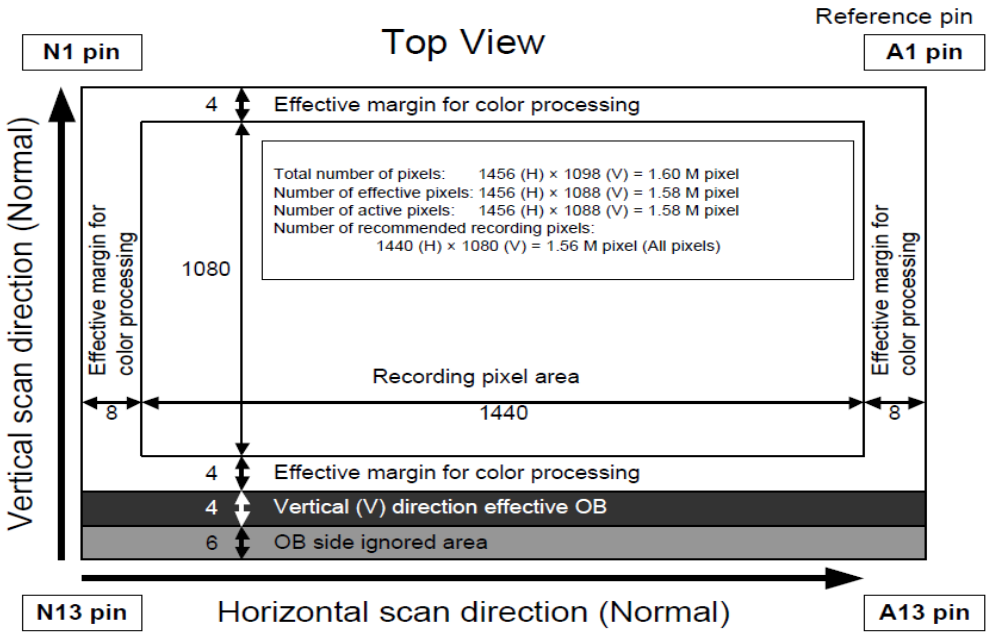


Figure 2.2. Pixel scan direction.

2.2. Sensor hardware PCB design

2.2.1. Power supply for sensor board

Power supply contains three voltages, 1.2 V digital, 1.8 V interface, and 3.3V (2.9V) Analog.

For IMX273, the Analog supply is 3.3V while IMX226 needs 2.9V. For compatibility consideration, to make sure the sensors have a uniform interface to the mainboard (U3 main board or GE main board), the power supply design block diagram is shown as in Figure 2.3, which includes converters to 2.9 V,1.8 V and 1.2 V.

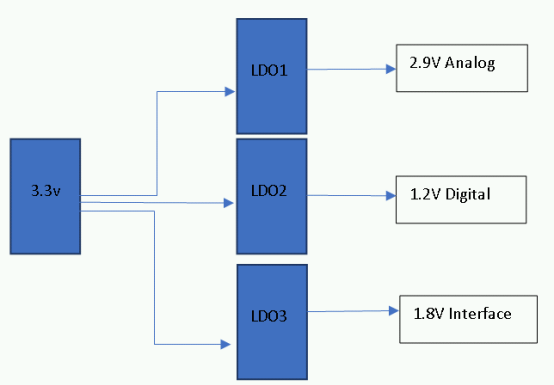


Figure 2.3. Power regulation for sensor board.

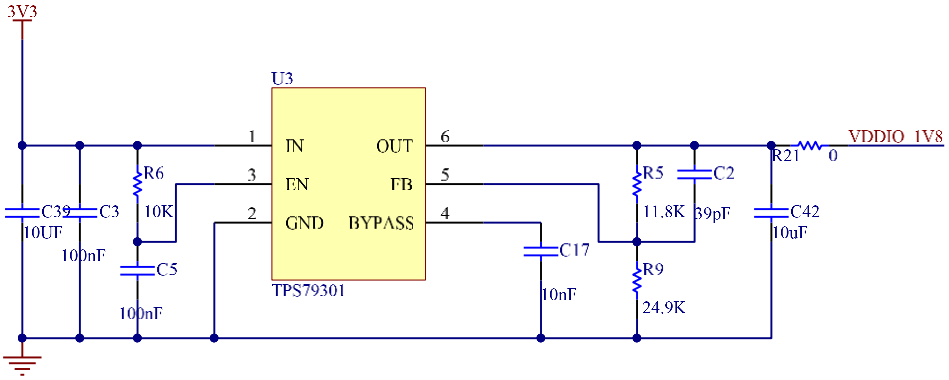


Figure 2.4. 3.3V to 1.8V converter.

When connecting the power to sensor chipset, some critical factors must be considered:

1. Analog power supply must use LDO to avoid large ripple.
2. Add ferrite bead to each power supply branch before connecting to sensor.
3. At each power pins, put a typical shunt capacitor, whose capacitance depends on the power and characteristic of the load. If the load is heavy, one capacitor with several uF capacitance must be used. Along with the large value capacitor, small values of capacitors will be used for coupling high frequencies to ground. The typical circuit is shown in Figure 2.4.

2.2.2. Sensor board connector

The sensor board connects to the main board through a connector consisting of 4 parts as described below:

1. Data path:12 pairs LVDS (or MIPI format) data channels
2. 3 pairs LVDS (sub LVDS) clock path along with 4 data channels each.
3. CLOCK to sensor: which is normally provided by FPGA PLL, for clock selection flexibility
4. Control signals: such as H-sync, V-sync SPI or I2C etc.

Connector signals are defined as Figure 2.5.

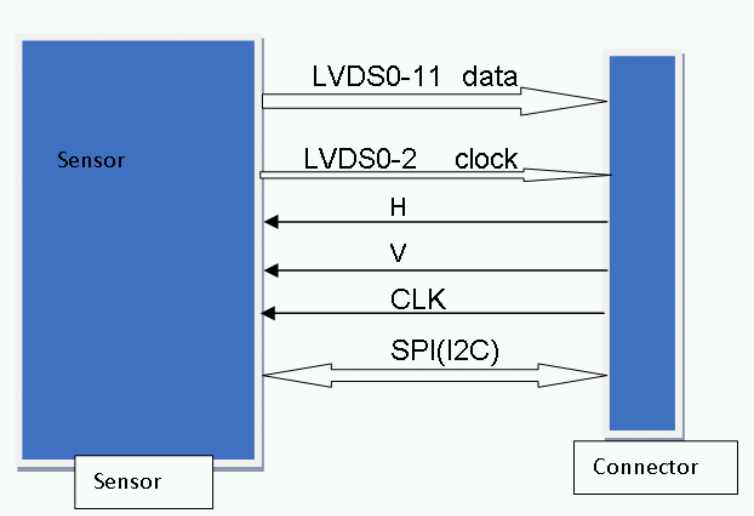


Figure 2.5. Connector on sensor board.

2.2.3. Sensor board layout Requirement

The sensor board uses 6-layer PCB layout. According to the requirement of the transmission speed, in order for the sub-LVDS to perform at 720Mbps, the wires of differential pairs need to have equal lengths. The tolerance should be within 10 mils.

The terminal resistor of differential pair should match the transmission impedance. Here we keep 100Ω for differential pair and 50 Ω for single-ended transmission line.

All traces are calculated according to PCB stack up requirement, trace width. The separation distance of differential pair must be kept within the tolerance to satisfy the impedance specs.

Chapter 3.

Main Board Design

3.1. Block Diagram of Main Board

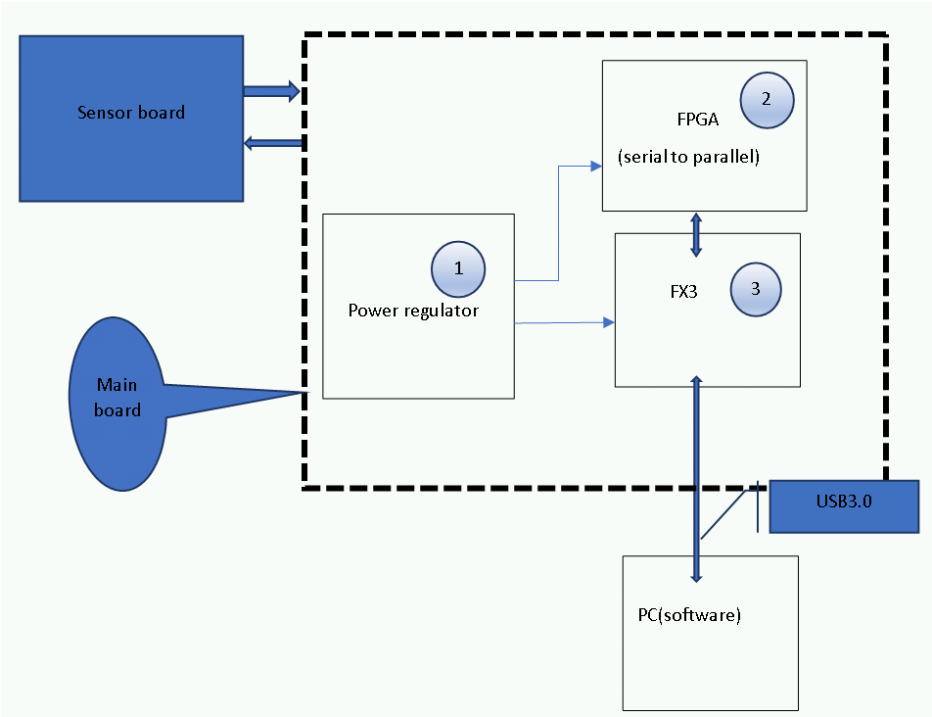


Figure 3.1. Functionality block diagram of main board.

The main board consists of 3 main parts, as showed in Figure 3.1:

1. Power supply,
2. FPGA,
3. FX3.

The following paragraphs will introduce the characteristics of the main components (FPGA and FX3), and discuss selection considerations of relevant components.

3.2. FPGA Hardware

3.2.1. FPGA Model Selection

The main functions of FPGA in U3 camera system are: (1) receive bit stream from LVDS channel. (2) read or write control information from/to sensor control registers to control sensor's behavior.

After a rough estimate on Altera evaluation board, Cyclone 4 EP4CE10U14I7 FPGA has been chosen based on the resource, the number of differential I/O pins and the speed of LVDS interface. Cyclone 4 EP4CE10U14I7 FPFA has the following features:

1. low power,
2. high functionality,
3. low cost.

Logic Resource

The logic resource of the EP4CE10U14I7 FPGA is shown in table 3.1.

Table 3.1. FPGA Logic Resource.

LEs	Embedded memory	18X18 multiply	General-Purpose PLLs	Global clock	I/O banks
10320	414k	23	2	10	8

I/O resource of EP4CE10U14I7

For the specific footprint package of F256, the I/O resource is listed in table below.

Table 3.2. FPGA I/O Resource.

User I/O	LVDS	CLK PINS
179	66	15

Cyclone IV I/O Elements

Cyclone IV I/O elements (IOEs) structure is showed in Figure 3.2.

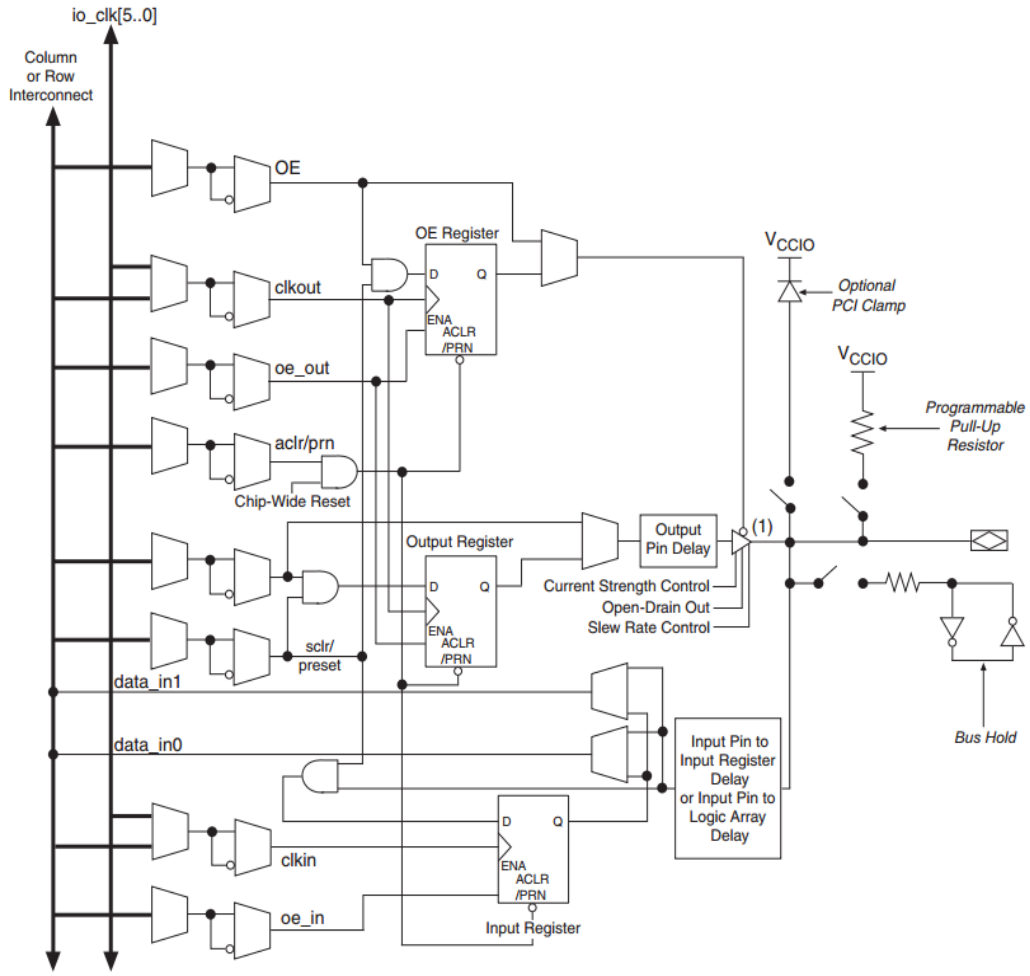


Figure 3.2. Cyclone IV IOEs in a Bidirectional I/O Configuration for SDR Mode [15].

The I/O element resources are listed as follows.

1. bidirectional I/O buffer
2. support various single-ended I/O standards.
3. support various differential I/O standards.

4. The IOE contains one input register, two output registers, and two output-enable (OE) registers.

5. The two output registers and two OE registers are used for DDR applications.

6. The input registers can be used for fast setup times and output registers for fast clock-to-output times. Additionally, the OE registers can be used for fast clock-to-output enable timing.

7. Normally IOEs can be used for input, output, or bidirectional data paths, unless it is specified as dedicated input or clock pins [15].

Table 3.3 lists the I/O standards that Cyclone IV devices support.

Table 3.3. I/O Standards Support for the Cyclone IV Device Family.

Type	I/O standard
Single-ended I/O	LVTTL, LVCMOS, SSTL, HSTL, PCI, and PCI-X
Differential I/O	SSTL, HSTL, LVPECL, BLVDS, LVDS, mini-LVDS, RSDS, and PPDS

I/O Element Features

The Cyclone IV IOE offers a range of programmable features for an I/O pin. These features increase the flexibility of I/O utilization and provide a way to reduce the usage of external discrete components, such as pull-up resistors and diodes [15].

Programmable Current Strength

The output buffer can be programmable to drive different I/O standard, such as LVTTL, LVCMOS, SSTL-2 Class I and II etc. Different I/O standards have several levels of current strength with fully controlled within Quartus II. The current strength setting procedure is:

(1) Determine each I/O load, and margin,
(2) Calculate the total current load to ensure the total current load within the current budget within one I/O bank,
(3) Assign current strength for each I/O.

The current strength control can help decrease the effects of simultaneously switching outputs (SSO) in conjunction with reducing system noise. When programmable current strength has been used, the on-chip series termination (RS OCT) is not available [15].

Slew Rate Control

Slew rate control is available for single-ended I/O standards with current rating from 8 mA or higher.

Fast transition is essential for high speed transceiver; however, these fast transitions may introduce transient noise. A slower slew rate reduces system noise, but adds delay to rising and falling edge.

Because each I/O pin has an individual slew-rate control, we can specify the slew rate on a pin-by-pin basis. The slew-rate control affects both the rising and falling edges. There are 2 exceptions listed below.

1. The programmable slew rate feature cannot be used when using OCT with calibration.
2. the programmable slew rate feature cannot be used when using the 3.0-V PCI, 3.0-V PCI-X, 3.3-V LVTTL, or 3.3-V LVCMOS I/O standards. Only the fast slew rate (default) setting is available.

Open-Drain Output

In some situation, we need Open-Drain output features for system level control, in this design I2C is a control interface from FPGA to sensor. Both SCL and SDA signals need open-Drain output design. Cyclone IV devices provide an optional open-drain (equivalent to an open-collector) output for each I/O pin.

Programmable Pull-Up Resistor

The pull-up resistor holds the output to the VCCIO level of the output pin’s bank. Each Cyclone I/O pins can enable and disable Pull-Up Resistor. Some exceptions are:

1. Programmable pull-up resistor and bus-hold feature cannot be used simultaneously.
2. Dedicated configuration, JTAG, and clock pins do not support Programmable Pull-Up Resistor feature.

3. DEV_OE signal drives low, all I/O pins remains tri-stated even with the programmable pull-up option is enabled.

Programmable delay

FPGA IOE programmable delay feature is much helpful on high speed transmission and receiving. Especially with the help of input or output delay, FPGA logic can compensate data and clock path when it goes a long journey from sensor to FPGA internal register. Three situations are listed below.

(1) Input delay, use the programmable delays to ensure zero hold times, minimize setup times.
(2) Output delay, use Programmable delays can increase the register-to-pin delays for output registers.
(3) clock delay, use programmable delay to increase the delay to the global clock networks.

Table3.4 shows the programmable delays for Cyclone IV devices.

Table 3.4. Table 3.4 programmable delays for Cyclone IV device.

Programmable delay	**Quartus II logic option**
Input pin-to-logic array delay	Input delay from pin to internal cells
Input pin-to-register delay	Input delay from pin to input register
Output pin delay	Delay from output register to output pin
Dual purpose clock input delay	Input delay from dual purpose clock pin to fan-out destinations

I/O Banks

AS showed in Fig. 3.3, Cyclone IV consists of 8 I/O banks, each bank support single ended VCMOS 3.3-V LVTTL/LVCOMS,2.5-V LVTTL/LVCOMS,1.8-V LVTTL/LVCOMOS and differential mini-LVDS interface.

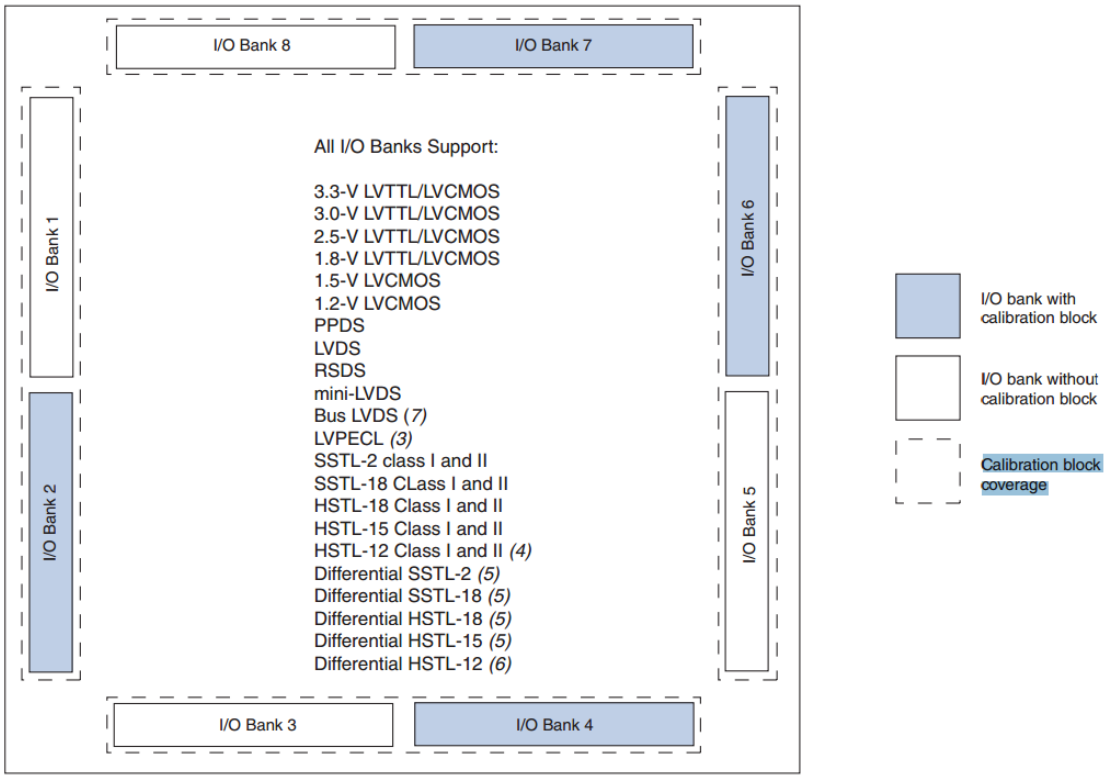


Figure 3.3. I/O banks [15].

According to the description on I/O features and I/O banks partition, Bank 4 and 5 are selected for mini-LVDS and bank voltage level is 1.8V, which satisfy mini-LVDS requirement. Since Cyclone IV does not have OCT support for mini-LVDS, so 100 Ω external terminal resistor is required for each LVDS pair. The corresponding circuitry design is showed in figure 3.4 below.

Bank 3 power supply is also arranged to be 1.8V voltage level. We can use I/O pins and internal logic to implement a high-speed differential interface in Cyclone IV devices.

Cyclone IV devices do not contain dedicated serialization or deserialization circuitry. We use general element to construct the SERDES functionality.

(1) shift registers, internal registers inside FPGA logic cell,
(2) internal phase-locked loops (PLLs),
(3) I/O cells, I/O register, DDR interface, programmable delay, differential buffers

The differential interface data serializers and deserializers (SERDES) are automatically constructed in the core logic elements (LEs) with the Quartus II software ALTLVDS megafunction. All I/Os in this bank are connected to sensor for sensor feature control.

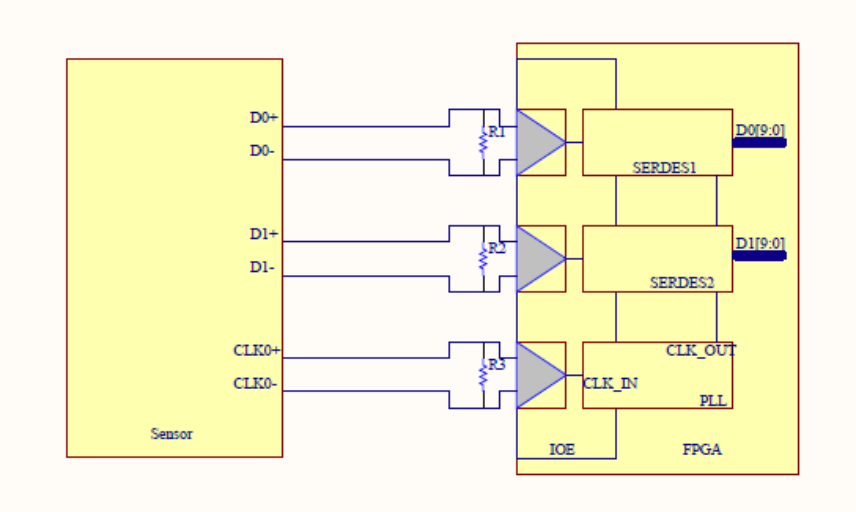


Figure 3.4. FPGA LVDS interface design.

RX_LVDS initiation in Quartus II

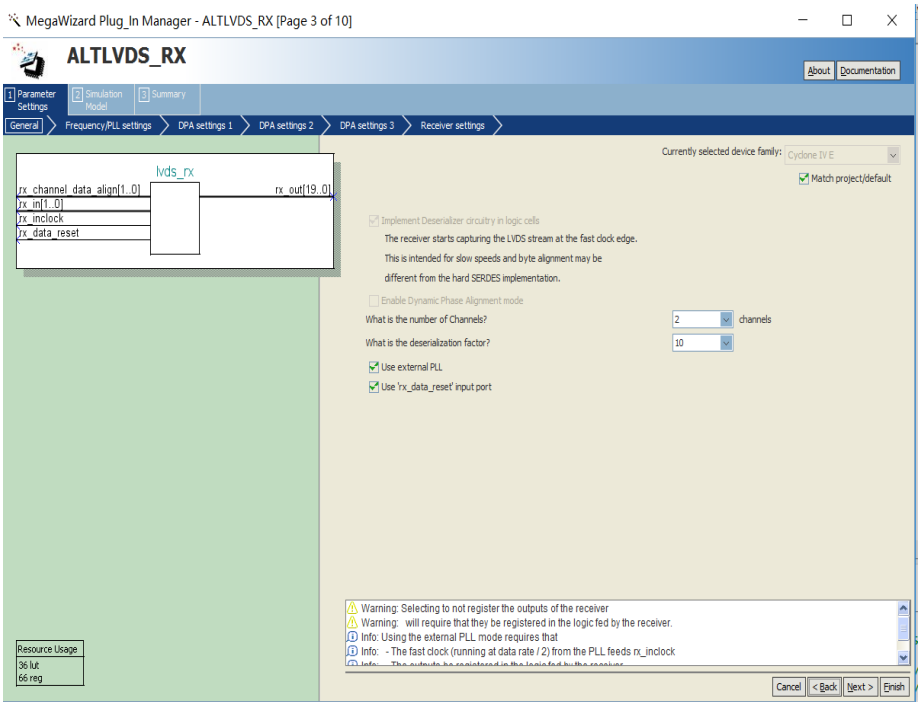


Figure 3.5. LVDS module initiation in Quartus II.

In Figure 3.5, parameters and control signal are selected as following.

1. Each RX_LVDS support 2 RX channel.
2. Each channel Deserialization factor is 10 (for sensor ADC bit width equal to or less than 10, if ADC is 12-bit width, each channel need deserialization factor 6, and combine two of them to one 12 bit).
3. Use “rx-data-reset” in case of RX_LVDS state machine becomes disorder.
4. Use “rx-channel-data-align” for parallel data synchronization.
5. How to use these control signal please refer to FPGA coding part.

Clock Management

Cyclone IV devices include up to 30 global clock (GCLK) networks and up to eight PLLs with five outputs per PLL to provide robust clock management and synthesis. We can dynamically reconfigure Cyclone IV device PLLs in user mode to change the clock frequency or phase. General purpose PLLs are used for general-purpose applications in the fabric and periphery, such as external memory interfaces [15].

Cyclone IV devices use SRAM cells to store configuration data. Altera provides several ways to configure FPGA.

(1) JTAG interface, the interface follow IEEE 1149.1 standard I/O interface, can directly download *.sof file to FPGA device.

(2) Active serial configure mode. In this configuration mode, during power up, FPGA actively generates DCLK signal along with other control signal to drive the serial EPROM, and read configuration data from EPROM and configures by itself.

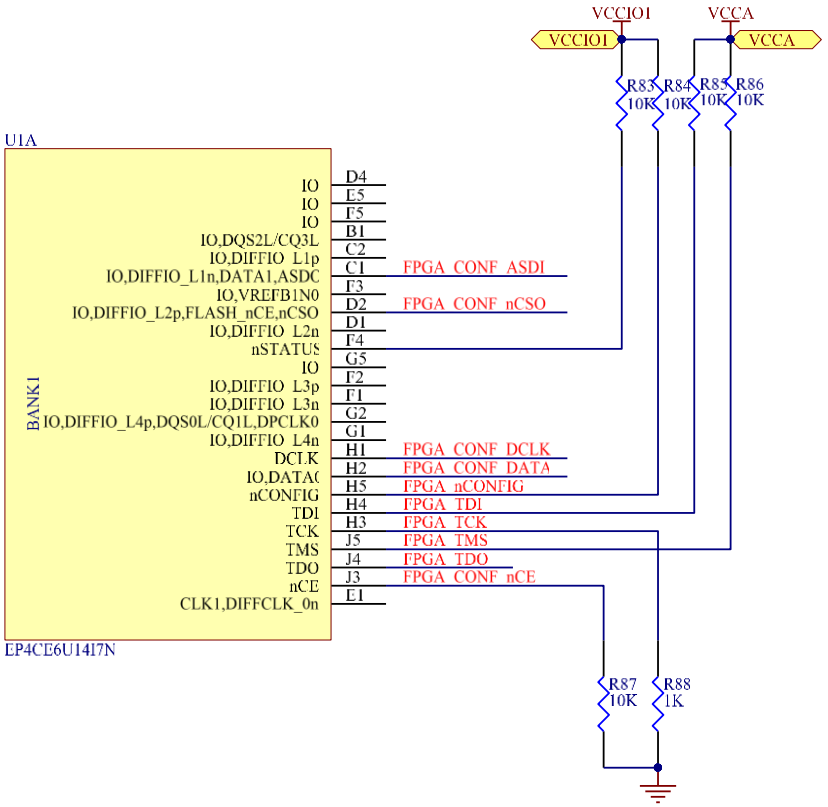


Figure 3.6. FPGA configuration interface.

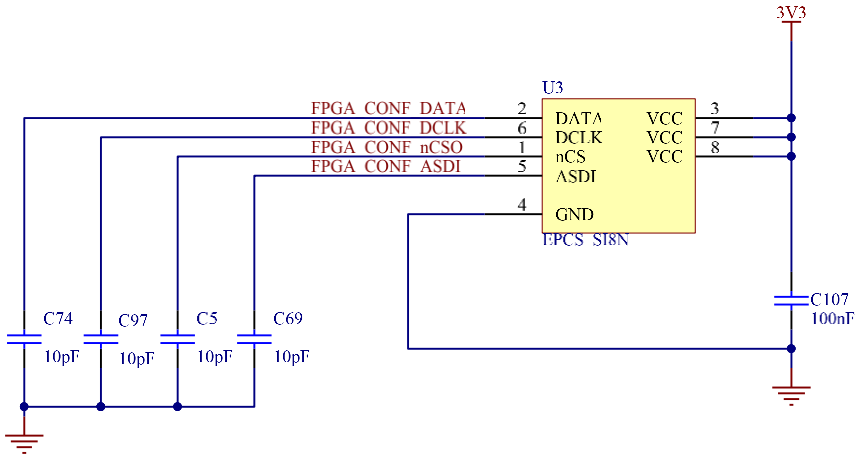


Figure 3.7. FPGA active serial configuration memory interface.

3.3. EZ-USB FX3

3.3.1. Introduction

Cypress's EZ-USB® FX3™ is the USB 3.0 peripheral controller with a fully-configurable, parallel, general programmable interface called GPIF II, which can connect to any processor, ASIC, or FPGA. FX3 has an embedded 32-bit ARM926EJ-S microprocessor for powerful data processing and for building custom applications. It implements an architecture that enables 375 MBps data transfer from GPIF II to the USB interface [8].

FX3 also provides interfaces to connect to serial peripherals such as UART, SPI, I2C, and I2S. FX3 comes with application development tools. The software development kit comes with application examples for accelerating time to market [8].

3.3.2. Schematic design for FX3

FX3 in U3 camera system is a very critical part, as all video data must serially pass through USB 3.0 interface whose transmit speed can be up to 5Gbps. Several guidelines for trace width, stack up, and other layout must be followed to ensure the system will perform as expected, especially for USB 3.0 to work properly.

Power supply network

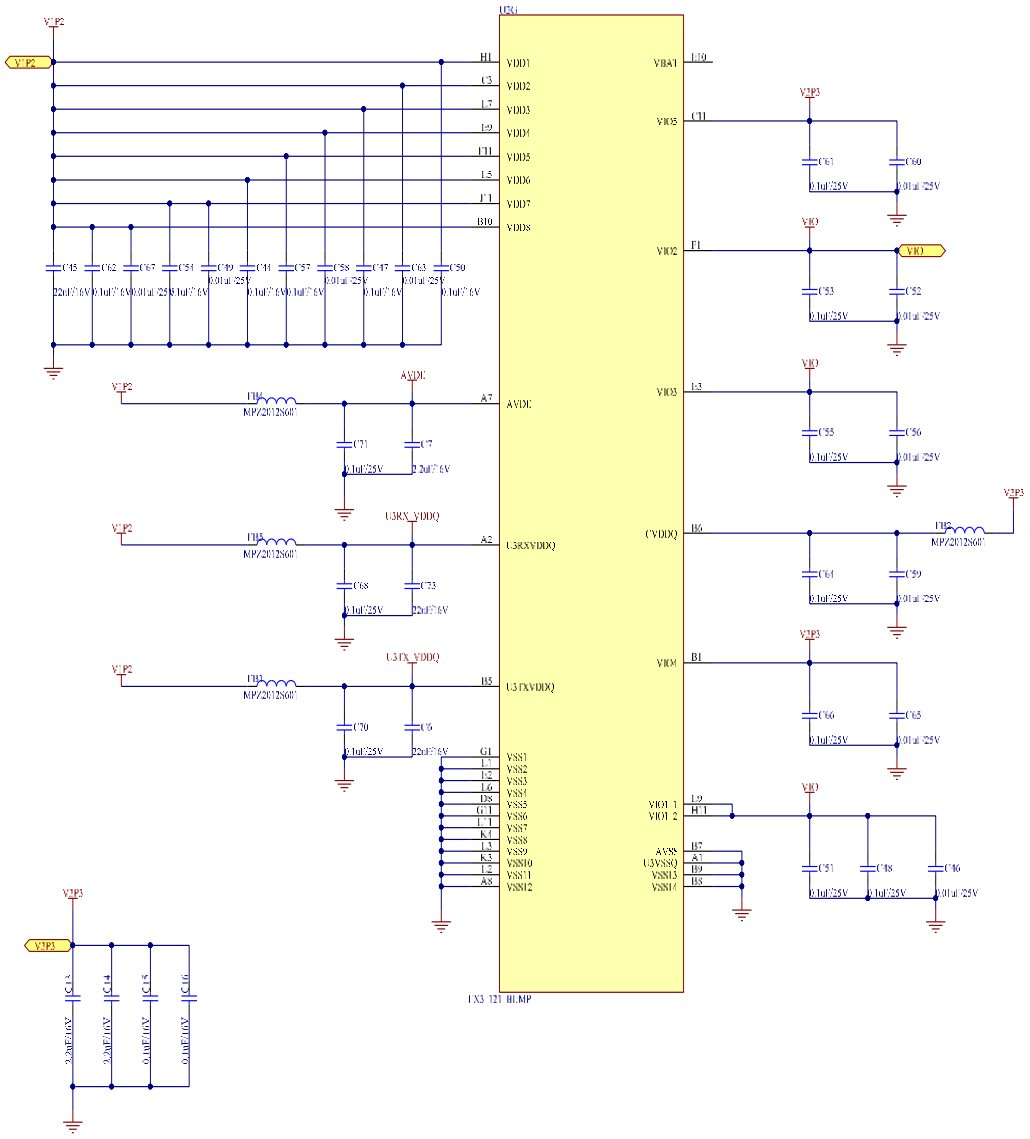


Figure 3.8. EZ-USB FX3 Power domains diagram.

EZ-USB FX3 Power Domains voltage setting is showed in Table 3.5. In this design, the typical voltage value is chosen as VIO1, 3.3V; VIO1, 3.3V; VIO2, 3.3V; VIO3, 3.3V; VIO4, 3.3V; VIO5, 3.3V; CVDDQ,3.3 V;

Table 3.5 . FX3 power supply voltage level setting.

Name	Min (V)	Typical(V)	Max(V)	Description

VDD	1.15	1.2	1.25	Core voltage
AVDD	1.15	1.2	1.25	Analog voltage
VIO1	1.7	1.8,2.5 and 3.3	3.6	GPIF II I/O
VIO2	1.7	1.8,2.5 and 3.3	3.6	IO2 power domain
VIO3	1.7	1.8,2.5 and 3.3	3.6	IO3 power domain
VIO4	1.7	1.8,2.5 and 3.3	3.6	UART/SPI/I2S power domain
VIO5	1.15	1.2,1.8,2.5 and 3.3	3.6	I2C and JTAG
VBATT	3.2	3.7	6	USB voltage supply
VBUS	4.0	5	6	USB voltage supply
CVDDQ	1.7	1.8,3.3	3.6	Clock voltage
U3TXVDDQ	1.15	1.2	1.25	USB 3.0 TX voltage
U3RXVDDQ	1.15	1.2	1.25	USB 3.0 RX voltage

Inrush Current and Power Supply Design

An inrush current with magnitude as high as 800mA will appear on 1.2 V U3RXVDDQ and U3TXVDDQ supplies for around 10 μs, which will cause 1.2 V power supply drop. If VDD (1.2V core supply) happens to use the same 1.2 V source, care must be taken to make sure that the Inrush current will not cause dramatic drop on VDD, since the POR circuit of FX3 will start if VDD drop to below 0.83V with the duration last longer than 200 ns [9].

There several ways to solve this problem:

(1) Separate VDD (core voltage) supply with other 1.2 V power supply,

(2) Use Choke (inductor) to separate the VDD with other 1.2V power supply and use capacitor to smooth the 1.2 V DC voltage. This is the recommended way by reference design. As showed in figure 3.10, we followed the recommended way by reference design.

Figure 3.9 to figure 3.11 give the details of analysis.

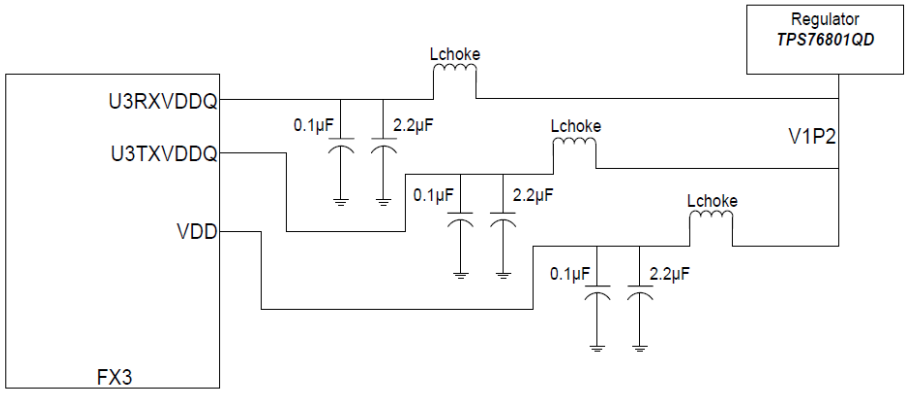


Figure 3.9.Non-optimized 1.2V power supply design. [9]

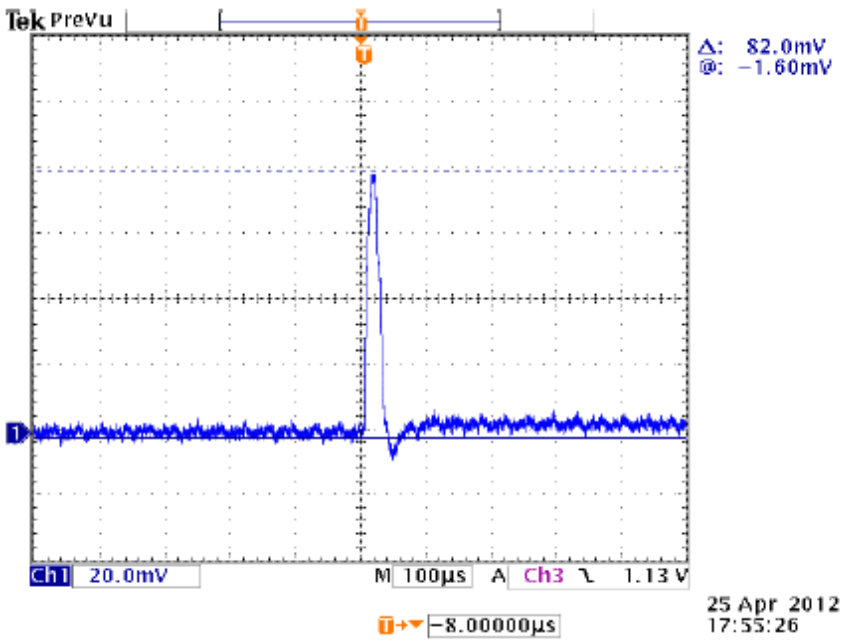


Figure 3.10. Inrush Current (80 mV / 0.1 Ω = 800 mA) [9].

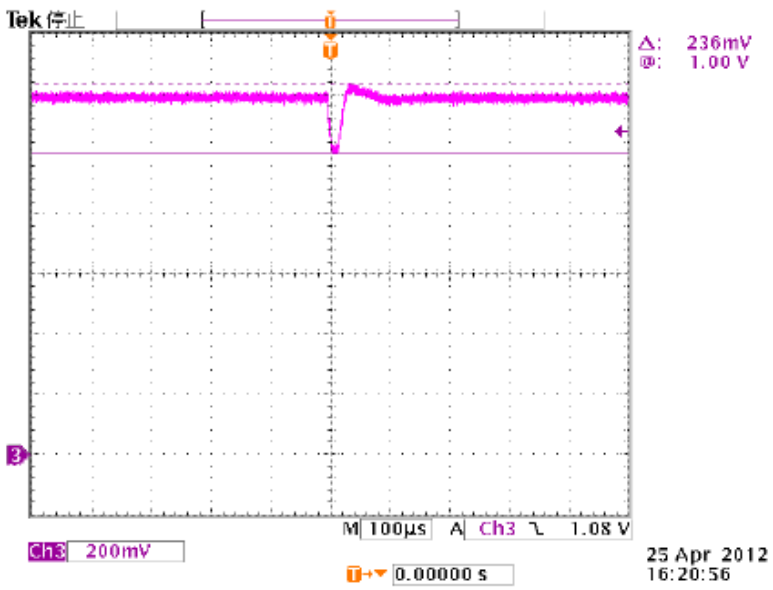


Figure 3.11. 1.2V Power Domain Voltage Drop (200 mV) [9].

In contrast, the modification as showed in Figure 3.12 below uses the same regulator (TPS76801QD), with the modification of using a 22-µF decoupling capacitor, and the inrush is showed in Figure 3.13. The power supply drop is showed in Figure 3.14. The improvement can be observed clearly.

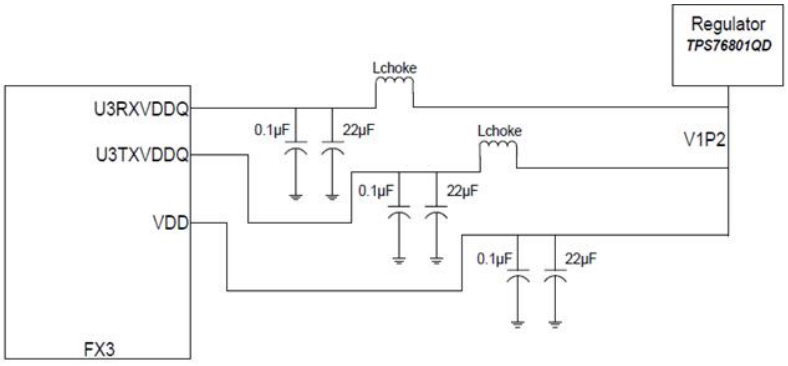


Figure 3.12. Optimized power supply design [8].

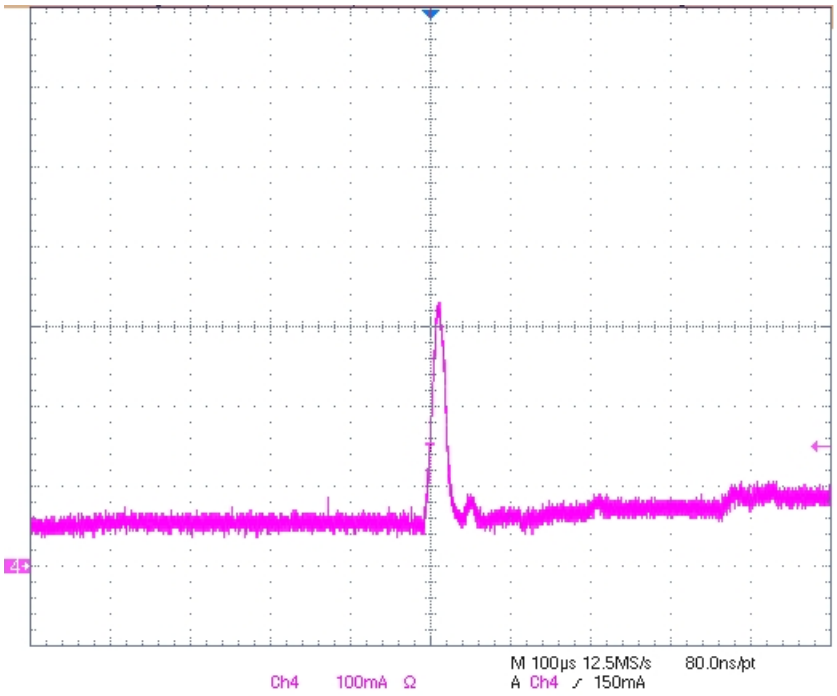


Figure 3.13. Inrush Current (320 mA).

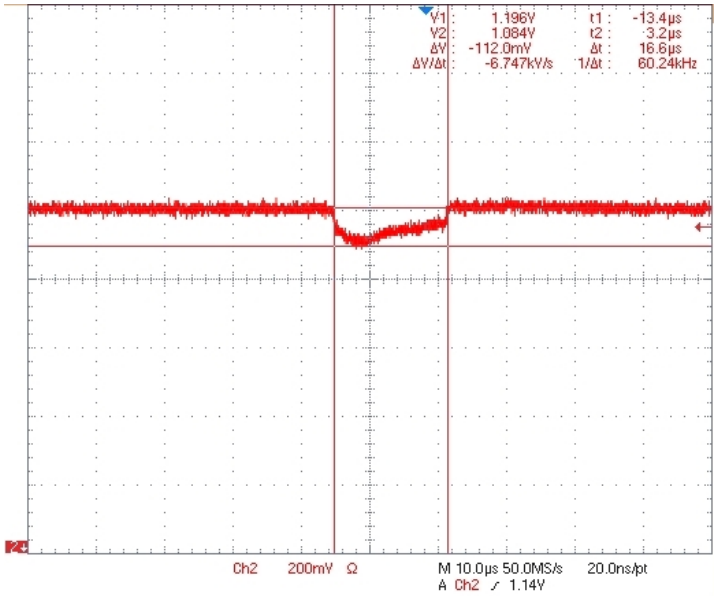


Figure 3.14. 1.2V Power Domain Voltage Drop (112 mV) [9].

It is always a good practice to isolate different power supplies from each other, as mentioned before. To short IO power supplies (VIO1-5) to CVDDQ, it is always recommended to isolate CVDDQ using a choke. This will help in reducing the PHY errors. Operating VIO1 at lower voltages (1.8 V) also helps in reducing the PHY errors, but considering the voltage level needs to be compatible to other parts on FPGA bank 2 and bank 3, we choose 3.3 V voltage power supply for VIO1.

3.3.3. Clocking

The EZ-USB FX3 device can use any of 19.2 MHz, 26 MHz 38.4 MHz, or 52 MHz clock as the clocking source, either crystal or active oscillator. In this design, crystal has been used for clock source.

3.2.3.1. Crystal selection

In this design, we use 19.2M crystal as recommended by Cypress. The requirement of crystal is listed in Table 3.6.

Table 3.6. The requirement of crystal.

Parameter	specification	Unit
Tolerance	±100	Ppm
Temp level	-40--85	°C
Drive level	USE Equation 1	mW

The power dissipation of the crystal depends on:

1. The drive level of the XTALOUT pin (for FX3, this is 1.32 V),
2. The desired frequency (19.2 MHz), and
3. The equivalent resistance of the crystal

The equation is given by

$$P=I^2R=2[\pi f\ (C0+\ CL)\ Vx]^2R \quad (1)$$

where f is the crystal frequency; C0 is the shunt capacitance of the crystal obtained from the crystal datasheet; CL is the load capacitance; R is the crystal ESR obtained from the data sheet of the crystal; Vx is the maximum voltage on XTALOUT pin.

3.2.3.2 Crystal Effective Load Capacitor Calculation

Load capacitance CL plays a critical role in providing an accurate clock source to FX3. The capacitors C1 and C2 must be chosen carefully based on the load capacitance value of the crystal.

The load capacitance is calculated using the following equation:

$$CL=\frac{c1*c2}{c1+c2}+Cs \quad (2)$$

Cs is the stray capacitance of XTALOUT and XTALIN traces on the PCB. Usually the value of Cs is around 2-5 pF as long as good layout practice is followed and the trace from the crystal to the pins on the EZ-USB FX3 is kept as short as possible [9].

3.4. Considerations for main board layout and impedance matching

3.4.1. Impedance requirement

USB differential transmission line impedance

There are 3 pairs of transmission lines on USB interface, D+, D- for high speed usb2.0 differential pair, SSTX+, SSTX- for super speed USB 3.0 transmitter differential pair, SSRX+ and SSRX- for super speed USB 3.0 receiver differential pair. Each impedance between differential pair is 90Ω. EZ-FX3 receiver is already integrated 90Ω terminal resistor for termination matching to prevent reflection from terminal.

LVDS differential pair impedance

LVDS has 100 Ω differential impedance requirement, so on the receiver side near the terminal a 100Ω resistor should be used for impedance matching. Since Cyclone IV FPGA I/O element does not integrate terminal resistance for impedance matching, 100Ω shunt resistors are needed for each differential pair. The tolerance is ±10%.

Single-ended signal wire impedance

All the other signal wires which are not mentioned here on board have 50Ω impedance requirement.

3.4.2. PCB board stack up

Impedance Information

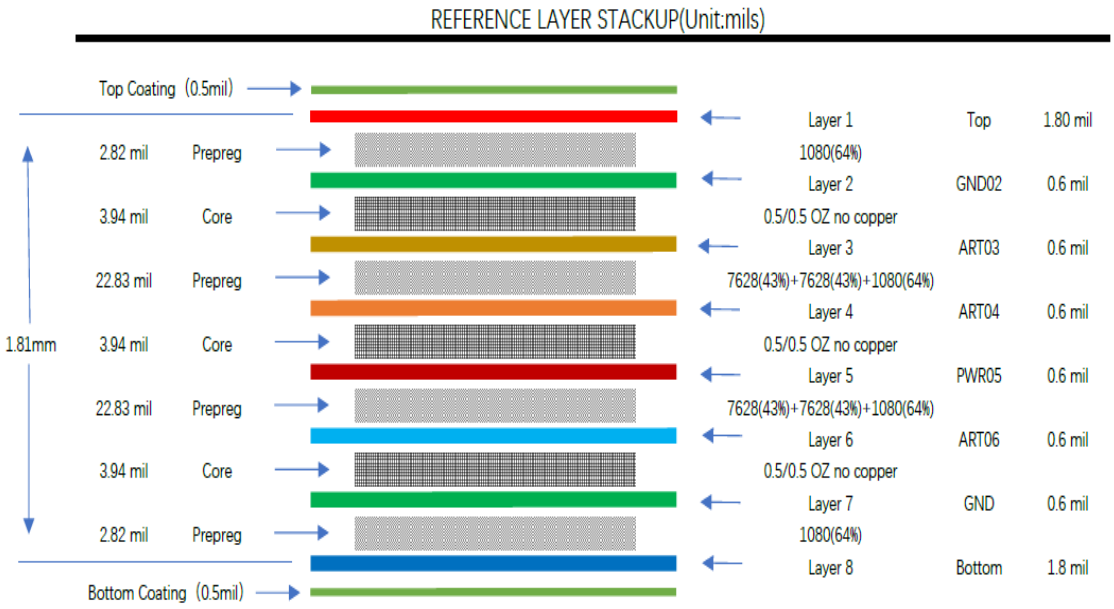


Figure 3.15. Layer Stack up.

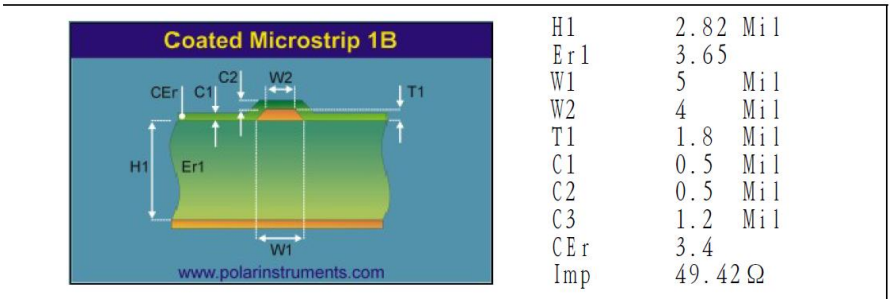
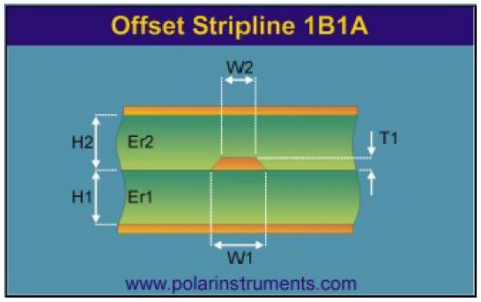


Figure 3.16. Model 1.

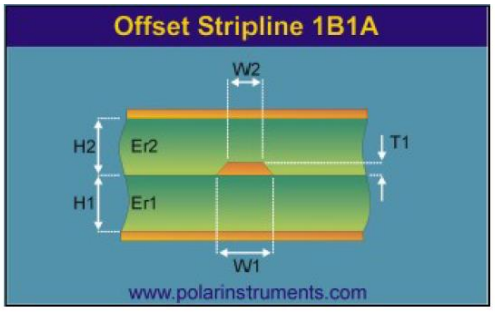
Model 1 in Fig/ 3.16 represents the top layer single-ended wire impedance calculation with one continuous ground or power plane as reference GND. C3 is the thickness solder mask.



H1	3.94 Mil
Er1	4
H2	22.83Mil
Er2	4.04
W1	5 Mil
W2	4.7 Mil
T1	0.6 Mil
Imp	51.67Ω

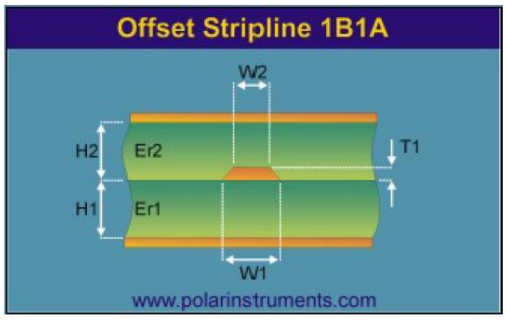
Figure 3.17. Model 2.

Model 2 represents internal layer for single-ended impedance calculation with two continuous groundes (or power) as reference ground.



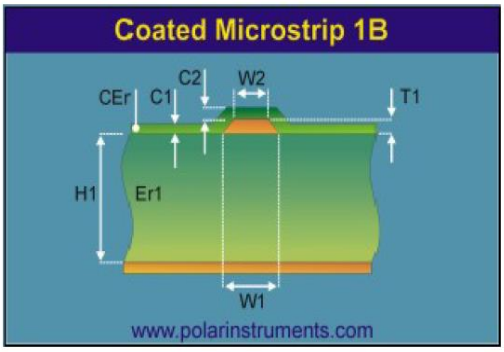
H1	3.94 Mil
Er1	4
H2	22.74Mil
Er2	4.04
W1	5 Mil
W2	4.7 Mil
T1	0.6 Mil
Imp	51.66Ω

Figure 3.18. Model 3.



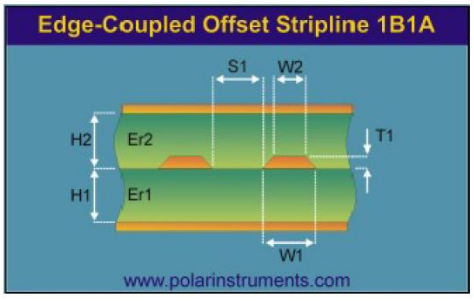
H1	3.94 Mil
Er1	4
H2	18.38Mil
Er2	4.08
W1	5 Mil
W2	4.7 Mil
T1	0.6 Mil
Imp	51.04Ω

Figure 3.19. Model 4.



H1	2.82 Mil
Er1	3.65
W1	5 Mil
W2	4 Mil
T1	1.8 Mil
C1	0.5 Mil
C2	0.5 Mil
C3	1.2 Mil
CEr	3.4
Imp	49.42Ω

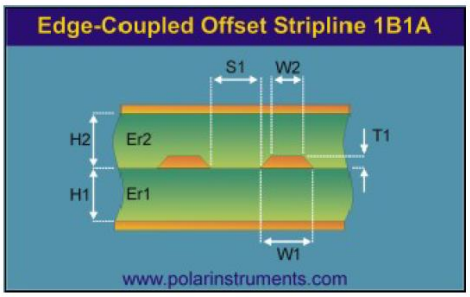
Figure 3.20. Model 5.



H1	3.94 Mil
Er1	4
H2	22.83Mil
Er2	4.04
W1	4.5 Mil
W2	4.2 Mil
S1	8.4 Mil
T1	0.6 Mil
Imp	99.1 Ω

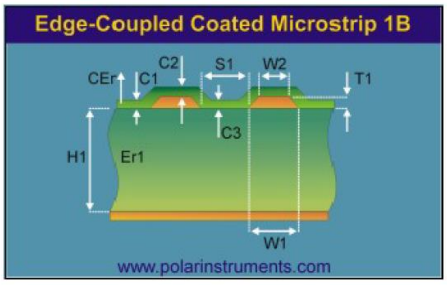
Figure 3.21. Model 6.

Model 6 represents internal layer differential pair impedance calculation with two continuous grounds (or power).



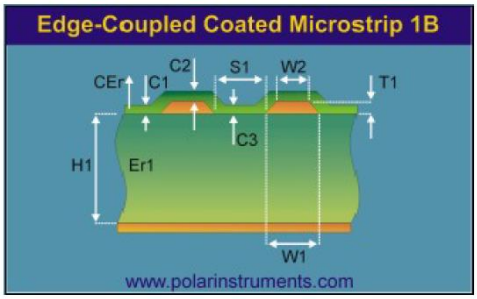
H1	3.94 Mil
Er1	4
H2	18.38Mil
Er2	4.08
W1	4.5 Mil
W2	4.2 Mil
S1	8.4 Mil
T1	0.6 Mil
Imp	98.64Ω

Figure 3.22. Model 7.



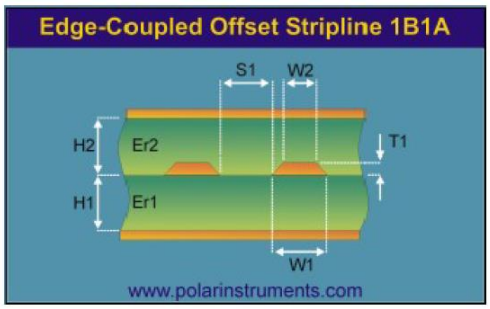
H1	2.82 Mil
Er1	3.65
W1	4.2 Mil
W2	3.2 Mil
S1	8.7 Mil
T1	1.8 Mil
C1	0.5 Mil
C2	0.5 Mil
C3	1.2 Mil
CEr	3.4
Imp	99.32Ω

Figure 3.23. Model 8.



H1	2.82 Mil
Er1	3.65
W1	5 Mil
W2	4 Mil
S1	7.6 Mil
T1	1.8 Mil
C1	0.5 Mil
C2	0.5 Mil
C3	1.2 Mil
CEr	3.4
Imp	90.32Ω

Figure 3.24. Model 9.



H1	3.94 Mil
Er1	4
H2	18.2 Mil
Er2	4.08
W1	5.3 Mil
W2	5 Mil
S1	7.3 Mil
T1	0.6 Mil
Imp	90.07Ω

Figure 3.25. Model 10.

As the models listed above, each parameter to be determined not only needs to consider the factor related to the layer itself, but also needs to consider the board thickness and as well as how much effect will be for split power and split ground. In most cases we need to balance all kinds of affected factors and try many times of calculations before it reaches our goals. Table 3.7 lists each layer's calculation result according to the proper models above.

Table 3.7. Calculation result for 8 layer models.

Number	type	Signal Layer number	Ref. layer number	Trace width(mils)	Differential pair distance	impedance
Model-1	Top layer single-end	L1	L2	5		49
Model-2	Internal layer Single-end	L3	L2/L5	5		52
Model-3	Internal layer Single-end	L4	L2/L5	5		52
Model-4	Internal layer Single-end	L6	L5/L7	5		51
Model-5	Internal layer Single-end	L8	L7	5		49
Model-6	Internal layer differential	L3	L2/L5	4.5	8.4	99
Model-7	Internal layer	L6	L5/L7	4.5	8.4	99

	differential					
Model-8	Internal layer differential	L8	L7	4.2	8.7	99
Model-9	Internal layer differential	L1	L2	5	7.6	90
Model-10	Internal layer differential	L3	L2/L5	5.3	7.3	90

Chapter 4.

FPGA Video Data Stream Processing

The video data processing block diagram is shown in Figure 4.1.

1. LVDS module: the LVDS module deals with LVDS interface, utilizes SERDES MEGA function of Altera IP to convert serial data stream to parallel data. Here we combine 2 channels together and realize 10-bit width parallel data.

2. Video data process: this module mainly synchronizes the received parallel data and writes to dual port memory in the specific format.

3. Dual port memory: since for video sensor IMX226, 10 channels are used for high speed data transmit, each memory is used as data buffer for 2 channels. 10 channels totally need 5 memory.

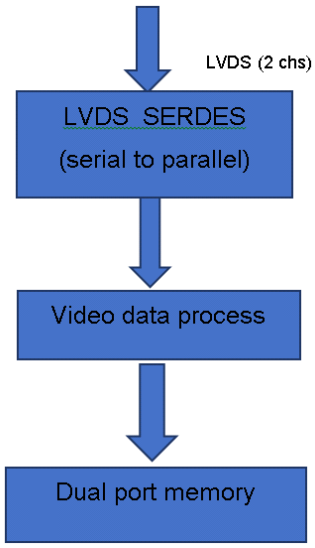


Figure 4.1 . Data processing block diagram.

4.1. Clock generation

Each video sensor needs input clock with specific frequency value. This clock is used to drive video data acquisition, video data processing and video data serialization as well. After clock is locked, the sensor also provides an output clock for video data synchronization, as shown in Figure 4.2.

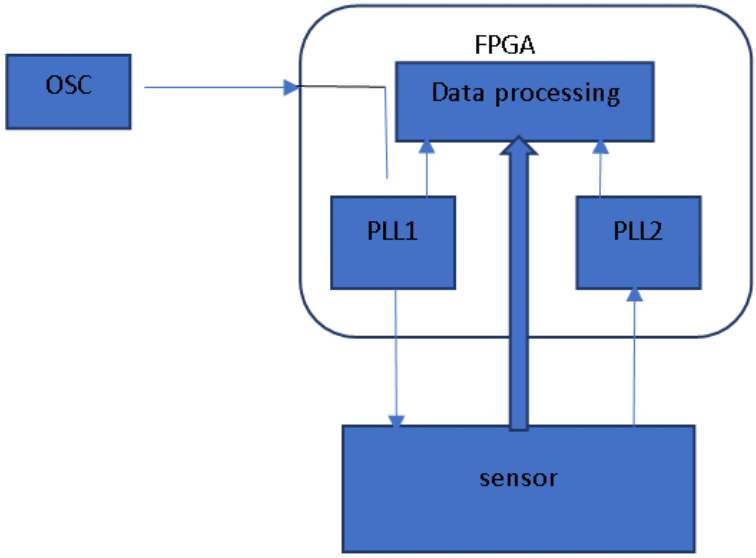


Figure 4.2. Camera clock generation system.

4.2. Serial to parallel conversion and SERDES synchronization

4.2.1. Sensor serial data stream structure

For different sensors and different resolutions, the data stream structures are different. Here we use IMX226 as an example to introduce how the serial data are organized together in different readout drive modes.

Sensor readout drive mode

For IMX226 there are totally 6 readout drive modes. Different readout drive modes need specific control register settings. Even when the readout mode is the same, MDVREV can control the readout sequence. For example, MDVREV bit0=0h/1h will control normal or inversed vertical reading operation. All readout modes are listed in Table 4.1.

Table 4.1. Readout drive mode [1].

Readout mode NO.	Readout drive mode	Mode description
0	All pixel scan mode (12M, A/D 12-bit,10 ch output)	ALL pixel with 12-bit out
1	All pixel scan mode (12M, A/D 10-bit,10 ch output)	ALL pixel with 10-bit out
2	All pixel scan mode (4K2K, A/D 12-bit, 8 ch output)	ALL pixel with 12-bit out
3	All pixel scan mode (12M, A/D 12-bit,4 ch output)	ALL pixel with 12-bit out
4	All pixel scan mode (12M, A/D 10-bit,10 ch output)	ALL pixel with 10-bit out
5	Horizontal/vertical 2/2-line (4K2K, A/D 10-bit,4 ch output)	Horizontal and vertical direction 2-line binning

Fig. 4.3 shows the sync signal and data timing during 12-bit serial output for sensor IMX226. The horizontal and vertical timings of the output data are controlled by XVS and XHS sync signals. Timing control is performed at the falling edge of both the XVS and XHS signals. The data are output in order, from the start sync code (SAV) after the horizontal front blanking period after the falling edge.

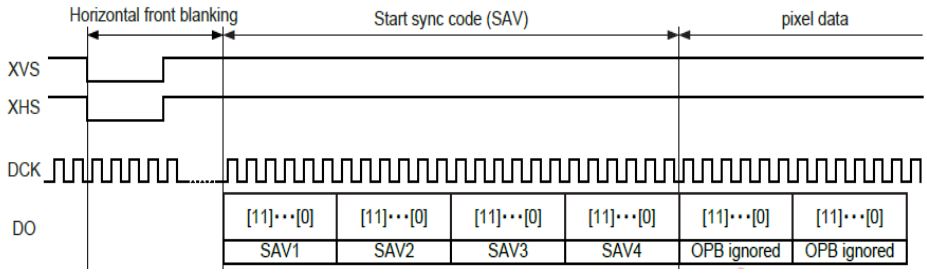


Figure 4.3. Sync signal and XVS, XHS timing (start sync) [1].

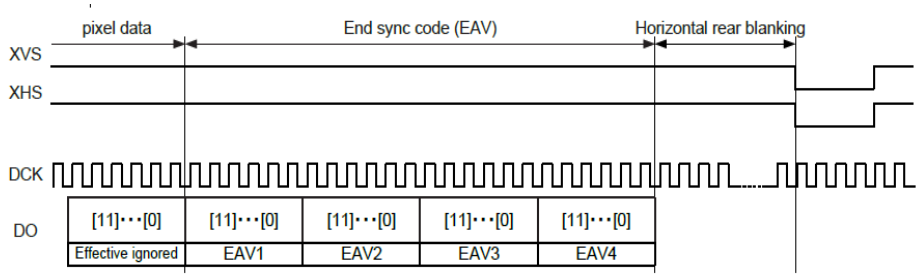


Figure 4.4. Sync signal and XVS, XHS timing (end sync) [1].

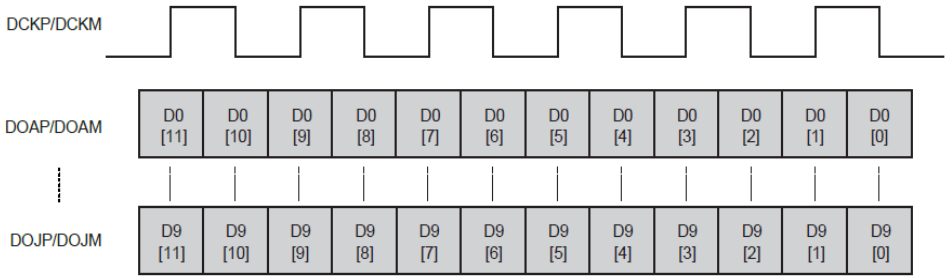


Figure 4.5. Serial data timing. [1].

From Figure 4.4 we can see that serial data are aligned as 180 degree at each clock edge. This gives better margin for setup time and hold time, even when the clock edge is shifted because of jitter or delay caused by unequal length in layout [1].

Table 4.2 gives more detail on start sync code and end sync code.

Sync code details

LVDS output bit No.		Sync code (4 words)				
12-bit output	10-bit output	1st word	2nd word	3rd word	4th word	
11	9	1	0	0	1	
10	8	1	0	0	0	
9	7	1	0	0	V	1: Blanking line 0: Except blanking line
8	6	1	0	0	H	1: End sync code 0: Start sync code
7	5	1	0	0	P3	Protection bits
6	4	1	0	0	P2	
5	3	1	0	0	P1	
4	2	1	0	0	P0	
3	1	1	0	0	0	
2	0	1	0	0	0	
1	—	1	0	0	0	
0	—	1	0	0	0	

Table 4.2. sync code format [1].

		Protection bits			
V	H	P3	P2	P1	P0
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

Table 4.3. sync code protection bit [1].

		1st word	2nd word	3rd word	4th word
Blanking line	Start sync code (SAV)	FFFh	000h	000h	AB0h
	End sync code (EAV)				B60h
Except blanking line	Start sync code (SAV)				800h
	End sync code (EAV)				9D0h

Table 4.4. Table4.4 12-bit sync code detail [1].

Table 4.5. 10-bit code detail [1] .

		1st word	2nd word	3rd word	4th word
Blanking line	Start sync code (SAV)	3FFh	000h	000h	2 ACh
	End sync code (EAV)				2D8h
Except blanking line	Start sync code (SAV)				200h
	End sync code (EAV)				274h

4.2.2. Serial to parallel SERDES module

The serial to parallel conversion module is described as follows in Verilog format. The module name is “lvds_rx”, and the instance name is” lvds_core_inst1”.

```
lvds_rx lvds_core_inst1 (
. rx_data_reset              (rst),
.rx_channel_data_align       (rx_channel_data_align[1:0]),
.rx_in                       (MIPI_DP[1:0]),
.rx_inclock                  (MIPI_CLK),
.rx_out                      ({imx_out1,imx_out0})
);
```

“MIPI_DP [1:0]”: 2-channel serial video data input which follows LVDS format. In Quartus II environment, for LVDS interface we only need to provide positive data path like MIPI_DP, and the negative path will be derived by Quartus.

“rx_channel_data_align”: bit shifting control signal for alignment

“MIPI_CLK” is the clock generated by PLL2, which is sourced from clock generated by sensor.

“rx_out” is 20-bit video data. The lower 10 bits belong to channel 1, and the higher 10 bits belong to channel 2.

4.2.3. SERDES data synchronization

The data stream structure will be different when different sensors are connected, and even when the same sensor is used with different video resolutions.

According to table 4.3 to table 4.5, module “lvds_rx “of FPGA code first waits for “SAV” to follow the start of sync, then checks if “EAV” appears just before H-sync signal. If the sequence of “SAV” marker is followed by data then followed by “EAV” marker, the sync signal will be asserted. See Fig.4.6.

4.3. FPGA code running results

The figures below are the running results of each algorithm introduced above. The picture is captured from SignalTap II Logic Analyzer, which is fully integrated into Quartus II.

4.3.1. line_start [0] set up under condition of XVS equal to high

Line_start [0] is set up after 0_SENSOR_HD (representing XHS) changes from low to high, since in SignalTap II, we cannot show H and SAV together (limited by memory occupied), Figure 4.6 shows how line_start [0] changes based on SAV.

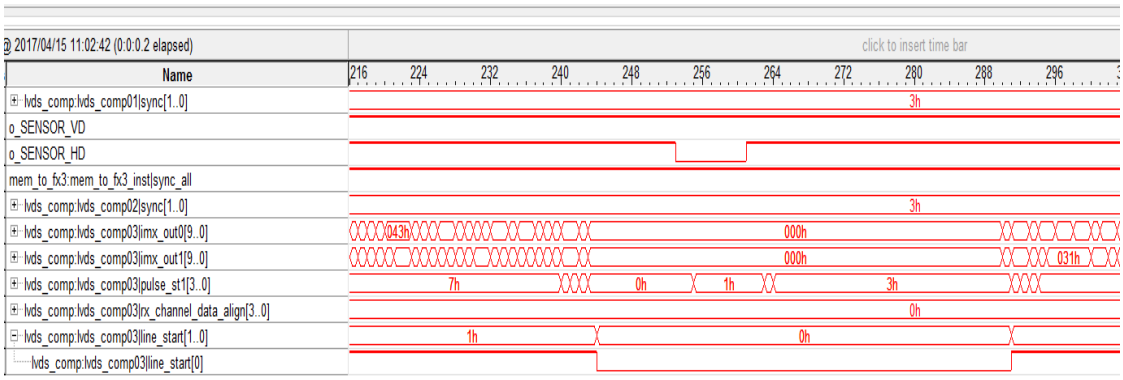


Figure 4.6. line_start [0] with relationship of XHS signal(XVS=1).

Figure 4.7 shows line_start [0] changes status after successfully detecting the correct SAV sequence. Sync [0] status is set up according to the correctly detected SAV sequence, then based on the Sync [0] signal and XHS jumping from low to high, line_start [0] signal will be correctly set.

Figure 4.7. line_start [0] flip after detecting first word of SAV(XVS=1).

line_start [0] is set up under condition of XVS equaling to high. In Figure 4.8, o_SENSOR_VD, o_SENSOR_HD are the V and H signals provided by FPGA to sensor for horizontal and vertical sync. After low pulse of V-sync signal, the SAV (start sync code) is different from non-blanking horizontal line. Compared to Figure 4.7 and Figure 4.8 we can see clearly that when the SAV is in non-blanking stage, the value is ”3FF 000 000 200”, yet when in blanking stage, it becomes “3FF 000 000 2AC”.

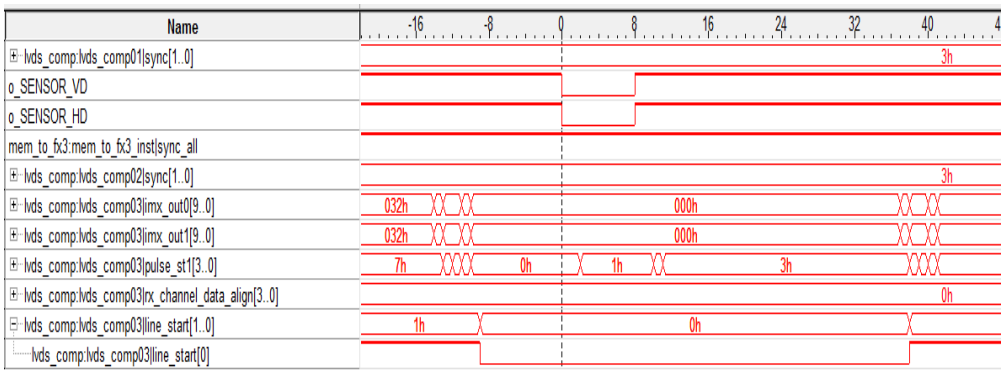


Figure 4.8. line_start set up in V_sync stage.

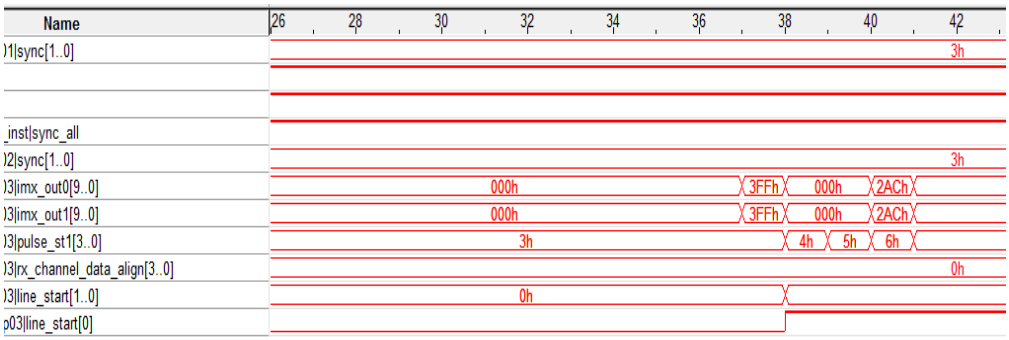


Figure 4.9. line_start [0] flip after detecting first word of SAV(XVS=0).

Line stop detecting result is shown in Fig. 4.10 under the condition of non-blanking(vertical). Here we use line_start [0] =0 instead of line_stop [0] =1, which will be more convenient in memory write control process. Figure 4.10 shows line_start [0] falls to 0, under the condition of non-blanking stage (vertical).

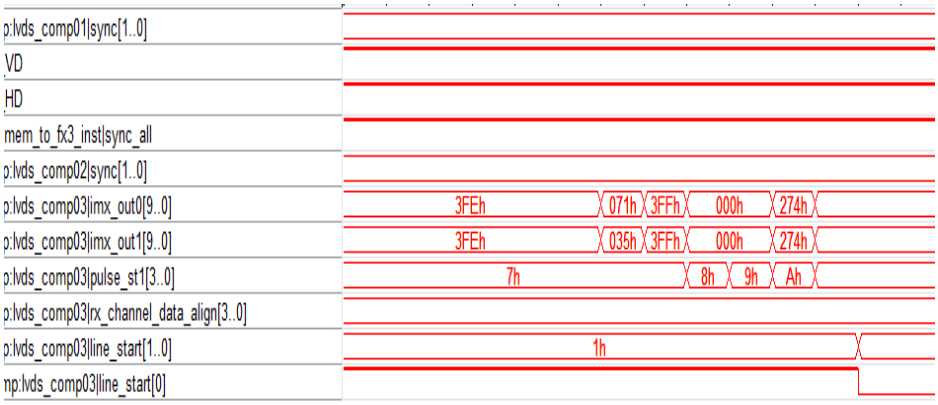


Figure 4.10. line_start [0] falls to low after correctly detecting EAV (end of sync coding).

Figure 4.11 shows line_start [0] falls to 0, under the condition of blanking stage(vertical).

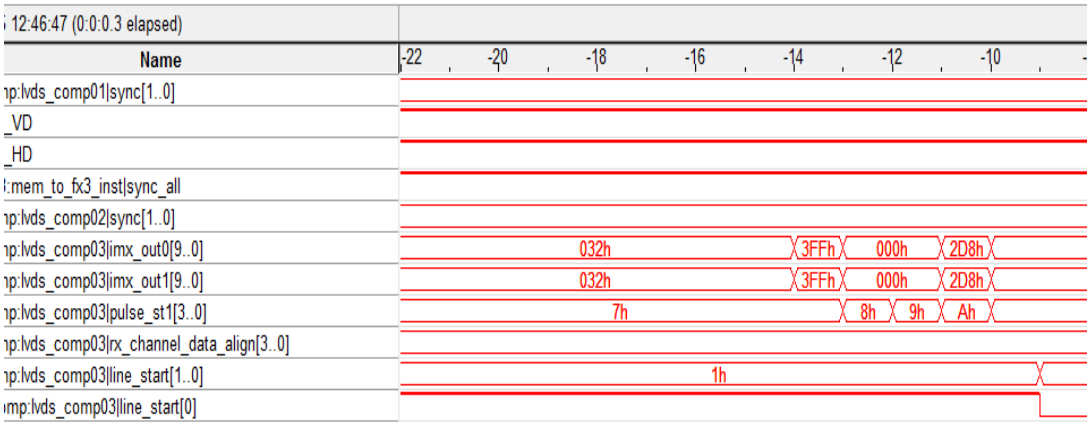


Figure 4.11. line_start [0] falls to low after correctly detecting EAV (end of sync coding).

4.4. Store dual channel data to FIFO memory

4.4.1. FIFO buffer interface description

The instant of module LVDS_to_buf is listed below.

```
lvds_to_buf   lvds_to_buf0 (
.rst(rst),
.rx_outclk(rx_outclk),
.mem_rd_clk(mem_rd_clk),
.rd_req(rd_req[0]),
.H(H),
.V(V),
.line_data_start(line_start[0]),
.sync_all(sync_all),
.imx_out0(imx_out0x),
.imx_out1(imx_out1x),
.mem_ep(mem_ep[0]),
.imx_buf_out(imx_buf_out0)
);
```

where

1. rx_outclk: LVDS parallel clock signal, the frequency of rx_outclk depends on LVDS input frequency and deserialization factor. For example, if input LVDS frequency is 100MHz DDR and the deserialization factor is 10, then the output frequency is 20MHz.
2. mem_rd_clk: FIFO read clock. This clock will be described later in “data collection” part.

3. Rd_req: memory read enable signal.
4. Sync_all: multi-channel sync signal. combine all channel together. If every active channel is synchronized, sync_all signal will be high.
5. imx_out0x, imx_out1x: one latency from imx_out0 and imx_out1. For sync data and clock alignment.
6. imx_buf_out: 32-bit memory output data, since memory structure is 16-bit in, 32-bit out FIFO.

4.4.2. Memory write process

The memory write process is as follows.

1. initialization, set initial variable value
2. waiting for Sync_all=1;
3. start from V=0;
4. Wait for rising edge of V;
5. Waiting for line_start=1;
6. Write data to memory,

   Write_req=1;

   imx_buf_in={{6'b0,imx_out1},{6'b0,imx_out0}}; // align two 10-bit to 32-bit

   //here the format is based on IMX226 10-bit structure, if ADC acquisition width is 12 //bit then imx_buf_in will be

   imx_buf_in= {{4'b0,imx_out1}, {4'b0,imx_out0}};

   //If line_start=0, go to next step, otherwise keep in this step for continuing writing.
7. Writing_end step, when line_start=0, then write 8-12 of 32-bit 0's for line separation.
8. If sync_all=1, go to step (3), otherwise go back to initialization step.

The results are shown in fig 4.12 to 4.14.

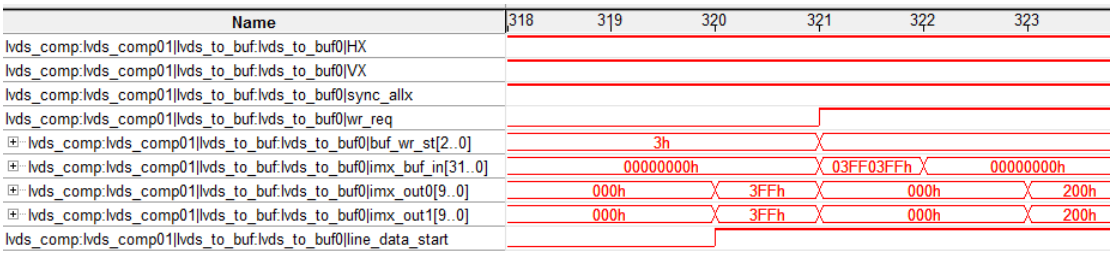


Figure 4.12 Memory write process.

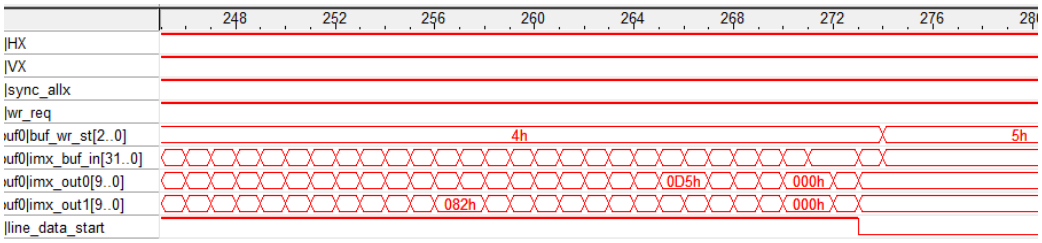


Figure 4.13. Figure 4.12: Line data writing start.

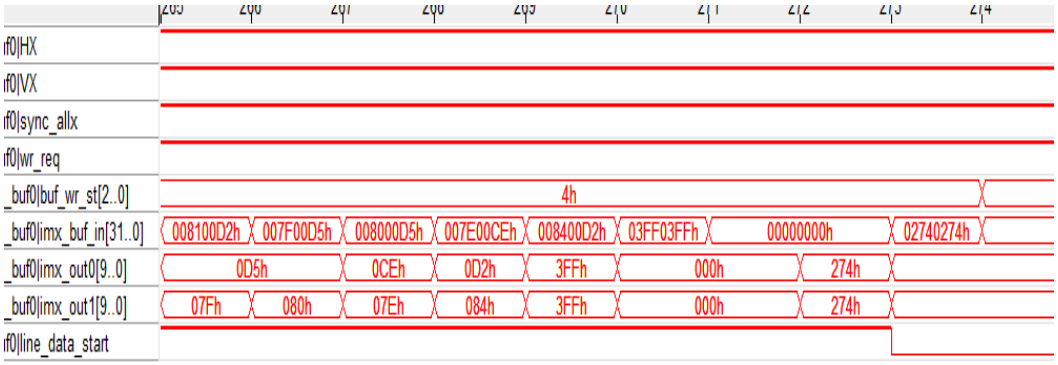


Figure 4.14. Line data write stop control signal.

4.5. Multi channel data collection

IMX226 has 10 channels. Each channel is 10-bit wide, and 2 channels together occupy a FIFO. In order to write all data to EZ-FX3 GPIF II interface, 10 channel data have to be collected together and well-arranged according to incoming data sequence.

4.5.1. Data sequence received from sensor

Different configurations will have different active channels, and the data output from sensor also will have different sequence. The figure below shows one of the configurations for IMX226. The 10 channel numbers are DOA to DOJ. Data scan sequence is column by column, and from left to right until the last column is completed. i.e., from DOA SAV1 to DOJ SAV1, then DOA SAV2 to DOJ SAV2…. etc.

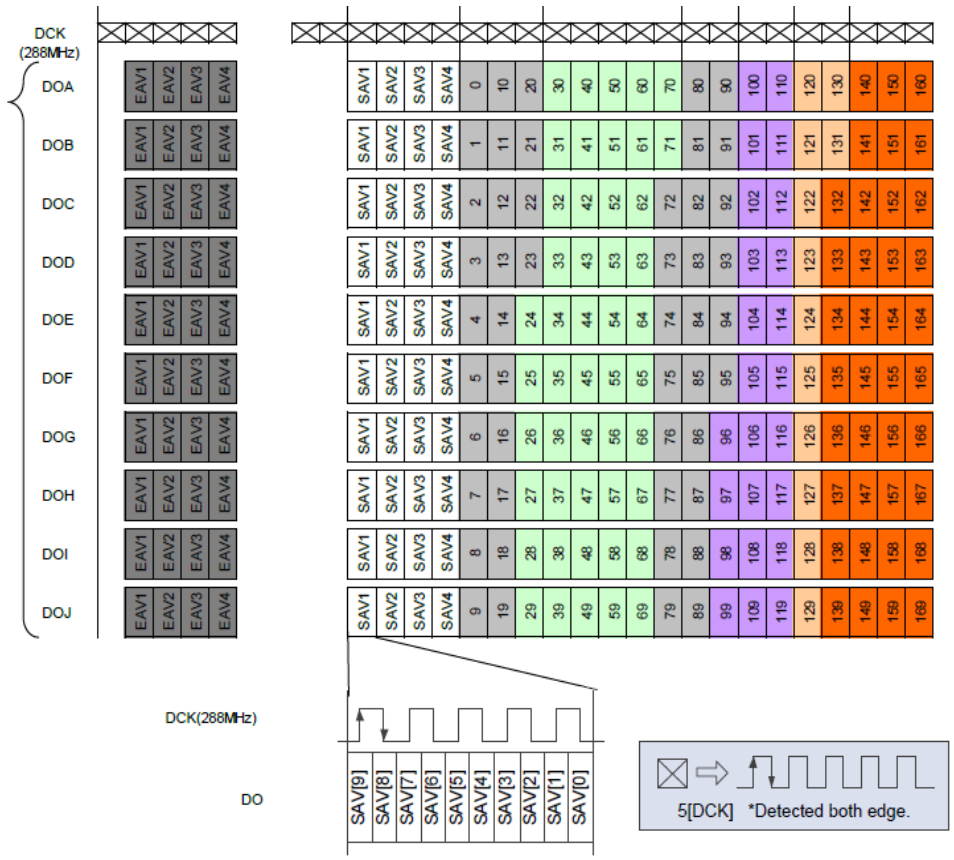


Figure 4.15. 10 channel data Readout sequence start part [1].

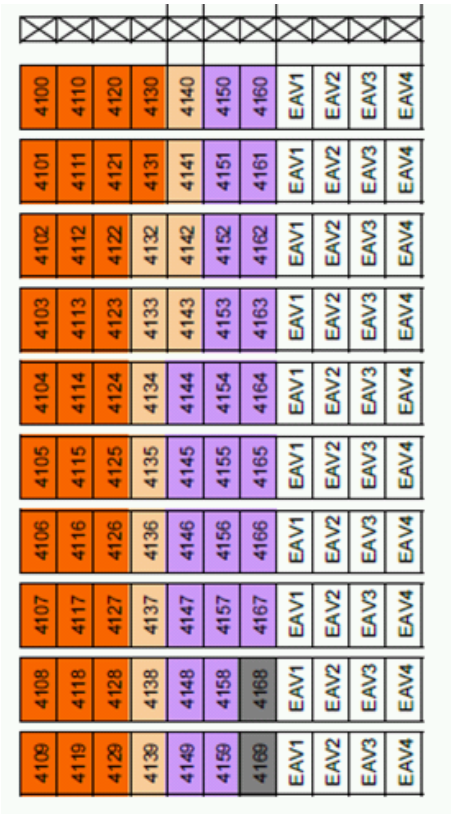


Figure 4.16. 10 channel data Readout sequence end part [1].

4.5.2. FPGA data collection block diagram

The data collection block is to collecting multiple channels stream data sequentially to one side of dual port memory, and on the other side of dual port memory, the parallel data can be read out. For writing stage, there will be two processes need to deal with, the data start writing stage and the data stop writing stage.

Data start writing

1. Initialization, variable initialize as following,
2. Wait for all active memory is filled with data.
3. Scan each active memory until captured each SAV from each memory.
4. Write one SAV to destination memory, then begin to write each data read from channel memory.

The procedure is shown in fig. 4.17 and 4.18.

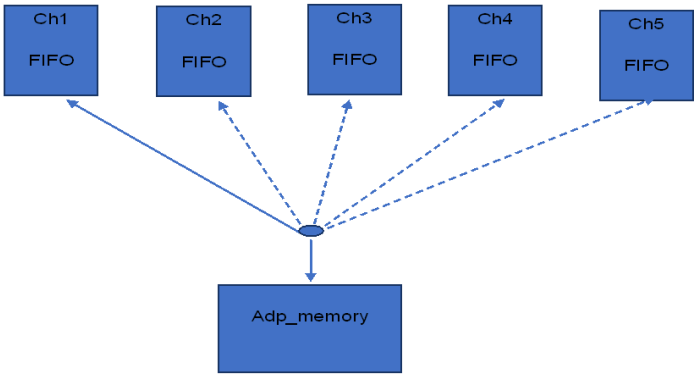


Figure 4.17. Schedule reading from channel memory to adaptive memory.

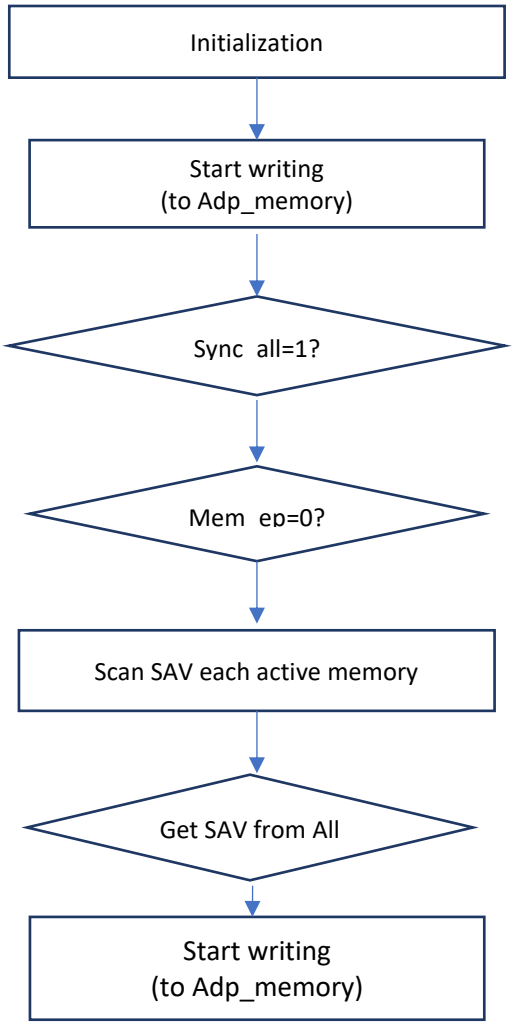


Figure 4.18. Collection process data writing.

Stop data writing

1. Detect each channels EAV(10h3ff,10'h000,10'h000,10'h274(or 10'h2AC).
2. Write single EAV to memory (64'h03ff_0000_0000_03ff).
3. Write several 64-bit 0 to separate different frame.

The procedure is shown in Fig. 4.19.

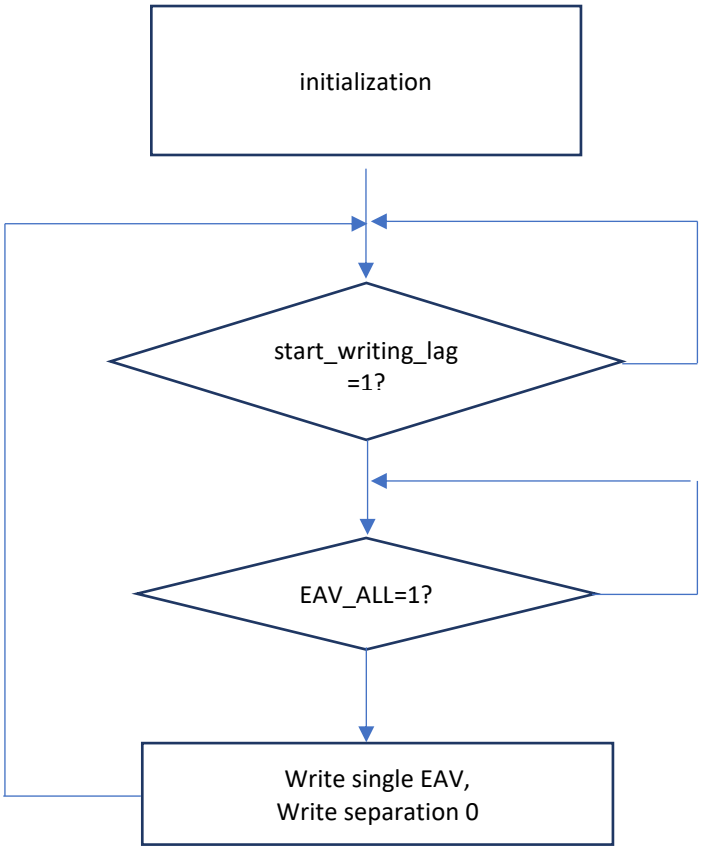


Figure 4.19. Stop writing detect.

Running results

The figures below are captured from Quartus II signal tap II. From first SAV writing to second word writing, it will last more than 10 clocks. In order to show data clearly, we use multiple pictures to show the results.

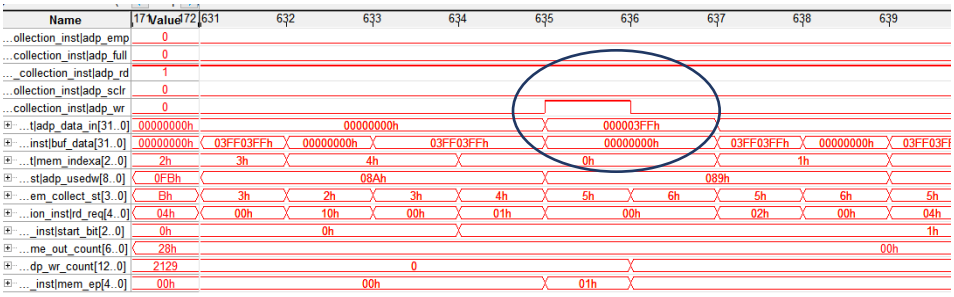


Figure 4.20. First SAV word, write to dp_memory.

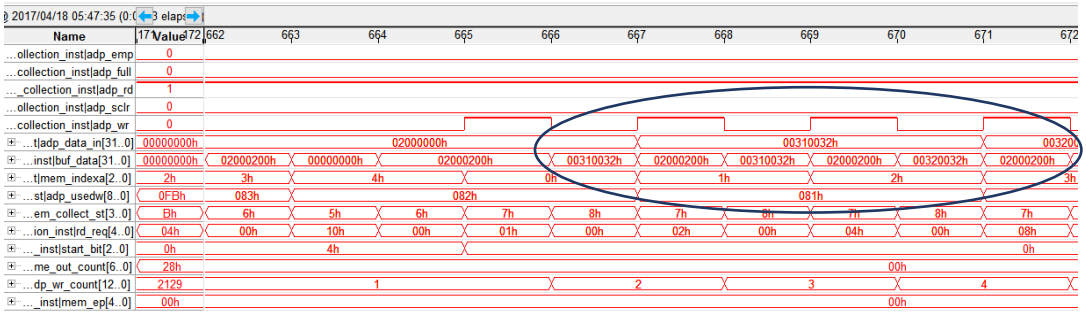


Figure 4.21. Second SAV word and the following data write to adp_memory.

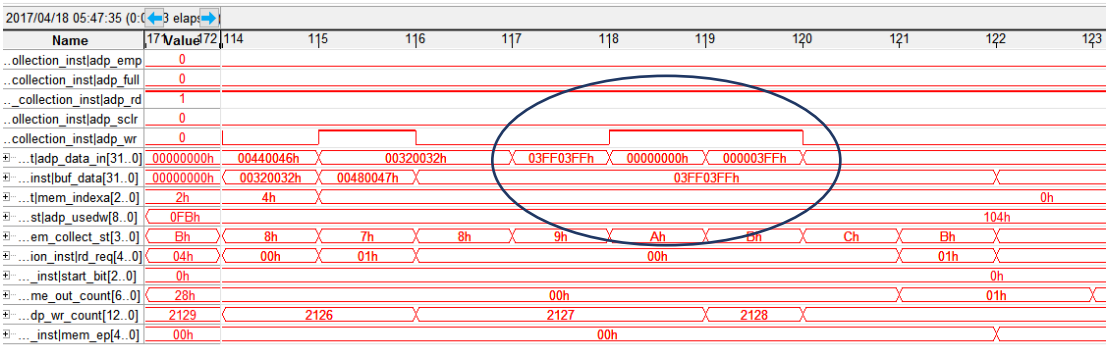


Figure 4.22. Stop writing sequence I: Write first EAV word.

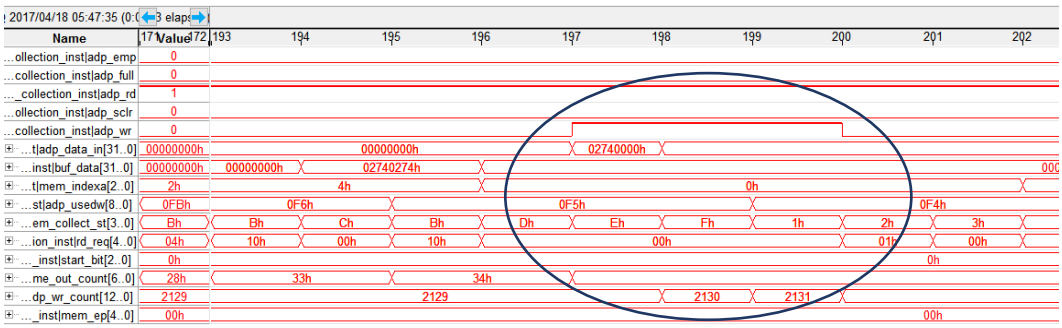


Figure 4.23. Stop writing sequence II": Last EAV writing, and write two 32-bit "0000_0000" for separation.

Chapter 5.

Color Correction from FPGA to EZ-FX3 GPIFII Interface

5.1. Introduction to GPIF II

The FX3 GPIF II subsystem is a programmable state machine that allows us to implement an industry standard or a proprietary interface. It is quite easy to use with the help of GPIF II designer tools.

5.2. Parallel video interface

We can easily create a parallel video interface by using GPIF II designer. Figure 5.1 gives the interconnection between FPGA and FX3 for parallel video data path. The brief description for signals between FPGA and FX3 are as follows:

1. Data [31:0], 32-bit video data. This 32-bit data come from video sensor, and are collected and re-organized by the algorithm running inside FPGA. The 32-bit data structure for video will provide several ways of video data organization as showed below.
2. 8-bit pixel data: Each pixel data width is 8-bit, so 32-bit can transmit 4 pixels each time. The Data structure is data [31:0] = {pixela [7:0], pixelb [7:0], pixelc [7:0], pixeld [7:0]}.
3. 10-bit pixel structure: Each video pixel is 10-bit. For 32-bit structure, the easiest way to organize the data within 32-bit is to divide one 32-bit to two 16-bits, lower 16-bit data [15:0] hold one pixel, and the high 16-bit hold another pixel. The unused bit will be filled with 0's. The data structure is data [31:0] = {6'h00, pixela [9:0], 6'h00, pixelb [9:0]}.
4. 12-bit pixel structure: same as above. The data structure as data [31:0] = {4'h0, pixela [11:0], 4'h0, pixelb [11:0]}.
5. Other structures: There are still other structures have not been discussed above, like 32-bit hold the 10-bit width pixels, such as data [31:0] = {2'b00, pixela [9:0], pixelb [9:0], pixelc [9:0]}.
6. Pclk: this is the clock signal generated by PLL inside FPGA, which is synchronized with video data and H, V.
7. H: horizontal video data synchronization signal.
8. V: vertical video data synchronization signal. The data sent through FPGA module will only be captured by FX3 under condition that both H, V signal are high.

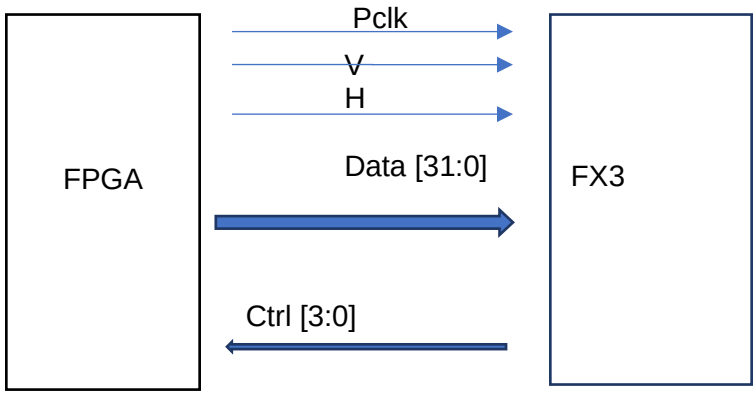


Figure 5.1. Video data interface between FPGA FX3.

5.3. Data re-organization

For the sensor IMX226, the channel output format is 10-bit or 12-bit wide. To adapt to 32-bit wide interface, we can choose two 12-bit wide pixels for 32-bit, or two 10-bit structure, or scale down to 8-bit structure. In the latter case, the 32-bit FX3 interface is able to hold 4 pixels in a single period. From chapter 4, we already collect multi-channel video data and write to a single FIFO. The structure is 32-bit in and 64-bit out. Besides, the read and write clocks are independent.

5.4. Signal and register definition

Here we use fu, ep, en, clk to represent full, empty, enable and clock for FIFO control and status signals, and define the corresponding register for each side.

(1) 32-bit side, video data write in side. (please refer to chapter 4)

adp_full	-----FIFO full
adp_wr	-----FIFO write enable
adp_usedw	------FIFO used words (how many 32-bit words store in FIFO)

(2) 64-side, video data

adp_emp	-----FIFO empty status
adp_rd	-----FIFO read enable
o_fx3_dq_tmp	-----FIFO data output(data_out[63:0])

(3) FX3 side,

o_fx3_v -----video vertical sync signal
o_fx3_h ------video horizontal sync signal
o_fx3_dq -------video data from FPGA to FX3.

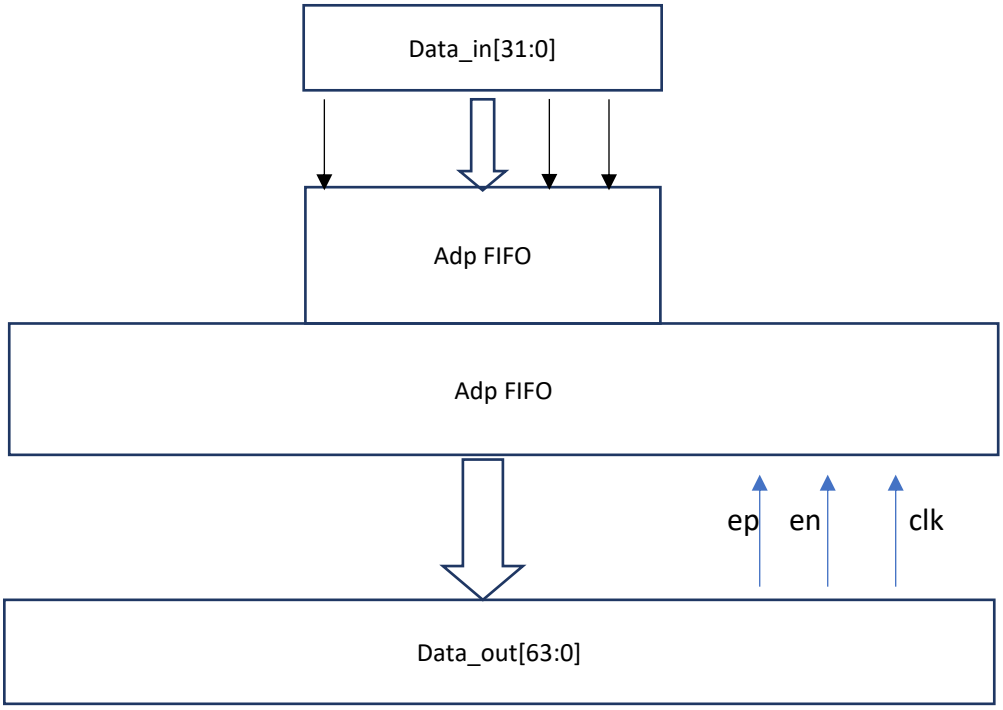


Figure 5.2. Adaptive FIFO structure.

5.4.1. Video data re-organization diagram

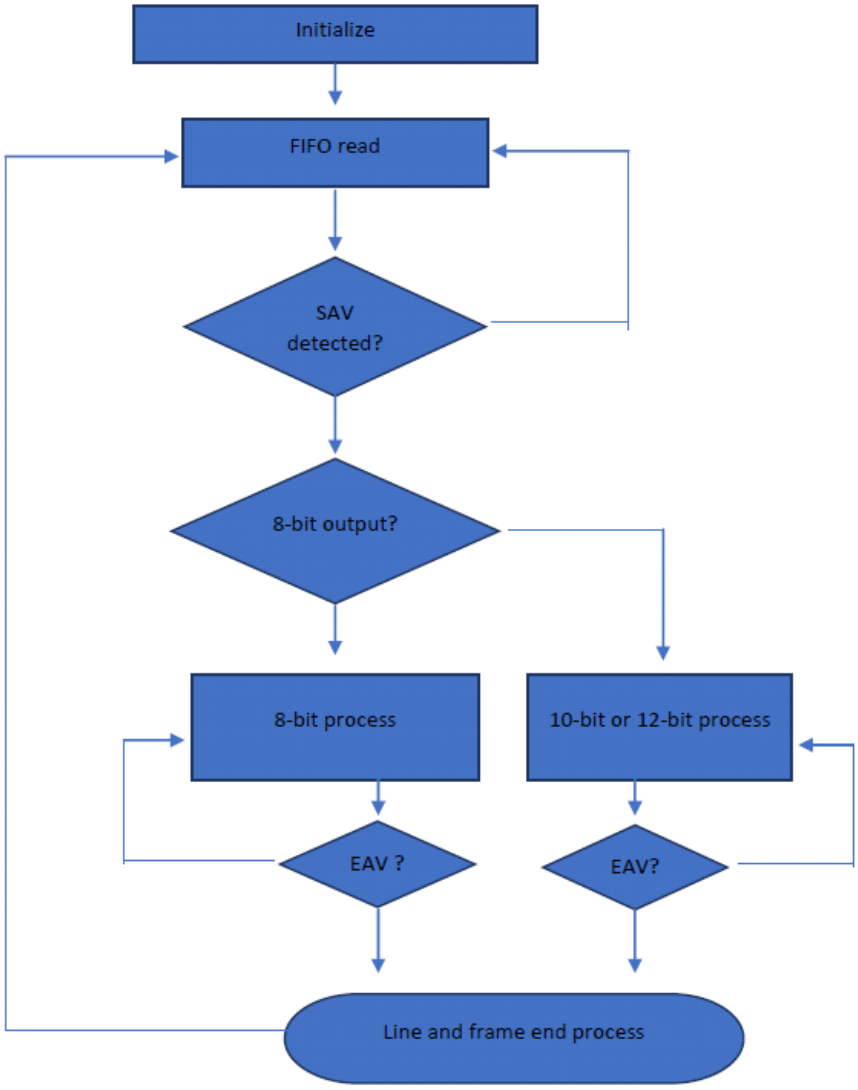


Figure 5.3. Video data re-organize block diagram.

5.4.2. FX3 video input data format

To facilitate the software to deal with the received video data, the FPGA to FX3 interface organizes data as the following.

(1) Frame start, when both V and H sync change from low to high, this marks the first line of the whole frame video data. Before the real video data is sent to FX3, the formatted data frame start “c0,0c,55,55” and frame number are sent to FX3 first.

(2) If it is not the frame start line, these two 32-bit words will be padding with 0’s.

(3) After frame start is sent, the real video data will follow, which is surrounded by SAV and EAV, as showed in Figure 5.4.

Figure 5.4. FX3 video input data format.

5.4.3. White balance algorithm

White balance is very important for improving video quality, and there are different algorithms in different situations. The purpose to move the algorithm from software to FPGA is to speed up software calculation. Here we just introduce one of the

algorithms, which is used in normal circumstance. According to the white balance formula, we divide 3 steps to achieve the result.

- Average color R, G, B within one frame, V_R, V_G, V_B.
- Calculate α, β, ϒ.
- Multiply α, β, ϒ to corresponding pixels of next frame.

Since the sensor data readout is in Bayer mode (which is different from RGB separation mode), to deal with the color average, the critical things is to find out the correct color sequence for specific sensor. The table below is the IMX226 color coding sequence. In different sensor Bayer modes, color coding may be different. So when performing white balance, this needs to be paid attention to.

Table 5.1. Bayer pattern in sensor.

Gb	B	Gb	B
R	Gr	R	Gr
Gb	B	Gb	B
R	Gr	R	Gr

In table 5.1, Gb represents color G (green) in B line, B is blue, R is red, Gr is green in R line. From the table above, we can see that in even line and even column, the corresponding color is Gb (green). In even line odd column the color is B. In odd line and odd column the color is Gr (green too). In odd line and even column the color is R (red). In order to calculate the average of each color, we define 2 variables p_num and l_num. p_num is the pixel number counter in one video line, and l_num is another counter for video line in one frame. For mode 4 (here we choose mode 4 for as example), l_num starts from pixel 124, and stops at 4220; p_num starts from line 18, and stops at line 2178. See table 5.2 and table 5.3.

Table 5.2. Minimum horizontal operation period in each Readout drive mode [1].

Readout mode No.	Data rate [MHz]	Horizontal front blanking [DCK] *2 *3	Minimum horizontal operation period (Number of pixels conversion)					XHS minimum period [INCK]*2
			Front OPB	Front effective pixel margin	Recommended recording pixels	Rear effective pixel margin	Rear OPB	
0	576 (288DDR*1)	348 to 353	96	36	4000	36	0	644
1	576 (288DDR*1)	348 to 353	96	36	4000	36	0	536
2	576 (288DDR*1)	348 to 353	96	28	4096	28	0	811
3	576 (288DDR*1)	348 to 353	96	28	4096	28	0	1608
4	576 (288DDR*1)	348 to 353	96	28	4096	28	0	546
5	576 (288DDR*1)	354 to 359	48	14	2048	14	0	407

Table 5.3. Minimum Vertical Operation Period in Each Readout Drive Mode [1] .

Readout mode No.	Number of lines per minimum vertical operation period (output data 1H conversion)						XVS minimum period [XHS]
	Vertical front blanking	Front OPB	Front effective pixel margin	Recommended recording pixels	Rear effective pixel margin	Rear OPB	
0	17*1 16*2	16	24	3000	22	0	3079
1	17*1 16*2	16	24	3000	22	0	3079
2	17*1 16*2	8	10	2160	4	0	2199
3	17*1 16*2	8	10	2160	4	0	2199
4	17*1 16*2	8	10	2160	4	0	2199
5	9 (20XHS)	4	10	1080	4	0	2216

To correctly calculate the color average for the whole frame, we need to know the recommended recording pixels in one horizontal line and recommended recording line in vertical direction within one frame. For IMX226 mode 4 as showed in table 5.2 and 5.3, the start point in horizontal line is 124, the end pixel point is 124+4096, and the number of recommended pixels is 4096 in one line. The vertical start line is 18, the line is 18+2160, and total recommended line is 2160.

Variable definitions:

- pixel_num, pixel counter within one line to position the start and end pixel for calculation.
- line_num, video line counter within one frame to position the start and end line for white balance calculation.

- pixel_start, pixel end, start and end flag.
- line_start, line_end. Line start and end flag
- Sum_r, Sum_b, Sum_g. Color R, G, B summation within one frame.
- Ave_r, Ave_g, Ave_b. Result of color R, G, B average.
- SumRGB, AveRGB, mean of Ave_r, Ave_g, Ave_b.
    - AveRGB= SumRGB/3
    - SumRGB = (Ave_r+ Ave_g+ Ave_b);
- Kr= AveRGB/Ave_r; Kg=AveRGB/Ave_g; Kb=AveRGB/Ave_b;

According to table 5.1, 5.2, 5.3, we can calculate the corresponding summation of R, G, B, respectively, as Sum_r, Sum_b, Sum_g.

Ave_r= Sum_r/Num_r; Ave_g =Sum_b/Num_g; Ave_g=Sum_b/Num_b;

SumRGB = (Ave_r+ Ave_g+ Ave_b);

AveRGB=SumRGB/3.

Kr= AveRGB/Ave_r=(Num_r*SumRGB)/(3*Sum_r);

Kg= AveRGB/Ave_g=(Num_g*SumRGB)/(3*Sum_g);

Kb= AveRGB/Ave_b=(Num_b*SumRGB)/(3*Sum_b);

Kr, Kg, Kb multiply back the pixel in the next frame.

Results

Figure 5.5 and Figure 5.6 are the real pictures which captured by our USB 3.0 machine vision camera through different sensors. One is captured by gray-scale sensor IMX273 and another one is captured by color sensor IMX226. From the pictures, we can see that the hardware quality is satisfactory, and there is no ripple and no noise.

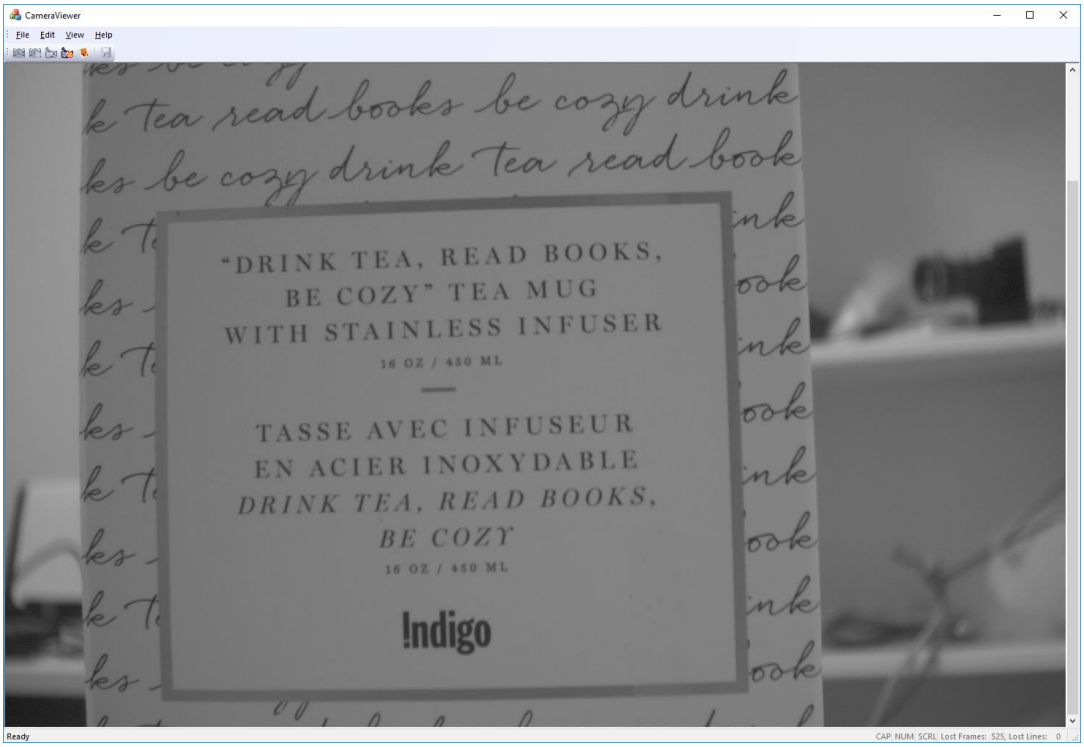


Figure 5.5. Black/white picture captured through IMX273.

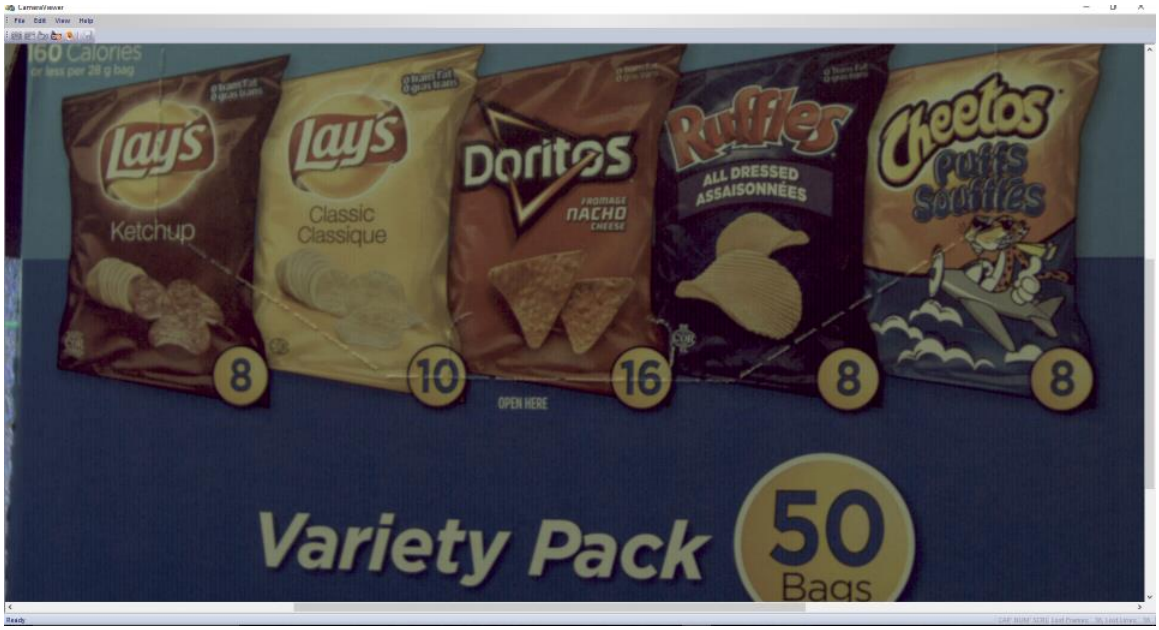


Figure 5.6. Color picture captured from IMX226.

Chapter 6.

Conclusion and Future Work

In this project, the hardware and FPGA modules of a USB 3.0 machine vision camera were successfully designed, from which deep understandings of the USB 3.0 protocol, Cypress FX3 processor, SONY image sensors, and image processing are obtained. I also got a chance to learn some mechanical design and gained valuable experience in how to design a robust camera product, for example, how to do the PCB layout, trace length, impedance matching, and power supply design etc.

There are many ways that the camera can be improved, for example, supporting more ADC resolutions and more sensors, reducing the size of the camera, and improving the image quality. The experience I learned from this project has provided a solid foundation for my future works.

References

1. IMX226CQJ-C-(E) Data_Sheet_E146A48, Sony.
2. IMX226 support package, Sony.
3. IMX250_252_264_265_PCB Design Guide, Sony.
4. IMX250_252_264_265_Support_Package_E_Rev2.0, Sony.
5. IMX273_287_Support_Package_E_Rev0.1, Sony.
6. IMX273LLR-C_TechnicalDatasheet_E_Rev0.2, Sony.
7. Cyusb301x, Cypress.
8. AN75705 Getting Started with EZ-USB FX3, http://www.cypress.com/file/139296/download
9. AN75779 How to Implement an Image Sensor Interface Using EZ-USB® FX3™ in a USB Video Class (UVC) Framework http://www.cypress.com/file/123501/download
10. CYUSB3KIT-003 SuperSpeed Explorer Kit, April 2017. http://www.cypress.com/file/134006
11. Designing with the EZ-USB® FX3™ Slave FIFO Interface http://www.cypress.com/file/136056/download
12. AN70707 EZ-USB® FX3™/FX3S™ Hardware Design Guidelines and Schematic Checklist http://www.cypress.com/file/139936/download
13. EZ-USB® FX3 Technical Reference Manual
14. AN76405 - EZ-USB® FX3™/FX3S™ Boot Options http://www.cypress.com/file/201991/download
15. FPGA cyclone IV datasheet https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-iv/cyiv-53001.pdf
16. Universal Serial Bus 3.1 Specification, Jul. 2013. Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, Renesas Corporation, STEricsson, and Texas Instruments.
17. CYUSB3KIT-003 SuperSpeed Explorer Kit Setup: http://www.cypress.com/file/134006
18. E. Y. Lam, and G. S. K. Fung, “Automatic White Balancing in Digital Photography,” in Single-Sensor Imaging: Methods and Applications for Digital Cameras, Edited by Rastislav Lukac, CRC Press 2008, Pages 267–294.