Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

**By reading and using the thesis, the reader understands and agrees to the following terms:**

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.

2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.

3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

**IMPORTANT**

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.
Abstract

In today’s traffic monitoring system, image processing with software cannot meet the requirement of real-time processing demand because of its serial data processing structure. In this thesis, one hardware based image processing and networking system prototype is raised out to achieve the high throughput real-time image processing as well as data transmission.

The hardware architecture is mainly composed of four parts: raw data collecting part, image processing part, data transmission part and control part. The result of the design is to detect the number plate in a real-time system, locate the position of the number-plate and transport the data from FPGA side to PC through Ethernet networking system.

There are three highlights and contributions for this design:

1. It introduces a new way for designing the hardware architecture by using ANSI-C instead of RTL so that to ease and shorten the design period. For number-plate detection part, all the algorithm is finished in pure ANSI-C and then using a tool called CyberWorkBench to verify the correctness of the design and automatically generate the Verilog code for implementation.

2. In the past designs, the designer usually focuses on either image processing or data transmission. This design fully provides a prototype for integrating both these two parts in one on board system, which is also a key difficult point for the design.

3. The design can tolerate high data throughput at the rate of 60 frames per second, both for image processing side and networking data transmission side. Thus, this design is suitable and ideal for processing real-time data.

Overall, the design provides a complex hardware based all-in-one system that can collect raw data, do image processing and transmit data through network. The implementation tools for this design is CyberWorkBench, Quartus II and Wireshark.

Key word: FPGA, feature detection, data transmission
Acknowledgments

First of all, I’d like to thank my supervisor, Dr. Ivan Ho and Dr. Benjamin Carrior Schafer, who have offered me great instruction and suggestion for my research work and thesis. Under their patient instruction and guidance, I managed to finish my research step by step within one year’s time and get myself well improved. During the study, I met with a lot difficulties and sometime even suspected my ability to finish all the tasks. But with their patient encouragement and instruction, I finally overcome all the obstacles and successfully finish all my research work.

During one year’s research study, I become more familiar with the FPGA design and put a lot effort to accomplish the design tasks. This years’ experience not only teaches me how to perform well in academic area but also teaches me how to set correct attitude towards failures and challenges.

Last but not least, I want to thank all my parents, friends, colleagues, classmates and especially my roommate for their support.
Table of Contents

Contents

1. Introduction .................................................................................................................................. 1
   1.1. Background ......................................................................................................................... 1
   1.2. Current academic achievement ............................................................................................ 2
   1.3. Objectives ........................................................................................................................... 3
   1.4. Thesis Organization ............................................................................................................. 3

2. Platform ...................................................................................................................................... 4
   2.1. Introduction of FPGA ........................................................................................................... 4
   2.2. Hardware description language ............................................................................................ 5
   2.3. VHDL and Verilog HDL ....................................................................................................... 6
   2.4. Hardware design and simulation tools .................................................................................. 7
       2.4.1. Altera’s Quartus II ....................................................................................................... 7
       2.4.2. Signal Tap Logical Analyzer .................................................................................... 8
       2.4.3. Introduction of High Level Synthesis ...................................................................... 8
       2.4.4. Introduction of CyberWorkBench .......................................................................... 9
   2.5. Quartus designing flow introduction .................................................................................... 10
   2.6. DE2_115 FPGA board introduction ................................................................................... 13

3. Image processing in CyberWorkBench ..................................................................................... 14
   3.1. Development flow and concept ........................................................................................... 14
       3.1.1. General idea of the image processing design ............................................................ 14
       3.1.2. Design flow .............................................................................................................. 14
   3.2. Algorithm for number-plate detecting and locating ............................................................ 15
       3.2.1. Basic algorithm for number-plate detection ............................................................... 15
       3.2.2. HSV module ............................................................................................................. 20
   3.3. CyberWorkBench for algorithm implementations ............................................................... 20
       3.3.1. Editing an ANSI-C description for synthesis ............................................................ 21
       3.3.2. Setting up synthesis mode and other synthesis constrains ...................................... 22
       3.3.3. Create the Function Unit constrain files (Resource Allocation) ............................... 23
5.1. Brief introduction of data transmission designing method .......................................................... 59

5.2. UDP protocol ........................................................................................................................................ 60
   5.2.1. UDP protocol brief introduction ............................................................................................. 60
   5.2.2. UDP packet structure ........................................................................................................ 60
   5.2.3. Checksum computation ........................................................................................................ 61

5.3. IP protocol ........................................................................................................................................... 62
   5.3.1. IP protocol instruction .......................................................................................................... 62
   5.3.2. Packet structure – IPV4 .................................................................................................. 62

5.4. MAC protocol ..................................................................................................................................... 64
   5.4.1. MAC protocol introduction .............................................................................................. 64
   5.4.2. Ethernet frame structure .................................................................................................... 64
   5.4.3. Ethernet II frame structure .............................................................................................. 64

5.5. Ethernet package structure based on DE2_115 board ........................................................................... 66
   5.5.1. MAC header .................................................................................................................. 66
   5.5.2. IP header ....................................................................................................................... 67
   5.5.3. UDP header .................................................................................................................. 67
   5.5.4. Header checksum method ............................................................................................. 67

5.6. Ethernet data transmission module design ......................................................................................... 68
   5.6.1. Data format reconstruction module .............................................................................. 68
   5.6.2. Data-generation module ............................................................................................... 69
   5.6.3. CRC checksum module ................................................................................................ 70
   5.6.4. Data-transmission module ............................................................................................ 70

5.7. Quartus II implementation ................................................................................................................. 71

5.8. PC side data receive ............................................................................................................................ 72
   5.8.1. Environment and Software Preparation ........................................................................ 72
   5.8.2. Data transmission results .............................................................................................. 72

6. Historical methods used to achieve data transmission ......................................................................... 74
   6.1. NIOS II processor with NicheStack TCP/IP stack ....................................................................... 74
      6.1.1. Design idea ................................................................................................................. 74
      6.1.2. Problems of the design .............................................................................................. 77

7. Conclusion ........................................................................................................................................... 78
7.1. Conclusion of the project ................................................................. 78
7.2. Further improvement ........................................................................ 78
  7.2.1. Detection algorithm ................................................................. 78
  7.2.2. Data storage format ................................................................. 78
Bibliography ............................................................................................ 79
Appendix .................................................................................................... 81
1. Detailed information of the DE2_115 board ....................................... 81
   1.1.1. DE2_115 FPGA board tool package ........................................ 81
   1.1.2. The composition and modules of DE2_115 board ..................... 81
   1.1.3. The system diagram of the DE2_115 board ............................... 82
2. Source codes for number-plate detection algorithm ........................... 84
3. CRC checksum codes ........................................................................ 88
List of Figures

Figure 1  FPGA manufacture market share brief overview in 2013 ............................................................. 5
Figure 2 VHDL entities and architectures .................................................................................................... 6
Figure 3 Architecture of Verilog Modules. .................................................................................................. 7
Figure 4 Example of Data Acquisition of a Recurring Data Pattern ............................................................. 8
Figure 5 Comparison of general design period and C-based design period [6] ............................................ 9
Figure 6 comparison of RTL-based chip area and C-based chip area [6] ................................................... 10
Figure 7 Quartus II design flow .................................................................................................................. 11
Figure 8 Top view of the DE2_115 board [7] ............................................................................................. 13
Figure 9 Brief process of number-plate detection and location .................................................................. 18
Figure 10 Number-plate detection algorithm flow ..................................................................................... 19
Figure 11 HSV color module [14] .............................................................................................................. 20
Figure 12 project setting according to the FPGA board model ................................................................. 21
Figure 13 part of the source ANSI-C code ................................................................................................. 22
Figure 14 successful parsing of ANSI-C codes .......................................................................................... 22
Figure 15 Automatic pipeline scheduling model set .................................................................................... 23
Figure 16 Signal table of automatic pipeline mode .................................................................................... 23
Figure 17 two constraint files ..................................................................................................................... 24
Figure 18 details of the library operators ................................................................................................. 24
Figure 19 Cyber RTL Synthesis Report ...................................................................................................... 25
Figure 20 bdltran Error Viewer which shows error or warning of the design ............................................ 25
Figure 21 the datapath viewer which shows the schematics of the system ................................................. 25
Figure 22 the signal table of the system ..................................................................................................... 26
Figure 23 SW pattern files includes RGB values and coordinate information of the BMP file ................. 27
Figure 24 console window for cycle-accurate simulation ........................................................................... 28
Figure 25 waveform of the cycle-accurate simulation ................................................................................ 28
Figure 26 the automatically generated Verilog codes ................................................................................ 29
Figure 27 the designing process on CWB platform .................................................................................... 30
Figure 28 Delay information from the QoR file ............................................................................................ 30
Figure 29 Function Unit performance of the system .................................................................................. 31
Figure 30 System brief structure ................................................................................................................. 34
Figure 31 Pixel array description [11] ........................................................................................................ 35
Figure 32 Pixel color pattern detail (top right corner) [11] ........................................................................ 35
List of Tables

Table 1 VGA Connector Pinout and Signals .............................................................. Error! Bookmark not defined.
Table 2 the functional keys of the digital camera demonstration ................................................. 46
Table 3 ATPLL generated clock signals ...................................................................................... 47
Table 4 User defined Ethernet frame structure .............................................................................. 66
## Glossary of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>PLD</td>
<td>Programmable Logic Devices</td>
</tr>
<tr>
<td>CPLD</td>
<td>Complex Programmable Logic Device</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>HLS</td>
<td>High Level Synthesis</td>
</tr>
<tr>
<td>CWB</td>
<td>CyberWorkBench</td>
</tr>
</tbody>
</table>
Chapter 1

1. Introduction

Introduce the project here with some background information

1.1. Background

The increase in population density in cities across the world has led to the development of wide public transportation systems in most of them. Hong Kong, as the most densely populated city in the world, benefits tremendously from mass public transportation systems. These include a metro (MTR), buses and ferries.

The widespread network of these public transportation systems, combined with their conveniences and low prices makes them the most popular means of transportation, but re-evaluating closely their role could lead to their complementary use in new application domains. For example, buses continuously circulate through pre-defined routes in the city. This characteristic makes buses perfect mobile ‘data acquisition’ centers. Equipped with different sensors could monitor air quality, traffic conditions, and even serve as surveillance patrols.

One of the main application domains of the proposed system is its ability to work as a mobile surveillance platform. In order to achieve this purpose, each bus will be equipped with multiple cameras which can capture real-time data such as the feature of people and number plates of the cars. With this first-hand information, a lot of further work can be done. For instance, the human face feature detection can help the police to capture criminals and the information of car number plates can help to surveillance whether a car is stolen or if the car is parked in the proper places or not. Since the data capturing process is real-time based and with the help of the buses, a huge amount of data can be selected and the useful information can be used or analyzed for different purpose. The system will then transfer this information to the control center at the mobile uplink point at the bus stop for further analysis.

A dedicated HW system might be needed in order to deal with the large amount of data that needs to be processed in real-time. For this purpose FPGA seem to be a promising solution due to their ability to fully parallelize applications and thus speeding up computationally intensive tasks as the one in this project.

One shining spot for this project is the unusual hardware designing method it uses. All the design will be done in using C instead of RTL, which will largely ease the workload and more easily for the designer to modify the system. The main designing idea for the entire system is to using C programming to realize
the basic function of the system and then using Cyber Work Bench to verify the designing, automatically transfer the C codes to Verilog codes and then construct the whole system in Quartus II software.

1.2. Current academic achievement

Because of hardware application can achieve better real-time performance compared with software applications, using FPGA to implement vision computation such as feature detection is becoming more and more popular in today’s academic research. For example, implementing feature detection and location algorithm in unmanned aircraft vehicle by using FPGA is a hot application in many universities or companies.

Usually, there are two ways to use FPGA for doing feature detection work. One is to combine the hardware application with software system to achieve the detection and data process function. Another way is to use pure hardware system to conduct all the detection, tracking and data process. Students coming from Brigham Young University raises out a Harris Feature Detector based algorithm to detect and track the real-time target object and then use software platform to accomplish the remaining work includes translation, rotation, and scaling [1]. Another work which purely use FPGA board to construct the whole system is to develop an architecture with a two-dimensional (2D) array of FPGA/DSP-based reprogrammable processors. This architecture can process parallel/pipeline procedures, handling various simultaneous input images, and cover a wide range of real-time computer vision applications from pre-processing operations to low-level interpretation [2].

- For data transmission out of the FPGA board, the historical researches discussed several ways to achieve data transmission function. For example, using Ethernet, serial bus, USB, etc. One former research describes a way of using serial communication technology to transfer data from PC to FPGA. The structure of the system is to use a camera to capture image and then doing image processing on PC side, using serial bus to transfer data to FPGA side to do comparison work [3]. If data throughput is not huge, using serial communication is a good option for it is relevantly easier to construct the data transmission protocol and high efficient. However, if the data throughput is large, using other ways of data transmission is necessary. Altera also provides its official demonstration of Ethernet data transmission by using NIOS II processor to construct a TCP/IP protocol based web server [4]. By using this method, data from FPGA can be sent through the on-board Ethernet port and received by the host PC. Considering the resource of the board, especially the memory consume, if one board needs to achieve both image processing and data transmission simultaneously, the entire system needs to be re-designed in order to avoid memory conflict.

For developing a traffic monitoring system, former researches also provided related works. One research is focused on developing an FPGA-based camera control system that can monitor the real-time traffic. The structure of the design is to use FPGA to control the camera to capture the live stream video, do
video analytics on board, transfer the video to PC side by using Ethernet and storage the video in PC as well as do further traffic status analytics on PC side [5]. This structure provided me great inspire. But the difference of my work is that all the algorithm and data process are done within the board, the host PC is used to store the data.

1.3. Objectives

The main purpose of this project is to implement the function of feature detection on FPGA board to achieve real-time data collecting, pre-processing and transfer the data to the receiver by using Ethernet transmission technology. Furthermore, the results of this project can be applied to many uses such as data analysis, cloud storage or others. The first-hand information collected by the cameras and pre-processed by the FPGA can also be used to analyses the real-time traffic on the road.

1.4. Thesis Organization

The thesis is mainly composed of seven parts. The first chapter is the instruction of the whole project, describing the motivation, key concern, highlight spot and design thought of the project. The second part mainly discuss the tools, both hardware and software which are used in this design and some brief introduction of the FPGA industry. The third and fourth part are the core of the thesis. Chapter 3 and chapter 4 illustrates the algorithm of the number-plate detection algorithm and how this algorithm is achieved both in software and hardware. Within this chapter, it also gives out some analysis about the result. Chapter 5 discusses how to realize data transmission through the Ethernet port on the FPGA board and how these data is received by the PC. In this chapter, the network is established through UDP, IP protocol and MAC layer by directly using the Verilog HDL language. Chapter 6 listed all the historic methods which were considered during the whole designing process and analyzes the disadvantages of using these methods. Chapter 7 is the conclusion part of this thesis. It describes the result of the design, its performance and possible aspects which can be further improved. Then comes the bibliography. In the appendix part, it listed some of the original codes of the design and some brief explanations.
Chapter 2

2. Platform

This design is mainly implemented in Altera’s DE2_115 FPGA board. Another hardware used is an external camera which can be mounted with the FPGA board through the GPIO port. For software side, CyberworkBench is used for algorithm designing, verification and generating Verilog HDL codes. Quartus II 13.0 is used for system architecture design.

2.1. Introduction of FPGA

The FPGA are the result of the convergence of two different technologies, the logical programmable devices and the ASIC [6]. FPGA is an integrated circuit which can be developed, configured and designed by a developer according to his customized requirements. Generally speaking, FPGAs is composed of larger sources like logic gates and memory blocks to achieve different digital computation functions. The “logic blocks” refers to components which can be programmed. FPGA allows hierarchy interconnect of these blocks to make them “wired linked”. The logic blocks within the FPGA can perform complicated computational functions or simple logic gates like NOR or XOR. These computational functions can be designed by developers to accomplish complex tasks. Usually, FPGA contains memory components, such as simple flip-flop or other memory blocks like SDRAM.

Historically, FPGA performed slower, less efficient and less functions than fixed ASIC and consumed larger areas. However, recently, new generation FPGA like Xilinx Virtex-7 or the Altera Stratix 5 have overcome the above weaknesses [7]. They provide significantly reduced power, faster speed, lower material cost and more importantly, they can be more possibly re-configured “on-the-fly”. Some FPGA allows partial-reconfigurable for allowing one part of the device can be reconfigured while other parts are still running. Compared with CPLD, FPGA has the advantage of more flexible because FPGA is interconnected. Usually, CPLD is much smaller in terms of resources than FPGA for its restrictive structure. Typically, only FPGA contain more complex embedded functions like adders, multipliers, memory and serdes[7]. Theoretically speaking, any computational work can be conducted by using FPGA and actually, a microprocessor can be also implemented in FPGA to perform various functions. With the development of FPGA, it can be applied into more and more areas such as Digital Signal Processing, , speech recognition, vision computation, ASIC prototyping, etc.

Nowadays, main manufactures of FPGA are Xilinx, Altera, Atmel, Actel and Lattice Semiconductor. In this designing project, we will use Altera FPGA board to conduct all the functions of the system. The board is DE2_115.
In figure 1, it shows the market share of the FPGA manufactures in 2013[8].

![FPGA manufacture market share brief overview in 2013](image)

**Figure 1** FPGA manufacture market share brief overview in 2013

2.2. Hardware description language

In electronics, hardware description language is a specialized computer language used to describe the structure, design and operations of the circuit. Most commonly, it is used in digital logic circuits. Usually, we can use HDL to define the function of different modules, the link relationship between each component and set various signals to test the whole circuit. It also allows for the compilation of the HDL programs to a lower level specification of physical electronic components, translating the HDL programs to numerous adders, multipliers which can execute the defined circuit functions. Thus, an integrated circuit can be designed.

The HDL is a text-based language. The main difference between HDL and other programming languages like C programming is its notion of time. For software programming languages, they execute codes serially and any execution cannot be controlled by time. But for HDL languages, these problems can be easily solved because HDL language can concurrently execute codes. Different modules can be triggered at the desired time and the whole system can do parallel process. Thus, for HDL codes, one important thing the designer should be careful with is the ‘time series’.

For designing procedure, a designer usually separate the whole system into different modules. Each module will perform its own function. For each module, it consist of expressions, statements and control structures. Then, by defining a top module, different modules can be connected together and form the whole system. Furthermore, clock signals, wires and other parameters are needed for mapping the entire system. After coding, a simulation procedure is performed to check whether the function of the designed system can be performed as expected. If not, the designer should add debugging signals to test where the
problems are and make proper correction. Finally, the synthesize tool will map the HDL descriptions into a gate net-list.

2.3. VHDL and Verilog HDL

VHDL is a Hardware Description language (HDL) and requires a synthesizer to convert the source code into a gate netlist. Usually, we can use VHDL to write the functional code and a testbench that verifies the correctness of the design. Usually, by viewing the simulation waves of the design and comparing the results with the expected results can tell the user whether the design is correct or not [6]. One key advantage of VHDL is that user can describe the behavior of the expected system and simulate it before the synthesis tools translate it to a real hardware code.

![VHDL entities and architectures](image)

*Figure 2 VHDL entities and architectures*

Verilog is more like C programming language, which is much easier for the beginner to use. The language is case-sensitive, has a preprocessor like C, and the major control flow keywords, such as "if" and "while", are similar [6]. A Verilog design usually consists of a hierarchy architecture. Functions are designed in individual modules and one top module will describe the connection relationship between each module. Generally, to define a module, the user needs to specify its input, output or bidirectional ports. Inside the module, registers and wires are defined. Then the statements describe the main function of the module. For example, the sequential codes are included in ‘begin-end’ structures. All concurrent statements and ‘begin-end’ statements are executed at the same time by the controlling of the clock signal,
which shows that Verilog is a dataflow language. The synthesizable statements can finally be synthesized to net-list that are used to implement the design into logic programmable device.

![Figure 3 Architecture of Verilog Modules](image)

For this project, we choose to use Verilog as the hardware designing language because we are more familiar with its C programming liked statements. On the other hand, since the first-hand design is achieved in C, it is much easier for a designer to read the translated C codes and find out mistakes.

### 2.4. Hardware design and simulation tools

#### 2.4.1. Altera’s Quartus II

The Quartus II is Altera’s integrated PLD/FPGA development software. It supports schematic design, VHDL & Verilog design and Altera Hardware Description Language design. Furthermore, synthesizer and simulation tools are also embedded. The Quartus II software provides the user a complete environment for design a system-on-chip project.

The Quartus also offers user a graphic interface for design, which includes every step from entry design to final device programming. Moreover, Quartus II supports using Altera’s IP core that includes a macro module library of LPM/MegaFunction. It allows users to directly invoke or embed the already existed mature modules into their designs, which will largely save the development period as well as ease the complexity of the whole project. Moreover, through the combine with DSP Builder or Matlab/Simulink,
Quartus II can easily implement various DSP application system, support the development of Altera’s SOPC and integrate system-level design, embedded software design as well as programmable logic in one project. Because of its powerful design ability and easy-to-use interface, Altera Quartus II is more and more welcomed by digital system designers.

2.4.2. Signal Tap Logical Analyzer

The signal tap provided by Altera is for helping to user to have a much easier way to debug their system while the project is running. User can define monitored signal like input ports, output ports, different parameter or other signals. While the entire project is fully running, the signal tap logical analyzer can show pre-defined signals’ numerical value, letting the designer to judge where the problem is. One key advantage of using the embedded signal tap is that it doesn’t rely on any external equipment. The designer can debug their system by tracking any internal signals in a real-time state. This will prove the efficiency of system debugging and provides the user a more user friendly environment to find out problems. Users can easily set up the target internal pins within the signal tap analyzer, define the clock period, configure the debugging project and run the process to monitor whether each target internal pins are working properly or not. The bellowing figure is an example of using signal tap to monitor the input port performance.

![Figure 4 Example of Data Acquisition of a Recurring Data Pattern](image)

2.4.3. Introduction of High Level Synthesis

High-level synthesis (HLS), not like the traditional RTL synthesis, is an automated design process. It uses high level language like ANSI C to describe the expected behavior in original codes and then automatically generate the relevant digital hardware that can implement such defined behavior [9]. Currently, the widely used and commonly accepted synthesizable high level languages are ANSI C, C++ or System C [10]. In this design, ANSI-C is chosen as the high-level language to describe the feature detection algorithm. The process of conducting high level synthesis is first to write all desired behavior in source C codes, then such codes will be analyzed, architecturally constrained and scheduled to create a register-transfer level (RTL) hardware design language (HDL) [10]. Then by using the logical synthesize tool, it will synthesis RTL script to the gate level.
By using this designing method, it provides the designer a more convenient environment compared with the traditional hardware design flow by directly writing HDL codes to describe all the system. In this design, the HLS tool is CyberWorkBench, which will be introduced in the later paragraph.

2.4.4. Introduction of CyberWorkBench

CyberWorkBench (CWB) is NEC’s C-based integrated environment for System LSI design [11]. It has all the tools needed for efficient C-based design: a behavioral synthesizer, simulator, and formal verifier [11]. CyberWorkBench provides all-in-c synthesis and verification: Any type of module (controller or data path) can be synthesized and verified at C-level.

By using the CWB, the design cost will be largely reduced [11]:
- Description reduction: 5-30%, simulation speed 100X
- Design man-month reduction (e.g., 80MM → 10MM)
- Design period for HW (blue) and SW (red) are both reduced
- Higher reliability: fast HW-SW co-verification (cycle accurate)

![Figure 5 Comparison of general design period and C-based design period [6]](image)

On the other hand, CWB can also efficiently reduce the chip cost [11]:
- HLS can generate smaller and lower power designs compared to manual RTL designs through maximum resource sharing
- Area and performance optimization for Altera, Xilinx FPGA
Due to its obvious advantages, CWB can be widely implicated in the following areas:
- Digital Circuits design: mobile phone, digital camera, etc.
- Accelerate of server: Big data
- Replacement of real time processing MPU
- Controller: automobile, factory automation, etc.

2.5. Quartus designing flow introduction

The development environment is the Altera Quartus II design software. It provides a complete development environment for SOPC design, including all solutions for all phases of FPGA design. From original Verilog code writing to final system compiling, all the process can be finished in Quartus II. The edition of Quartus is Quartus II 13.0 web edition. For each phase of the design, the user graphic interface, EDA tool interface or command-line interface are all provided within the software. One important component of Quatus II is its modular compiler which is used to compile all the project and to give out essential information of the design that indicates the quality of the design. The compiler includes the following modules [19]:
- Analysis & Synthesis
- Partition Merge
- Fitter
- Assembler
- Classic Timing Analyzer and TimeQuest Timing Analyzer
- Design Assistant
- EDA Netlist Writer
- HardCopy Netlist Writer

User can choose to run the complete compile or just run part of the compile functions.
Design Entry

In this design, all the functions of the system is realized through Verilog codes, no block diagram design is used in this design. Thus, the Quartus II Text Editor is used to text in the Verilog codes which describe the behavior of the function. According to the function of the module, the Verilog HDL codes is written separately and saved individually.

Altera megafuncitons are blocks which are already designed and embedded in Quartus II software, which are high-level and complex blocks. Generally speaking, the megafuncitons are already designed modules from Altera that user can directly insert them into their design through proper customized configuration. The parameterizable megafuncitons will largely comfort the user to design and optimize their design as well as improve the Altera device architectures. For example, it is very convenient to access some Altera device-specific features like memory, PLLs, double-data rate input/output (DDIO) circuitry through using the megafuncitons.

Synthesis

The function of synthesis is to integrate all the design files into one project hierarchy. This is like a database that involves everything needed for the project. This database is used while the project is under processing. Furthermore, some other compilers can update the database until it is fully optimized.

When a project is start to compile, the Analysis stage will examine the logical completeness and consistency of the project, check the syntax error for the HDL codes and the boundary connectivity. Then, Analysis and Synthesis synthesizes and conducts technology mapping on the logic in the project’s files.
Place & Route
Within this stage, the Fitter places and routes the design. It allocate the available hardware resources of a device according to the project’s logic and timing requirements. It will optimize logic function to the best suited logic cell and select appropriate interconnection paths and pin assignments. When doing pin assignments, two methods can be adopted. First is to manually assign the pin according to the user manual of the DE2_115 board. Second way is to import the pin assignment file which is provided by manufacture and do some proper modification so that all the pins can be assigned properly to meet the project’s requirement. Then, the assembler automatically converts the Fitter’s device, logic cell and pin assignments into a programming image for the hardware board.

Timing Analysis
The classic timing analyzer automatically performs timing analysis on the design following a full compilation [14]. It allows the user to specify the desired speed performance for the entire project for individual modules, nodes or pins, etc.

Simulation
Any part of the design can be simulated during the project execution. The user can specify the type of simulation, functional or timing simulation that should be performed. In this design, since the image process algorithm has been verified in CWB stage and the Verilog codes generated is assumed to be correct, the simulation stage is jumped.
After the whole design is completed, the configuration file is sent to the board to test whether the design is correct. If not, debugging step is launched.

Programming & Configuration
When all the design is finished and the compile has be operated, the configuration file is already created by the Assembler. This file is a .qpf file that be sent to the FPGA board through the USB blaster cable. When this file is completely sent to the FPGA board, thus the hardware device can operate all the functions as user designed.

Debugging
This is a very important step throughout the design. Generally, at the first time the user download the configuration file to the FPGA board and test the function of the design, the final result is often wrong. How to find out the error of the design is one of the keys to a successful design.
In this design, we choose to use signal tap which is provided by Quartus to do the system debugging. With this tool, user can set target signal for tracking and find out the problem of the design and then make proper changes to correct the result and optimize the whole project.
2.6. DE2_115 FPGA board introduction

The DE2-115 development board is a powerful hardware development platform that can be used to do a lot of applications. It integrates a lot of configurable devices such as logic devices, memory chips, different type of data communication chips, NIOS II processor chip, etc. With these wide range of chips, different applications can be conducted on this board [12]. Details of the composing of this development board can be seen in the Appendix or viewed through its manual book as well as the official website.

The bellowing picture shows the appearance of the DE2_115 development board. Details of this board can be reviewed in the Appendix section.

Figure 8 Top view of the DE2_115 board [7]
Chapter 3

3. Image processing in CyberWorkBench

3.1. Development flow and concept

3.1.1. General idea of the image processing design

For common traffic surveillance system, we use a dedicated web camera at a fixed place, for example, at the charging station, to capture all the traffic stream and then upload the source file to the server for storage. In this design, the camera is defined to be mobile. It can be installed on each bus and capture the real-time images. Since each bus has its own working routine, the camera based system can be seen as a moveable station which can grab millions of information every day. One problem needs to be solve is that since we will get millions of information each second but most of these information is not valuable, can we find a way to pick out the most valuable information within the image and largely reduce the amount of data which we need to transfer to the server?

The general idea for the image processing is to detect the interested area within the scenery and make the accurate location. For traffic surveillance system, it is assumed that the number plate on each car is a key element and the important information which is very valuable for data recording and further analyzing.

3.1.2. Design flow

3.1.2.1. Image processing algorithm ANSI-C source code and hardware C writing

The first task is to write the pure ANSI-C source code to achieve the image processing algorithm such as Sobel filter design and number plate detection. Then, the software C code will be modified into hardware C code which can be synthesized at RTL (resistor transistor logic) level. The code should be written carefully because there are many restrictions when the code is implemented at a hardware stage.

3.1.2.2. Algorithm function testing and Verilog/VHDL code generating

When the C code is written, it should be tested to see if the function performs correctly or not. This operation will be done in Cyber Work Bench (CWB). At this stage, original C code will be verified first in software simulation and the result will be generated to show if the function of the code is correct. Then, cycle verification and RTL verification will be done to verify if the C code can be run correctly on the hardware or not.

If both cycle verification and RTL verification can generate the same result as software simulation did, we can confirm that the C code which we wrote can achieve the function which we need. After that, the
Verilog or VHDL code can be generated automatically by CWB when we do the logic synthesis operation. Then we get the v files (also called a function module) which we need to construct a hardware project.

3.1.2.3. **FPGA project generating in Quartus II**

Different function modules of the system will be generated by the proposed method. At this stage, these modules will be connected together to form the whole FPGA project in Quartus. Different signals including control signals, variable signals and clocks will be defined. Different modules should be connected correctly and if needed, interfaces will be defined to link two modules. When the module connection step is done, further steps such as clock generating and pin configuration action should be taken to make the project applicable in the FPGA board. Finally, the whole project should be compiled to check if there is any error exited. After everything is correct, an sof file will be generated. This file will be sent to the FPGA board directly by using the USB links.

3.1.2.4. **Hardware testing and modification**

Then it comes to the testing procedure. When all the codes are sent to the FPGA, testing should be done to see whether the function of the board is correct. Monitoring of the signal can be done by using signal tap installed in Quartus to observe the waves of the signal. This will help the developer to see whether signals are performed properly. If there is some error or mistake, it is convenient to check. After finding the reasons of the wrong result, some modification might be needed in the original code or the algorithm might to be improved.

The whole testing process might be time consuming and the testing process will be taken for several times in order to ensure that all the results are correct.

3.2. Algorithm for number-plate detecting and locating

Not like using pure software programming methods to do the protection, which can use unlimited memory to buffer the image and save some intermediate results for future use, the algorithm for hardware usage should be carefully designed. Considering of the hardware resource problem, parallel processing speed of the FPGA and other constrains of the board, the pixel-based designing method is chosen for the project.

In this case, we assume that the background of the number-plate is YELLOW according to HK’s relevant traffic regulations.

3.2.1. **Basic algorithm for number-plate detection**

According to the characteristic and considering the efficiency of hardware, the feature detection algorithm is designed to be pixel-based. Generally speaking, just as normal screen scanning process, the algorithm is scanning the scene pixel by pixel and then line by line to detect whether the targeted feature is included in
the scene or not. Therefore, for the detection module, each time one pixel is coming in and be judged if it is the target pixel or not.

In the real case, since the background of the number-plate is yellow, each yellow pixel will be considered as the potential target pixel. The key point of the algorithm is to find the first line of the number-plate and eliminate the noise pixels so that to accurately locate the number-plate.

To achieve this function, each time when a yellow pixel is first founded, its coordinates will be recorded and a counter will start to calculate how many continuous yellow pixels are and compared the number with the threshold. If the number exceed the threshold, this line will be considered as the upper-line of the number-plate, else, it will be considered as noises. If the upper-line is found, one flag will be set and the algorithm will start to find the bottom line of the number-plate. The method is same. Thus, the number-plate area can be located and in order to show the plate area clearly, one red block will be displayed on the number plate area, the background of the plate will be highlighted and all the other parts will be dark.

One thing to mention is that for the purpose of increase the accurate of detecting “yellow” pixels, in this design, all the RGB values of pixels will be transferred to HSV domain.

The main steps of the detection process is showed as below.

3.2.1.1. **Pixel value transform and comparison**
Each time, one pixel will come into the detection module and be judged if it is the pixel within the number plate area according to its HSV value. If the HSV value is outside the threshold range, just turn it to black and output it.

3.2.1.2. **Find out the first pixel which meet the requirement**
At the first time one pixel meets the HSV requirement, its location should be remembered and one counter will start to work. The counter will count how many continuous yellow pixels are.

3.2.1.3. **Locate the first line of the number-plate**
If the counter’s number exceeds the pre-set threshold, this line can be judged as the first line of the number-plate. The first yellow pixel will be marked as the “start point” of the number-plate. Also, the width of the number-plate can be found. On the other hand, if the yellow pixels are not continuous or only a small amount of yellow pixels are continuous, we consider them like noises. For example, the taillight of the car might be yellow but should not be considered as number plate area.

3.2.1.4. **Highlight the background of the number-plate**
The following yellow pixels whose horizontal coordinates are within the number-plate range, they will be turned to white pixels.
3.2.1.5. **Find out the bottom line of the number-plate**

By calculating the continuous yellow pixels again, the bottom line of the number-plate can be easily found.

---

**Flowchart:**

1. **One pixel comes in:**
   - **RGB to HSV domain**
   - **Yellow pixel or not?**
     - **Yes:**
       - Record the coordinates; Count plus 1
       - (Count represents the number of continuous yellow pixels; Flag_1=1)
       - The first founded yellow pixel?
         - **No:**
           - Output the pixel as a black dot; Flag_1=0; Count=0;
         - **Yes:**
           - Count plus 1
           - Count > threshold?
             - **No:**
               - Count plus 1
             - **Yes:**
               - First-line founded; Flag_2=1; Record the position of the end pixel of the line
               - Highlight the number-plate area according to the coordinates recorded
               - Find out the bottom line of the number-plate (the method is the same as finding the first-line)
               - Display the red block
Figure 9 Brief process of number-plate detection and location
3.2.1.6. Example of Number-plate detection algorithm flow

In order to make the number-plate area more clearly for sightseeing, one red block is drawn to track the number-plate area.

After the detection process, the whole picture will be represented only with two colors, white for number-plate background and black for all the reset. Therefore, it largely reduce the amount of data which we need to transform.

Figure 10 Number-plate detection algorithm flow
3.2.2. HSV module

Usually, we use RGB values for representing one pixel in general cases. But for precise color detection, directly using pixel's RGB value might cause a lot problems. For example, if the desired color is yellow, the standard R, G, B value should be R=255, G=255 and B=0. But actually, since R=230, G=245 and B=50 should also be considered as yellow and be calculated, it is very difficult to set a dedicated threshold for R, G, B values to do the color detection.

Instead directly use the RGB value to represent a pixel, one possible way is to use the HSV domain to transform the raw R,G,B value. In the HSV module, H represents for hue, S represents for saturation and V represents for value (brightness). In this domain, pixel's color is represented in a cylindrical-coordinate system. Compared to traditional RGB domain, HSV domain can represent color's brightness and hue more cleared. Thus, it is more reliable to use HSV threshold for doing color-based detection and location.

RGB to HSV change module:

Supposing that one pixel's R,G,B value is known, set max equals to the maximum number of R,G or B, then

$$
\begin{align*}
    h &= \begin{cases} 
        0^\circ, & \text{if } \max = \min \\
        60^\circ \times \frac{g - b}{\max - \min} + 0^\circ, & \text{if } \max = r \text{ and } g \geq b \\
        60^\circ \times \frac{g - b}{\max - \min} + 360^\circ, & \text{if } \max = r \text{ and } g < b \\
        60^\circ \times \frac{b - r}{\max - \min} + 120^\circ, & \text{if } \max = g \\
        60^\circ \times \frac{b - r}{\max - \min} + 240^\circ, & \text{if } \max = b 
    \end{cases}
\end{align*}
$$

$$
\begin{align*}
    s &= \begin{cases} 
        0, & \text{if } \max = 0 \\
        \frac{\max - \min}{\max} - 1 - \frac{\min}{\max}, & \text{otherwise}
    \end{cases}
\end{align*}
$$

$$
\begin{align*}
    v &= \max
\end{align*}
$$

As defined, \(h \in [0,360^\circ]\), \(s \in [0,1]\), \(v \in (0,255]\).

In this number-plate detection case, the defined threshold of HSV is:

\(h \in (42,59), s \in (0,0,45), v \in (200,255]\)

The above HSV value is set to detect the background yellow area of the number-plate.

3.3. CyberWorkBench for algorithm implementations

One basic and highlight spot of this design is its “ALL-IN-C” designing method, instead of the traditional HDL designing procedure. The original algorithm is all written in ANSI-C and then verified by the High Level Synthesize tool. Finally, all the C-based design will be converted to HDL language automatically.
through the HLS tool. This procedure of design will largely improve the efficiency of the design as well as let the designer modify their design more comfortably.

In this design, the HLS tool we use is CyberWorkBench.

3.3.1. Editing an ANSI-C description for synthesis

After entering the CWB GUI, first create a project and make proper settings according to the FPGA device we use in this design. Also, select the basic library and standard function unit library which are compatible for the FPGA device.

![Figure 12: Project setting according to the FPGA board model](image)

Then, create a new C file for the project and input all the ANSI-C source codes. The C codes should be written in both in Software C and Hardware C type. The software C codes, including the main function, will compose a test environment which will be used to do software simulation test. The hardware C codes is the key kernel of the design, only the hardware C part of the source code will then be automatically converted to hardware description language. To identify where the hardware C is, there is an identifier called “process”. The hardware C part will be used to do the verification test in two ways, circle-accurate test and RTL test.

For this design, the input will be one BMP format picture.
When the editing part is finished, we need to parse the source code to check if there is any syntax error. The console window will indicate if any errors have occurred and where. A ‘light-blue’ folder with an .IFF file will be generated if the parsing was successful.

### 3.3.2. Setting up synthesis mode and other synthesis constrains

According to the characteristic of this design, there will be numerous data coming through the camera side and needed to be processed at the same time [15]. Thus, ‘Automatic pipeline Scheduling’ mode will be the only choice to assure the large throughput capacity.
For using the ‘pipeline’ mode, the whole process will be separated into several stages. Different ports, registers, function unit and memory can conduct instructions at the same stage which means realize the parallel process [16]. The key advantage of it is its capability of doing multiple instructions at the same time. At this case, latency is not the first concern but throughput is the main concern.

### Create the Function Unit constrain files (Resource Allocation)

Two new files will be generated as allocation the resource for the design:

- **Main-auto.FLIB**: FU Library file contacting FUs area and delay of FUs not contained in the initial FLIB/BILB files (empty in most cases).
- **Main-E.FCNT**: Constraint file limiting the number of FUs that the synthesizer can instantiate in parallel.
These two files will specify the kind, bit-width, delay, area and other information about each library operator, letting the designer to know how much resource the design might consume and make proper change if needed.

3.3.4. High level synthesis

After the preparation steps, high level synthesis can be taken. By clicking the “Synthesis/Generating RTL” option, the synthesis process will be conducted automatically. After the synthesis, one report file (Quality of Result-QoR) will be generated and appeared. It contains all the synthesis information, including FPGA resources used, critical path, design latency, etc. The information reported is based on the FLIB and BLIB file provided. Through this report, the designer can clearly figure out the design quality of the project according to several key results. For example, if the total latency exceeds the FPGA board’s maximum clock period (in this design, it is 20ns), the design might not work properly one the FPGA board and might face the problem of losing data.
Other valuable reports are also generated for designer’s reference. For example, the error file can show synthesis errors, warning and tips to help the designer to optimize their design.

3.3.5. Review the synthesis results

The synthesis results can be reviewed in different ways.

1) Schematic view: this opens the CWB’s datapath viewer. Clicking on any part of the schematic will also highlight the source code in CWB that corresponds to it (cross referencing)
2) Signal table: the Signal Table shows the timing diagram of the synthesis. When inputs and outputs are being read or written and when registers accessed. State = reset state and state 01 = stating doing the computation.

![Signal Table](image)

**Figure 22 the signal table of the system**

### 3.3.6. Design verification: software simulation

Inside the CyberWorkBench, there are three ways to do the simulation for checking the correctness of the design: software simulation, cycle-accurate simulation and RTL simulation.

For the simulation process, SW simulation turns the test file into tlv format files so that they can be input for further cycle-accurate simulation and RTL simulation. Besides, the SW simulation also gives out the standard ANSI-C results of the program. The cycle-accurate simulation and RTL simulations results will be directly compared with the SW simulation results to see if all the corresponding results are correct or not.

The first step to do normal SW simulation is to create the simulation scenario. This operation includes choosing ‘Behavioral’ as the simulation level for the scenario, set the top process, add C to preprocessor definition and input the BMP format file in to the ‘Pattern files’.

After all the preparations are done, build the project run the simulation until it is done. According to the ‘Automatically pipeline scheduling’ setting, all the simulations will be conducted as Transactional Level Simulation.

By conducting the SW simulation process, the compiler will only operate the software C part of the source code and provide the input tlv files which we need to do cycle and RTL simulation. Such input tlv files includes the R,G,B values of the BMP picture, coordinate information about each pixel within the picture and the output result of the detection module.
Explanation for the generated tlv files:
Input_row_a2.tlv: pixel’s red value of the BMP file
Input_row_a1.tlv: pixel’s green value of the BMP file
Input_row_a0.tlv: pixel’s blue value of the BMP file
Output_row_a2.tlv: pixel’s red value of the result
Output_row_a1.tlv: pixel’s green value of the result
Output_row_a0.tlv: pixel’s blue value of the result
ox.tlv: pixel’s horizontal coordinate of the BMP file
oy.tlv: pixel’s vertical coordinate of the BMP file

3.3.7. Cycle-accurate simulation

In order to verify the correctness of the timing and also the accurate performance of the synthesized designs, two options are available: (1) create a cycle-accurate model or (2) simulate the generated RTL code. Through the SW simulation, all the input files are ready for both verifications.

For doing cycle-accurate simulation, first create the scenario for it and then add the testbench. Since the design is under pipeline scheduling, all the input data will be read at the valid cycle.

After all the setting is done, import the existed input files, including input_row_a0.tlv, input_row_a1.tlv, input_row_a2.tlv, ox.tlv and oy.tlv to the pattern. Then, build the project and start the cycle-accurate simulation.

The test picture, through the SW simulation stage, is converted to 5 tlv files. 3 for R, G, B values of the pixel and 2 for pixel’s horizontal and vertical coordinates. All these files together can represent the original test picture. By inputting the 5 tlv files to the cycle-accurate process, it will give out 3 output files. The result of the output files will be directly compared with the output of the SW simulation. The console window will give out the result of the comparison, displaying how much data doesn’t match with each
other and the position of the error data. Thus, it provides a quick way for the designer to check where the error is.

Samples can be seen as below:

![Figure 24 console window for cycle-accurate simulation](image1)

On the other hand, the CWB also provides the wave form for users to view the results of the cycle-accurate simulation. After the simulation is done, a .vcd file will be automatically generated and by opening a waveform viewer (eg. GTKwave), the designer can easily find out whether the simulation result all matches with the SW simulation. Samples can be seen as below.

![Figure 25 waveform of the cycle-accurate simulation](image2)

Usually, by conducting the cycle-accurate simulation, the designer will find out timing errors of the design and can find out detailed problems of the design by checking the unmatched positions from the output files.

### 3.3.8. RTL simulation

The RTL simulation is the last step of the simulation for checking the RTL output. Like the cycle-accurate simulation process, first to create a RTL scenario and make proper settings. Also, choose to input the data at valid cycle. Then, input 5 tlv files which are generated from the SW simulation to the pattern.
Finally, build the project and start the simulation. Usually, the RTL simulation will consume a long time because it will simulate the real working environment of the hardware. Similarly, when the simulation is finished, 3 output files will also be generated so that they can be compared with the SW simulation’s outputs. If all the things is correct, both SW simulation results and RTL simulation results will match with each other.

If the results of cycle-simulation and RTL simulation are the same as the result of SW simulation and the result can meet with the requirement of the designer side, this means that the whole design should be theoretically correct and can be considered to put into hardware for testing.

3.3.9. Design space exploration

C-Based design allows the generation of different architectures with different area vs. performance constraints without having to modify the actual C code. This is mainly done by modifying the synthesis constrains. Eg. FCNT constraint file or synthesis options.

For example, by adding or decreasing FUs of the design, the area and performance of the design will be changed so that resources sharing can be maximized. Thus, after all the verification, doing some modification of the synthesis constraint might help to improve the quality of the design. But after each modification of the synthesis constraint, we need to re-verify the design and to see if the results are correct.

3.3.10. Verilog code generation

When all the testing, verification and debugging are done, it is assumed that the algorithm design is completed and the Verilog file, which is generated by the Synthesize/Generate RTL procedure, can be implemented into the hardware part design. All the C-based source codes are automatically converted to standard Verilog codes which uses hardware description languages to realize the function of number-plate detection function.

The Verilog file can be found in RTL-LS_script path under the “detect” blue folder.

![Figure 26 the automatically generated Verilog codes](image)
So far, the algorithm designing on CWB has been totally completed. The overall designing flow-chart can be seen as below:

![Designing Process on CWB Platform](image)

### 3.3.11. Analysis of the Result Quality

As mentioned before, the system will automatically generate a QoR (quality of result) report after the synthesis, showing the performance of the system and give out some key information like area of the components or delay of the whole design. Also, the QoR will specify each component’s performance and let the designer know where they can try to improve the quality.

Here, we will show some key performance of the design and analyze the effect. One key index of the report is the design’s delay. It is shown as below:

<table>
<thead>
<tr>
<th>Delay Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical Path Delay 21.29ns</td>
</tr>
</tbody>
</table>

![Delay Information](image)

For the DE2_115 board, its standard clock period is 20ns. It is obvious that this design exceeds the maximum clock period. Since the result is based on the
Figure 29 Function Unit performance of the system

FCNT (FU constraint), it is not 100% reliable and does not mean that the estimated delay will affect the accurate of the design when it is running on the hardware side. But, it is important to find out the reason why the critical path will exceed the maximum delay.

Through viewing the FU, it is clear that 2 function units called div20_20s_pipe and div21_21s_pipe are the main reason of the long delay. Looking back to the original source code, the two division FUs are generated by the following key codes:

```c
if (delta==0)
    {  
        h=0;
    }
else
    {  
        s=delta/maxhsv;
    }
```

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>FU name</th>
<th>kind</th>
<th>sign</th>
<th>bit width</th>
<th>area</th>
<th>delay (ns)</th>
<th>pipeline stage</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>add12u_1</td>
<td>+</td>
<td>unsigned</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>1.99</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>add12u_10</td>
<td>+</td>
<td>unsigned</td>
<td>10</td>
<td>10</td>
<td>13</td>
<td>1.93</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>add12u_11</td>
<td>+</td>
<td>unsigned</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>1.93</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>add16s</td>
<td>+</td>
<td>signed</td>
<td>10</td>
<td>10</td>
<td>16</td>
<td>2.31</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>add20s</td>
<td>+</td>
<td>signed</td>
<td>20</td>
<td>20</td>
<td>21</td>
<td>2.61</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>add20s_20</td>
<td>+</td>
<td>signed</td>
<td>20</td>
<td>20</td>
<td>21</td>
<td>2.63</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>div20_20s_pipe</td>
<td>/</td>
<td>signed</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>29.00</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>div21_21s_pipe</td>
<td>/</td>
<td>signed</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>29.00</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>geop12u_1</td>
<td>&gt;=</td>
<td>unsigned</td>
<td>10</td>
<td>10</td>
<td>13</td>
<td>2.69</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>geop12u_1</td>
<td>&gt;</td>
<td>unsigned</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>2.02</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>geop12u_1_1</td>
<td>&gt;</td>
<td>unsigned</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>2.02</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>geop20s_1</td>
<td>&gt;</td>
<td>signed</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>2.65</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>geop24s_1</td>
<td>&gt;</td>
<td>signed</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>2.99</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>geop8u_1</td>
<td>&gt;</td>
<td>unsigned</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>1.61</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>inct12u_1</td>
<td>++</td>
<td>unsigned</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>1.98</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>inct12u_11</td>
<td>++</td>
<td>unsigned</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>1.98</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>inct8u</td>
<td>++</td>
<td>unsigned</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>1.61</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>leop12u_1</td>
<td>&lt;=</td>
<td>unsigned</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>2.59</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>log12u_1</td>
<td>&lt;</td>
<td>unsigned</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>2.04</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>log20s_1</td>
<td>&lt;</td>
<td>signed</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>2.69</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>log24s_1</td>
<td>&lt;</td>
<td>signed</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>3.01</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>sub12u</td>
<td>-</td>
<td>unsigned</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>2.62</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>sub16s</td>
<td>-</td>
<td>signed</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>2.34</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>sub20s_16</td>
<td>-</td>
<td>signed</td>
<td>16</td>
<td>16</td>
<td>21</td>
<td>2.66</td>
<td>-</td>
<td>3</td>
</tr>
</tbody>
</table>
{ 
if (red==maxhsv&&delta!=0) 
{ 
  h= (green-blue)/delta;
}
else if(green==maxhsv&&delta!=0) 
{ 
  h=2+ (blue-red)/delta;
}
else if(blue==maxhsv&&delta!=0) 
{ 
  h=4+ (red-green)/delta;
}
}

During the conversion between RGB values to HSV domain, there will be several division operation. Thus, the system will generate dividers to accomplish such operation. As is known to all, doing division operation on hardware will consume large resources and time because hardware cannot handle decimals. Therefore, these division operations will cause long time delay. On the other hand, if the expression is very complicated, it will also increase the delay time.

In this case, since we must do HSV conversion and the conversion formula is defined, such division operation is unavoidable. Since the critical path only exceeds 1.92ns, it might not affect the real result very much. And through the real hardware test, maybe the delay is tolerable.

### 3.3.12. Summarize the algorithm designing on CWB

CWB provides the designer an easier way for design hardware projects. One shining spot of it is it supports using ANSI-C design flow and can automatically transfer the C code to HDL. It will largely shorten the design period and easier for the use to do any modification of the source code. But, we still meet with a lot problems during the using of CWB and algorithm design. Here we will make some analysis and explain.

- **Difficult to write hardware C program**

  As a rookie of using CWB, we spent long time to modify the general ANSI-C code to standard hardware C program. Unlike the general software C writing which can allocate memory, use dynamic range for “for loop”, dynamic storage of using array and easy use of pointers, hardware C writing is a bit more strict and not very comfortable to use.

- **Adapt to hardware design thought**

  Designing the algorithm for hardware is totally different from designing the algorithm for software. There should not be complicated logical operation existed in the algorithm or the synthesizer cannot properly map the algorithm to dedicated adder, subtractor, register, multiplexer, etc. On the other side, if too much “for-loop” is used, the area of the design will be very large which will increase the work load.
writing the source code, one should not only consider the function of the algorithm but also be aware of the hardware resource and component allocation. Thus, in order to design an efficient hardware algorithm, one must bear in mind that use the most simplified mind to design the method and keep trying until the algorithm become both stable and efficient.

➢ the original algorithm for number-plate recognition

Originally, we write the full algorithm for both number-plate detection and content recognition within the whole source code. The algorithm can not only detect where the number-plate is but also figure out the content within the number-plate. Unfortunately, we just think the way as doing software design and ignored the hardware constraint. Every time we did the synthesis process, it will go into endless synthesis and consume a very long time (several hours) but never give out the result.

One main problem of it is the recognition part uses a lot “for-loop” and try to create several buffers to load the content and the standard templates. In hardware, if “for-loop” is used, the synthesis will only copy the content of “for-loop” for the defined times which will drastically increase the resource and area. Furthermore, since the test picture will be changing, which means that all buffer size and “for-loop” times will be dynamic, it will lead the synthesis process to a disaster. Therefore, also the whole algorithm works well when doing SW simulation, it never works on the hardware side. So we have to simplify the algorithm to make sure that it can work out on the real hardware.

In order to solve this problem, we change the algorithm to be pixel-based, which can largely improve the process speed and utilize the pipeline scheduling mode perfectly. Each valid cycle, the module will read in one pixel value and do several judgment and process steps and then output the result to later modules. According to this way, we do not need any buffer to load data so that it will improve the design’s efficiency.

The full algorithm for number-plate detection and recognition can be seen in Appendix.

➢ Verification success does not mean true success

CWB provides two ways to do verification: Cycle-accurate simulation and RTL simulation. Both two ways will generate output files and such files will be compared with the result of SW simulation. If the results are the same, theoretically, the design is correct. But in reality, it doesn’t guarantee the success on the true board. The debugging and testing process on the real FPGA board will be discussed in the following chapter. Usually, when the code cannot work properly on the real board, the designer needs to go back to CWB to check or add more test signal to check where the problem is and modify the source codes, do the verification steps again.
Chapter 4

4. Image processing implementation on FPGA

4.1. Number-plate detection system key module structure

Key modules of the whole system is structured as in figure 29. Generally speaking, the whole system undergoes several stages include optical signal capture through the camera sensor, raw data collecting through CCD Capture module, data format transform to RGB through RAW2RGB module, feature detection function through Detect module, data storage and re-generation through SDRAM module, visible output display through VGA module and Ethernet data transmission through Data Transmission module. Other auxiliary modules like the I2C module is for adjusting the zoom in / zoom out and light exposure of the camera, Segment module is for displaying the counted number of the frames and the Reset module is for reset the whole system.

4.2. TRDB_D5M camera brief introduction

The camera we use is TRDB_D5M camera which can perfectly compatible with the DE2_115 FPGA board. The camera provides high-definition input image quality. Here is the brief explanation of the pixel data format of the TRDB_D5M camera.

**4.2.1. Pixel Array Structure**

The TRDB-D5M pixel array consists of a 2,752-column by 2,004-row matrix of pixels. Each pixel’s address is its column and row coordinators. It should be noted that (column 0, row 0) represents the upper-right corner of the entire array [16].

The format of the array matrix can be seen in the below figure. The white region represents the active image range with 2592 column by 1994 row. The white region is the default output image area. Surround
the white area, there are yellow pixels called boundary region. They are also active. Such boundary area is used to avoid edge effects. The dark region cannot be seen in the output image. The optically black column and rows can be used to monitor the black level [16].

![Figure 31 Pixel array description [11]](image1)

![Figure 32 Pixel color pattern detail (top right corner) [11]](image2)

4.2.2. Output data format (default mode)

The TRDB-D5M image data is read out in a progressive scan. As is shown below, the active data is read out and surrounded by both horizontal and vertical black pixels.
4.2.3. Output data timing

The output image data are divided into frames according to the different valid signals. These frames can be further divided into lines. As is defined in the active region, the sensor output active image with 2592 columns and 1994 rows for each frame. To identify the boundaries between frames and lines, two signals are used. They are FRAME_VALID and LINE_VALID.

The usage of PIXCLK is like a clock to latch the data. For each PIXCLK cycle, one 12-bit pixel datum is sent out from DOUT pins. The pixel is only valid when both FRAME_VALID and LINE_VALID signals are valid. When FRAME_VALID signal is negative, this represents the vertical blanking. Similarly, when LINE_VALID is negative, it is horizontal blanking.

4.3. System modules

Since the whole system will experience the process of pixel value collecting, data format converting, RGB value constructing, number-plate detection, SDRAM memory buffer, output data reconstructing and VGA displaying, it is a relevantly complicated system that it transfer the optical signal to digital signal and does a series signal processing as well as data format construction operations.

In this paragraph, we will emphasis on how every module within the system is working, including explaining the function of each module’s key input/output ports, how data format is constructed and converted, etc.
4.3.1. CCD_Capture module

There is an external camera inserted through the GPIO module on the FPGA board. The main function of the CCD_Capture module is to get the row material and information of the picture. In this first step, the optical information will be converted to electronic signal and then transferred to digital signal so that the hardware device can conduct all the later progress.

Key Module ports:

.iDATA(rCCD_DATA)
This port get the original data which is collected from the camera sensor without any process. The length of the data is 12bits. According to different valid signal, this data will be assigned to a register to construct the raw data format. Simply, when the line valid data is positive, then the original data will be assigned. Otherwise, the output data will be 0. Through this way, the valid image area can be presented.

.iFVAL(rCCD_FVAL)
.iLVAL(rCCD_LVAL)
These two input signals shows when the signal is valid so that the pixel can be read. Only when both frame valid signal and line valid signal are positive, the pixel value is valid.

.iCLK(~D5M_PIXLCLK)
This input clock represents the working cycle of the camera, with this clock, all the data read/write operation can be controlled.

.oDATA(mCCD_DATA)
This port output the raw data, which is assigned the by the iDATA, to the next module. The length of the raw data is 12 bits. This series of data can effectively separate where the valid pixel value is.

.oX_Cont(X_Cont)
.oY_Cont(Y_Cont)
.oFrame_Cont(Frame_Cont)
These three signals gives out the position information of each pixel. The information includes the x coordinate, y coordinate of the pixel and the frame number. Such position information is counted inside the module. These position information is important for the later on detection process.

4.3.2. RAW2RGB module

Obvious, this module’s function is to convert raw pixel value to RGB value. Originally, since it is a CCD sensor, this kind of camera samples the spectrum of visible light using three or more filters. Each location on the CCD captures one sample of the color spectrum. This gives us a “mosaic” of samples.
Each pixel from the raw data which directly comes from the sensor chip only has one color, red, green or blue. In order to represent one colorful pixel, the missing value of the pixel should be properly interpolated.

As mentioned before, the data format output from the sensor is in a Bayer Pattern. The character of this format is that the odd row only has green and red pixels, the even row only has blue and green pixels, the odd column only has green and blue pixels and the even column only has red and green pixels.

The main conversion process includes two operations: row buffer and pipeline scheduling mode. The main method is to convert the former buffered row Bayer color patterns to a 12 bits RGB value in real-time and conduct proper down-sampling so that the converted data can be processed later.

To accomplish this task, first step is to call a megafuction called “altshift_taps”. The altshift_taps is very often used in image processing program as data moving registers. In this conversion process, 2 row’s pixel data will be processed simultaneously. Thus, altshift_taps megafunction is used as creating buffers to provide 2 taps of pixel data. Each tap’s length is 1280, equals to each row’s length.

Four variables are mainly used in this conversion algorithm:

- \text{mDATA}_0: \text{the pixel of the } M^{th} \text{ row, } N^{th} \text{ column}
- \text{mDATAd}_0: \text{the pixel of the } M^{th} \text{ row, } (N - 1)^{th} \text{ column}
- \text{mDATA}_1: \text{the pixel of the } (M + 1)^{th} \text{ row, } N^{th} \text{ column}
- \text{mDATAd}_1: \text{the pixel of the } (M + 1)^{th} \text{ row, } (N - 1)^{th} \text{ column}
These four Bayer pattern values can form an mDATA_0-centered template, including one red value, one blue value and two green values. One pixel’s RGB value can be constructed through these four values by the following rules:

The RED value will be equal to mDATA_0’s red value, the BLUE value will be equal to the mDATA_0’s blue value and the GREEN value will be equal to the mean value of two green values (G1 and G2). In order to get the mean value of GREEN value, the higher 12 bits of (G1+G2) will be reserved.

\[
\text{Red value} = R; \\
\text{Blue value} = B; \\
\text{Green value} = \frac{G1+G2}{2};
\]

Therefore, each pixel’s RGB value can be obtained through this way. The output of the RAW2RGB module will be the converted R, G, B value of each pixel and the valid signal for controlling the latter modules.

Output signals:

- output [11:0] oRed;
- output [11:0] oGreen;
- output [11:0] oBlue;
- output oDVAL;

### 4.3.3. Detection module

After the pixel value being converted to standard RGB values, later on number-plate detection can be performed. This module’s function is designed and verified by the former CWB tool. To create the module, it is very simple. Just copy the automatically generated Verilog codes from CWB side and paste...
it to the quartus project. Then, setting proper inputs and outputs of this module so that this module can be inserted into the whole project.

Input signals:

\[
\begin{align*}
\text{input } [11:0] & \quad \text{input\_row\_a00} \\
\text{input } [11:0] & \quad \text{input\_row\_a01} \\
\text{input } [11:0] & \quad \text{input\_row\_a02} \\
\end{align*}
\]

These three signals represent each pixel’s RGB value.

\[
\begin{align*}
\text{input } [11:0] & \quad \text{OX} \\
\text{input } [11:0] & \quad \text{OY} \\
\end{align*}
\]

These two signals represent each pixel’s horizontal and vertical coordinates. The coordinates of the pixels can help to locate the position of the number-plate, drawing a red block on the detected area.

\[
\begin{align*}
\text{input} & \quad \text{CLOCK} \\
\text{input} & \quad \text{RESET} \\
\end{align*}
\]

The clock and reset inputs will control the read/write operations of this module.

\[
\begin{align*}
\text{output } [11:0] & \quad \text{output\_row\_a00} \\
\text{output } [11:0] & \quad \text{output\_row\_a01} \\
\text{output } [11:0] & \quad \text{output\_row\_a02} \\
\end{align*}
\]

The detection module will output the processed pixel value to the next module. From the input side, it is color image, from the output side, it is black-white results which can largely reduce the data size. As defined by the algorithm, the number-plate background area will be highlighted as white and all the other parts of the image will be all black. Furthermore, a red block will be displayed on the edge of the number-plate, showing the real-time location.

Theoretically, since the Verilog codes are automatically converted from the ANSI-C source codes, the function of the module should be exactly the same as we verified on CWB tool.

### 4.3.4. SDRAM control module

The SDRAM control module is used as data buffer within the whole system, mainly targeting at reconstruct the data form and send it to the VGA side. As is known before, there are three key outputs from the detection module are separated data. How to efficiently store such huge amount of data and improve the read/write speed of the system is the main concern in this module.

On this DE2_115 board, luckily, two SDRAM are included, 16 bits for each. Therefore, combing the two SDRAM memories can provide 32 bits data process ability. This will largely increase the read/write efficiency and handle the large data throughput of this project.

In order to efficiently use two SDRAM chips together, “FIFO” megafunction is called. Four FIFOs are generated to conduct the write/read operations. Two FIFOs are for writing and the other two are for
reading. According to different require commands, read and write commands can be operated alternatively.

For write side, which is also the input side of the SDRAM control module, data are input in the following format:

FIFO 1 write side: \texttt{.WR1\_DATA (\{1'b0, sCCD\_G[11:7], sCCD\_B[11:2]\})}

The input data for FIFO 1 is composed of 16bits, the highest bit is 0, then following the high-order 5 bits of green color and high-order 10 bits of blue color.

FIFO 2 write side: \texttt{.WR2\_DATA(\{1'b0, sCCD\_G[6:2], sCCD\_R[11:2]\})}

The input data for FIFO 2 is composed of 16bits, the highest bit is 0, then following the low-order 5 bits of green color and high-order 10 bits of red color.

Through this way of data format construction, we can use two data flow to properly store 3 colors of the pixel.

FIFO 1 and FIFO 2 then write the data into different address:

For FIFO 1 write side: \texttt{.WR1\_ADDR(0)}

For FIFO 2 write side: \texttt{.WR2\_ADDR(23'h100000)}

As is shown below, when the data is input from the former module, it is all 16-bits long. Then, using FIFO mega-function, two pixel’s data information is combined together and then be written into the SDRAM memory each time. Thus, it largely improves the bandwidth of the system.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure38.png}
\caption{Illustration of pixel value combination for SDRAM processing}
\end{figure}
Similarly, for the read side, two read FIFOs are created. Each time when the latter VGA module is asking from SDRAM, this FIFO module will read out the data according to the address. The output data width is then reduced to 16-bits.

FIFO 1 read side: .RD1_DATA(Read_DATA1)
FIFO 2 read side: .RD2_DATA(Read_DATA2)
For FIFO 1 read side address: .RD1_ADDR(0)
For FIFO 2 read side address: .RD2_ADDR(23'h100000)
Through the SDRAM control module, it offers the way of how we will store data, including the format of storage, the address of data and other control signals. Moreover, it guarantees the read side for grabbing the data out of memory accurately as well as efficiently.

4.3.5. VGA display module

According to VGA’s pin standard, it has 15 pins for signals. Details are shown as below:
VGA connection description:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>analog red, 0-0.7V</td>
<td>DAC output</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>analog green, 0-0.7V or 0.3-1V (if sync-on-green)</td>
<td>DAC output</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>analog blue, 0-0.7V</td>
<td>DAC output</td>
</tr>
<tr>
<td>4</td>
<td>EDID Interface</td>
<td>function varies depending on standard used</td>
<td>no connect</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>general</td>
<td>GND</td>
</tr>
<tr>
<td>6</td>
<td>GND</td>
<td>for R</td>
<td>GND</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>for G</td>
<td>GND</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>for B</td>
<td>GND</td>
</tr>
<tr>
<td>9</td>
<td>no pin</td>
<td>or optional +5V</td>
<td>no connect</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
<td>for h_sync and v_sync</td>
<td>GND</td>
</tr>
<tr>
<td>11</td>
<td>EDID Interface</td>
<td>function varies depending on standard used</td>
<td>no connect</td>
</tr>
<tr>
<td>12</td>
<td>EDID Interface</td>
<td>function varies depending on standard used</td>
<td>no connect</td>
</tr>
<tr>
<td>13</td>
<td>h_sync</td>
<td>horizontal sync, 0V/5V waveform</td>
<td>FPGA output</td>
</tr>
<tr>
<td>14</td>
<td>v_sync</td>
<td>vertical sync, 0V/5V waveform</td>
<td>FPGA output</td>
</tr>
<tr>
<td>15</td>
<td>EDID Interface</td>
<td>function varies depending on standard used</td>
<td>no connect</td>
</tr>
</tbody>
</table>

Table 1 VGA Connector Pinout and Signals [17]

The main challenges for displaying correct images on the VGA display contain two parts: sending R,G,B values to its corresponding pin and sent correct synchronize signals to the ports. First, to send the relevant R,G,B value to the correct pin, the data format is defined as below:

For input side of the module,

\[ \text{xRed}(\text{Read\_DATA2}[9:0]) \]
\[ \text{xGreen}([\text{Read\_DATA1}[14:10],\text{Read\_DATA2}[14:10]]) \]
\[ \text{xBlue}(\text{Read\_DATA1}[9:0]) \]

When the VGA module is reading the data from the SDRAM side, it combines all the RGB values together at this step. As is defined before, Read\_DATA1 includes the green and blue data.
If the horizontal timing signal and the vertical signal are both valid, output the input data to the output side.

\[ oVGA_R(o_{VGA_R}) \]
\[ oVGA_G(o_{VGA_G}) \]
\[ oVGA_B(o_{VGA_B}) \]

The signal timing diagram is shown as below:

![Signal Timing Diagram](image)

**Figure 42 signal timing diagram**

### 4.3.6. I2C_CCD_Config module

The function of this module is to perform camera exposure and zoom in/zoom out adjustment. Furthermore, user can use keys on the FPGA board to take a shoot of the image. With such functions, user can easily adjust the camera parameters so that even in bad testing environment like a dark place, this system can still work.

Key components of the modules:

- **iCLK(CLOCK2_50)**,  // input clock
- **iRST_N(DLY_RST_2)**,  // reset signal
- **iEXPOSURE_ADJ(KEY[1])**,  // light exposure adjustment control of level
- **iEXPOSURE_DEC_p(SW[0])**,  // light exposure adjustment control of increase or decrease
- **iZOOM_MODE_SW(SW[16])**,  // zoom in or zoom out control
\[ \text{I2C\_SCLK(D5M\_SCLK)}, // \text{output clock} \]
\[ \text{I2C\_SDAT(D5M\_SDATA)} // \text{input and output data port (inout format)} \]

Table 2 explains the function of each key or switch and how to achieve multifunction through the FPGA board like adjust light exposure, zoom in, zoom out, take picture, reset, etc.

<table>
<thead>
<tr>
<th>Component</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY[0]</td>
<td>Reset circuit</td>
</tr>
<tr>
<td>KEY[1]</td>
<td>Set the new exposure time (use with SW[0])</td>
</tr>
<tr>
<td>KEY[2]</td>
<td>Trigger the Image Capture (take a shot)</td>
</tr>
<tr>
<td>KEY[3]</td>
<td>Switch to Free Run mode</td>
</tr>
<tr>
<td>SW[0]</td>
<td>Off: Extend the exposure time</td>
</tr>
<tr>
<td></td>
<td>On: Shorten the exposure time</td>
</tr>
<tr>
<td>SW[16]</td>
<td>On: ZOOM in</td>
</tr>
<tr>
<td></td>
<td>Off: Normal display</td>
</tr>
<tr>
<td>HEX[7:0]</td>
<td>Frame counter (Display ONLY)</td>
</tr>
</tbody>
</table>

*Table 2 the functional keys of the digital camera demonstration*

4.3.7. Sdram_pll module

This module is mainly used for generating different clocks of different frequencies so that can meet with different modules’ clock period requirement. As is specified in the manual of DE2_115 development board, the default clock of the board is 50 MHz. One megafuction called “ATLPLL” is used to achieve the function. A PLL is a feedback control system that automatically adjusts the phase of a locally generated signal to match the phase of an input signal [18]. The usage of PLLs is to generate an oscillator frequency that can meet the requirement of the input signal. Through the “ATLPLL” megafuction setting interface, user can specify the input clock of the system and define the characteristics of the generated output clock signals. The setting tool interface is shown as below.
For this design, five clock signals are generated as the input signal of different modules. Below are the explanation of these five clock signals.

<table>
<thead>
<tr>
<th>clock name</th>
<th>frequency value</th>
<th>clock usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>100 MHz</td>
<td>sdram control clock</td>
</tr>
<tr>
<td>c1</td>
<td>100 MHz</td>
<td>for sdram use</td>
</tr>
<tr>
<td>c2</td>
<td>25 MHz</td>
<td>output clock for the GPIO port</td>
</tr>
<tr>
<td>c3</td>
<td>25 MHz</td>
<td>if defined 640*480 of VGA</td>
</tr>
<tr>
<td>c4</td>
<td>40 MHz</td>
<td>if defined 800*6000 of VGA</td>
</tr>
</tbody>
</table>

Table 3 ATPLL generated clock signals

### 4.3.8. SEG_LUT_8 module

The function of the SEG_LUT_8 module is to display how many frames has been displayed on the VGA. The segments on the FPGA board will display the counted number in HEX modes to show how many frames are presented.

### 4.3.9. Reset delay module

This module is to reset all the module to its initial status. If key 0 is pressed, the whole system will be reset.
4.4. Quartus II implementation procedure

4.4.1. Software implementation

First of all, one project is created by the user and some proper settings such as FPGA board choice is done before any design of the system.

By using Verilog HDL language to describe the whole design, each module is individually written in one Verilog file. As mentioned before, according to the function of the design, several modules such as CCD capture module, RAW to RGB module, detection module, SDRAM control module and VGA modules are written in pure Verilog HDL file and considered as components of the design. Other modules like PLL, FIFO modules are using megafunctions to achieve the function. The detection module is using HLS tool to automatically generate the Verilog codes which can be implemented in the project.

The functions and design methods of each module is discussed and explained in the former chapter. After each module is written, one top-layer module is set. This file describes the connection relationships between each modules, specifying the logic structure of the system and includes some control commands to unite all the modules together. Each module is considered as sub-layer of the system and perform different tasks individually. As showed below, file “DE2_115_CAMER A” is the top-layer of the project. Within it, different functional modules are included.

Figure 44 design architecture includes top module and each function modules
When the Verilog HDL codes are written properly, configuring the pins of the FPGA board is done. One useful tool can be used is the “pin planner” interface within the Quartus II software. According to the manual of the FPGA board, each pin of the chips on the board is pre-defined and user should assign pin resources carefully to avoid confliction. For example, user should specify several information of the pin such as the direction of the pin (input/output), location of the pin, IO standard, etc. Details of the pin assignment can be seen as below:

**Figure 45 pin planner for configuring pins of the FPGA board**

### 4.4.2. Compilation

After all the design and preparation are done, the project needs to be compiled to check if any syntax error exits, operate placing and routing operation, examine timing problems and generating .qpf file which can be downloaded to the FPGA board to execute real operations.

For the compilation stage, the following procedures are completed:

**Figure 46 compilation process and results**
During the compilation process, the consulting window will display which step is undergoing and if there is any error, the window will show the warning information, error message and terminate the compilation process if any error is found. According to the error message showed in the consulting window, it is easy for the designer to find out the problem of the project and try to modify the design. For example, if syntax error is found, usually there are some “grammar” mistakes or format mistakes in the Verilog HDL code. On the other hand, some structural mistakes will also be reported like address confliction. Generally, one project needs to be re-compile for several times until all the things are correct.

4.4.3. Compilation report

After full compile is done, a window will show the successful message to the user. This window will also tell the user how much warnings are existed in the compiled project.

Besides, one compilation is automatically generated to show all the details of the project and the performance of the design. Details of the compilation report can be seen as below.
As is shown in the above figures, key index in the compilation reports are:

- Total logic elements: 4,508 / 114,480 ( 4 % )
- Total registers: 2335/114,480(2%)
- Total pins: 428 / 529 ( 81 % )
- Total memory bits: 75,797 / 3,981,312 ( 2 % )
- Timing information of the entire system is listed as below:
### Summary (Setup)

<table>
<thead>
<tr>
<th>#</th>
<th>Clock</th>
<th>Slack</th>
<th>End Point TNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>udp_checksum</td>
<td>my_pll</td>
<td>auto_generated</td>
</tr>
<tr>
<td>2</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
<tr>
<td>3</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
<tr>
<td>4</td>
<td>CLOCK2_50</td>
<td></td>
<td>13.337</td>
</tr>
<tr>
<td>5</td>
<td>udp_checksum</td>
<td>my_pll</td>
<td>altpll_component</td>
</tr>
</tbody>
</table>

**Figure 51 Setup timing report**

### Summary (Hold)

<table>
<thead>
<tr>
<th>#</th>
<th>Clock</th>
<th>Slack</th>
<th>End Point TNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
<tr>
<td>2</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
<tr>
<td>3</td>
<td>CLOCK2_50</td>
<td></td>
<td>0.402</td>
</tr>
<tr>
<td>4</td>
<td>udp_checksum</td>
<td>my_pll</td>
<td>altpll_component</td>
</tr>
<tr>
<td>5</td>
<td>udp_checksum</td>
<td>my_pll</td>
<td>altpll_component</td>
</tr>
</tbody>
</table>

**Figure 52 Hold timing report**

### Summary (Recovery)

<table>
<thead>
<tr>
<th>#</th>
<th>Clock</th>
<th>Slack</th>
<th>End Point TNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
<tr>
<td>2</td>
<td>CLOCK2_50</td>
<td></td>
<td>11.999</td>
</tr>
<tr>
<td>3</td>
<td>u6</td>
<td>altpll_component</td>
<td>auto_generated</td>
</tr>
</tbody>
</table>

**Figure 53 Recovery timing report**
4.4.4. Resource consume of different modules

The bellowing figure gives out the resource consume of each module, which directly reflect the area occupation of the designed system.

4.4.4.1. Logic Cells resource occupation
4.4.4.2. Dedicated logic registers resource occupation
4.4.4.3. Memory bits

Memory bits

4.4.4.4. LUT-Only LCs

LUT-Only LCs
4.4.5. Register-Only LCs

When the project passed the compilation stage, theoretically it can be downloaded to the FPGA board to check if all the functions are correct or not. It uses USB blaster (on board) to for programming and downloading the design to the board. Actually, after the compilation, one .sof file is generated. This is the
file which is downloaded to the FPGA board through the USB blaster. By clicking “tools” and “programmer”, one interface will be displayed for downloading.

![Programmer tool for downloading qpf file to the FPGA board](image)

When the “progress” comes to 100%, it indicates that all the downloading is finished. Then, connecting the board to the VGA displayer and pre-plug in the camera, the system will work for number-plate detection.

For this design, at first there is nothing showed on the VGA displayer. At this moment, how to debug the system is very important. In this design, we choose to use the embedded “Signal Tap II Logical Analyzer” tool to do the debugging and testing.

Within the signal tap tool, user can choose which signals he/she is interested for tracking. For example, in this design, in order to test whether the number-plate area color is detected or not, it is essential to monitor the HSV value of each pixel. Since the environment including luminance, chrominance and reflection will affect the actual color value a lot, it is very important to test whether the pre-defined HSV threshold adapt to the real environment. One reasonable method to amend the threshold is to use one paper which color is the standard “yellow” color of the number-plate to test the system. Since every second, there is millions of data processed by the system and the signal tap will show all the real-time changes. The user should be patience enough to catch the right moment.
According to the actual HSV value tracked from the signal tap, the problem is found. The pre-defined HSV threshold doesn’t match the actual environment. Thus, the threshold should be carefully amended. Each time when the threshold is reset, we need to go back to CWB to do re-verification and re-generate the .v file and insert them to the current project. Through several times modification and testing, the threshold will be more accurate to capture the number-plate back ground color. Finally, the system can detect the number-plate accurately and display them on the VGA displayer.
Chapter 5

5. Ethernet data transmission implementation on FPGA

5.1. Brief introduction of data transmission designing method

Due to the large amount of data collected from the image processing segment, how to send the processed data out to be received by the external machine is another topic of this design. In the former chapter, VGA displayer is used for showing the image processing results of the design, but all the data cannot be saved for further process. In this chapter, we will discuss how to efficiently send out the processed data out of the FPGA board to the external environment and show the result of the design.

Since this project is part of an Internet of Thing project, using Ethernet to send out the data is always the first choice. On the DE2_115 board, there is an “88E1111” chip for Ethernet data transmission.

The Alaska Ultra 88E1111 Gigabit Ethernet Transceiver is a physical layer device for Ethernet 1000BASE-T, 100 BASE-TX, and 10BASE-T applications [20]. It is manufactured using standard digital CMOS process and contains all active circuitry required to implement the physical layer functions to transmit and receive data on standard CAT 5 unshielded twisted parity [20].

The 88E1111 device offers several data transportation ways such as Gigabit Media Independent Interface (GMII), Reduced GMII (RGMII), etc. In this project, we choose to use RGMII data transportation method to conduct the Ethernet communication. Therefore, the standard transmission rate is 125M.

Since the processed data only has two colors, white or black, it is easy to reconstruct the data format when we need to send them out. From the VGA module side, we already has each pixel’s RGB value and since only two colors are presented, we can only grab the high-order bits and dump the other low-order bits. Two FIFOs are need for alternatively conducting read or write commands.

assign fifo_wdata2 = {VGA_B[7:6],VGA_G[7:5],VGA_R[7:5]};
assign fifo_wdata1 = {VGA_B[7:6],VGA_G[7:5],VGA_R[7:5]};

Generally, when using Ethernet to send out data, it uses bytes as unit. Since 1 byte = 8 bits, we reconstruct the transmission data to 8 bits, which will be convenient for data transmission. Each byte includes high-orders of the pixel’s RGB value. On the other side, when one row’s pixel has been written into the FIFO module, the module will output a signal, asking the data transmission module to transmit the data.

In order to successfully transmit the data out through the Ethernet port, three protocols are used in this design: UDP protocol, IP protocol and MAC protocol.
5.2. UDP protocol

5.2.1. UDP protocol brief introduction

UDP protocol refers to User Datagram Protocol, which uses a simple connectionless transmission model with a minimum of protocol mechanism [21]. Since it does not have any handshaking dialogues, its reliability is quite low. Furthermore, unlike other transmission protocol like TCP, UDP doesn’t have any protection of package delivery, order and duplicate. With UDP, the sender just send out data through the IP network without pre-setting any special communication channels. Thus, data sent from UDP protocol might not be very reliable but relevantly more efficient than other protocols if user do not care very much on data transmission accuracy. In this design, due to the large data throughput generated from the image processing side, using UDP protocol to do the data transmission is both easy and efficient. Because it is a real-time system, using UDP can also meet with the need of sending out data timely.

5.2.2. UDP packet structure

Through UDP protocol, it offers the application of heard and payload multiplexing and integrity
verification, but transmission reliability is not included. If user cares about the reliability of data transmission, this function should be included in user’s application by user self.

As is shown above, the UDP header is composed of four fields, each field’s length is 2 bytes (16 bits).

**Source port number**
This field shows the information of sender’s port. If not used, it is zero. The port number type is based on the property of the source. It will be an ephemeral port number if the source host is the client. On the other hand, it will be a well-known port number if the source host is the server [22].

**Destination port number**
This place represents the receiver’s port and is must needed. The description of the port number is similar to the source port number.

**Length**
In this place, it specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes. The field size sets theoretical limit of 65,535 bytes for a UDP datagram. The practical limit for the data length which is imposed by the IPV4 protocol is 65,507 bytes.

**Checksum**
In this field, it contains the information for error-checking of the header and data. If no checksum is generated by the transmitter, the filed uses all zeros [22].

5.2.3. **Checksum computation**
According to RFC 768, the method of checksum computation is clearly defined. As is defined, the length of the checksum is 16 bits long. In this design, the UDP is run over IPV4 protocol. Thus, the checksum is computed using a “pseudo header” that contains some of the same information from the real IPV4 header [23]. The source and destination addresses are the same in the IPV4 header. Within the length field, it represents the sum length of UDP header and data. UDP checksum computation is not mandatory for IPV4. If no checksum is needed, it should be zero.
5.3. IP protocol

5.3.1. IP protocol instruction

As is known to all, IP protocol is the basic communications protocol which can relay data packages through network. Though the function of routing, IP protocol ensures nowadays networking framework and sets up today’s Internet world [24].

Main task of IP protocol is to deliver packets from the original place to its destination across one or more IP networks according to the IP addresses in the packet headers. In order to achieve this task, IP protocol defines the packet structures that data being sent are encapsulated standardly. Furthermore, the protocol also set the principle of how to address and label the datagram according to its source and destination information.

5.3.2. Packet structure – IPV4

One IPV4 packet is consist of two sections. One for header and another for data. The IPV4 packet consists of 14 fields. The first 13 fields are required but the last one is optional. Details of the IPV4 header format is displayed as below.

**Version**

The first four bits within the header area indicates the version of IP packet. Since it is an IPV4 packet, the value should obvious be 4.

**Internet Header Length (IHL)**
This field specify the size of the header [25]. The field’s minimum value is 5, equals to 20 bytes. The maximum value will be 60 bytes.

**Differentiated Services Code point (DSCP)**
The new technologies like real-time data streaming uses the service of DSCP [26].

**Explicit Congestion Notification (ECN)**
This field defines and allows end-to-end notification of network congestion without dropping packets [27]. ECN is optional and this feature is only used when both endpoints support it and have the willing to use it.

**Total length**
This field explains the total length of the packet in bytes, including header and data. The minimum length of the packet is 20 bytes and the maximum length is 65,535 bytes.

**Identification**
This field is primarily used for uniquely identifying the group of fragments of a single IP datagram [28].

**Flags**
This field used for controlling or identifying fragments. Usually, it is 3 bits long.
Bit 0: Reserved; must be 0.
Bit 1: Don’t Fragment (DF)
Bit 2: More Fragments (MF)

**Fragment offset**
This fragment offset shows the offset of a particular fragment according to the first unfragmented IP datagram [23].

**Time to live (TTL)**
This field is used to avoid letting datagrams to be existing on an internet forever. When the TTL becomes zero, the router will discard the packet and send an ICMP time exceeded message to the sender.

**Protocol**
This file defines the protocol used in the data portion of the IP datagram [28].

**Header Checksum**
This filed is used to check header error. Every time when a packet arrives at a router, the router will calculate the checksum of the header and then compare the result with the header checksum. If these two number does not match each other, then this packet will be discarded.

**Source address**
This field explains the sender of the packet in type of IPV4 address. This address might change in transit by a network address translation device [28].

**Destination address**
This field specifies the receiver of the packet. Since the source address might change in transit, this address might be changed [28].

Options
Usually, this field is not used.

5.4. MAC protocol

5.4.1. MAC protocol introduction
MAC protocol refers to media access control data communication protocol. It is a sublayer of the data link layer and is also the bottom layer of the seven-layer OSI module of computer networking. The MAC sublayer is like a bridge between the logical link control (LLC) sublayer and the network’s physical layer [29]. This channel can provide unicast, multicast or broadcast communication service.

5.4.2. Ethernet frame structure
Generally, Ethernet packet is a data packet on the Ethernet, which transports an Ethernet frame as payload [30]. Each frame is preceded by a preamble and start frame delimiter (SFD). The start of an Ethernet frame is the Ethernet header. The first two fields of the package are destination and MAC address. The middle section of the frame is payload data including any header for other protocols carried in the frame. For example, Internet Protocol IP). At the end of the frame, there is a frame check sequence (FCS), which is a 32-bit cyclic redundancy check (CRC check) [30]. The structure of the Ethernet frame can be seen as below.

![802.3 Ethernet packet and frame structure](image)

5.4.3. Ethernet II frame structure
In this design, we choose to use Ethernet II type Internet frame to construct the data package. It is the most commonly used type currently. Ethernet II frame structure defines the two-octet EtherType field in an Ethernet frame. At the beginning are destination and source MAC addresses. Details of the Ethernet II frame structure is listed as below.
5.5. Ethernet package structure based on DE2_115 board

According to the standard UDP, IP, MAC and Ethernet packet structure, data frame which is running on the DE2_115 board can be created and defined. The table below listed the user defined Ethernet frame which is used in this project to transmit the real-time data through the Ethernet port.

Table 4 User defined Ethernet frame structure

<table>
<thead>
<tr>
<th>Byte number</th>
<th>Byte length</th>
<th>Value</th>
<th>definition</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-6</td>
<td>7</td>
<td>h'55</td>
<td>preamble</td>
<td>Frame header 8 bytes</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>h'D5</td>
<td>delimiter</td>
<td></td>
</tr>
<tr>
<td>8-13</td>
<td>6</td>
<td></td>
<td>Destination MAC address Ethernet II type 14 bytes</td>
<td></td>
</tr>
<tr>
<td>14-19</td>
<td>6</td>
<td>Source MAC address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>h'08</td>
<td>Type IP=h'8808</td>
<td>IP header 20 bytes</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>h'88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>h'45</td>
<td>version/IP header length=20 bytes</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>h'00</td>
<td>unknown TOS</td>
<td></td>
</tr>
<tr>
<td>24-25</td>
<td>2</td>
<td>h'041c</td>
<td>total length of IP header and data=20+8+1024=0x41c</td>
<td></td>
</tr>
<tr>
<td>26-27</td>
<td>2</td>
<td>h'0000</td>
<td>adds one when on package is sent out</td>
<td></td>
</tr>
<tr>
<td>28-29</td>
<td>2</td>
<td>h'0000</td>
<td>FLAGS/offset</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>h'80</td>
<td>TTL</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>h'11</td>
<td>UDP protocol</td>
<td></td>
</tr>
<tr>
<td>32-33</td>
<td>2</td>
<td></td>
<td>IP header checksum</td>
<td></td>
</tr>
<tr>
<td>34-37</td>
<td>4</td>
<td></td>
<td>IP source address</td>
<td></td>
</tr>
<tr>
<td>38-41</td>
<td>4</td>
<td></td>
<td>IP destination address</td>
<td>UDP header 8 bytes</td>
</tr>
<tr>
<td>42-43</td>
<td>2</td>
<td></td>
<td>port source address</td>
<td></td>
</tr>
<tr>
<td>44-47</td>
<td>2</td>
<td></td>
<td>port destination address</td>
<td></td>
</tr>
<tr>
<td>48-49</td>
<td>2</td>
<td></td>
<td>UDP header checksum</td>
<td></td>
</tr>
<tr>
<td>50-1073</td>
<td>1024</td>
<td>data</td>
<td>1024 byte</td>
<td></td>
</tr>
<tr>
<td>1074-1077</td>
<td></td>
<td>CRC</td>
<td>4 byte</td>
<td></td>
</tr>
</tbody>
</table>

5.5.1. MAC header

The first 7 bytes of the frame is a preamble. The value is “h’55”. The eighth byte is a delimiter. The value is “h’D5”. Thus, the first eight bytes consist the frame header.

The 8th byte to the 13th byte is the destination address of MAC. The 14th byte to the 19th byte is the destination address of MAC. The 20th and the 21st byte are the Type IP. The value is “h’8808s”. This indicates that this is a Ethernet II type frame.
5.5.2. IP header

The 22nd byte of the frame includes the version and the length of the IP header. The length of IP header is 20 bytes. This defined value is “h’45”. The 23rd byte of the frame shows the Type of Service. It is defined as “h’00”, indicating that it is an unknown service. Byte 24 to byte 25 includes the total length of IP header and data. The value is “h’041c”. Byte 26 to byte 27 is the counter of package. When one package is sent, this value adds one. The initiate value is “h’0000”. Byte 28 to byte 29 indicates the FLAG/offset information. Byte 30th shows the Time to Live (TTL), which is set as “h’80”. The 31st byte shows which data transmit protocol it is. Since this design is based on UDP protocol, this value should be “h’11”. Byte 32 to byte 33 shows the checksum of the IP header. Detailed checksum method will be discussed later. Byte 34 to byte 37 indicates the source address of IP. Byte 38 to byte 41 indicates the destination address of IP.

5.5.3. UDP header

The 22nd byte of the frame includes the version and the length of the IP header. The length of IP header is 20 bytes. This defined value is “h’45”. The 23rd byte of the frame shows the Type of Service. It is defined as “h’00”, indicating that it is an unknown service. Byte 24 to byte 25 includes the total length of IP header and data. The value is “h’041c”. Byte 26 to byte 27 is the counter of package. When one package is sent, this value adds one. The initiate value is “h’0000”. Byte 28 to byte 29 indicates the FLAG/offset information. Byte 30th shows the Time to Live (TTL), which is set as “h’80”. The 31st byte shows which data transmit protocol it is. Since this design is based on UDP protocol, this value should be “h’11”. Byte 32 to byte 33 shows the checksum of the IP header. Detailed checksum method will be discussed later. Byte 34 to byte 37 indicates the source address of IP. Byte 38 to byte 41 indicates the destination address of IP.

5.5.4. Header checksum method

5.5.4.1. IP header checksum

The checksum is calculated by forming the ones' complement of the ones' complement sum of the header's 16-bit words. The result of summing the entire IP header, including checksum, should be zero if there is no corruption. At each hop, the checksum is recalculated and the packet will be discarded upon checksum mismatch [31].

For example, if the IP header (20 bytes) of one data packet is:

```
45 00 00 30 80 4c 40 00 80 06 b5 2e d3 43 11 7b cb 51 15 3d.
```

1) The checksum code is “b5 2e”. First replaced the checksum code with “0000”. It is changed to 45 00 00 30 80 4c 40 00 80 06 00 00 d3 43 11 7b cb 51 15 3d.

2) Then, calculate the sum of one’s complement.
3) After that, add the carry bit to the lower-order 16 bits. If the result is larger than 16 bits, keep calculating until the result length is smaller than 16 bits.

\[3 + 4ace = 4ad1\]

4) Make one’s complement

One’s component of 4ad1 should be 5b2e. So the checksum is 5b2e.

When the IP data package is received, the receiver side will do the IP checksum as follow:

1) Calculate the sum of 16 bit one’s complement.

2) Calculate one’s complement of 1)’s result to see if it is 0.

The received IP packet header is:

```
45 00 00 30 80 4c 40 00 80 06 b5 2e 11 7b cb 51 15 3d
```

Sum of one’s complement:

\[4500+0030+804c+4000+8006+b52e+d343+117b+cb51+153d = 3fffc\]

\[3 + 3fffc = ffff\]

One’s complement of “h’ffff” is 0. Thus, this packet is correct.

5.5.4.2. CRC checksum

The usage of CRC checksum is to detect accidental changes in digital networks. The CRC value is a redundancy and the algorithm is based on cyclic codes. A short check value is attached at the end of the blocks entering these systems, based on the remainder of a polynomial division of their contents [32]. On retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not much [32].

5.6. Ethernet data transmission module design

![Ethemet data transmission design flow](image)

5.6.1. Data format reconstruction module

The transmitted data will be reconstructed before it enters the final transmission session. As mentioned before, all the output pixels only have two color, black or white. This largely reduce the data amount
which is needed to be sent and ease the sending procedure. From the VGA side, the out data is 10 bits long. Since the Ethernet usually sends out data based on bytes (8 bits) type, we reconstructed the data format so that it is easier to be encapsulated in the data packet. Since the pixel only has two formats, 11111111 for white and 00000000 for black, thus using the higher order is reasonable for data reasonable. The reconstruction method is listed as below.

```verilog
assign fifo_wdata2 = {VGA_B[7:6],VGA_G[7:5],VGA_R[7:5]};
assign fifo_wdata1 = {VGA_B[7:6],VGA_G[7:5],VGA_R[7:5]};
```

After the data reconstruction process, they will be sent into to FIFOs. One FIFO is for writing and the other is for reading. These two FIFOs work alternatively thus can support the whole system’s read/write requirements.

**Reference codes:**

```verilog
myfifo myfifo1(
    .data(fifo_wdata1),
    .rdclk(fifo_rdclk),
    .rdreq(fifo_rdreq1),
    .wrclk(VGA_CLK),
    .wrreq(wrreq1),
    .q(fifo_rdata1),
    .rdusedw(rdusedw1),
    .wrempty(wrempty1),
    .wrusedw(wrusedw1)
);
```

### 5.6.2. Data-generation module

This module is the key component of the data transmission part. Within this module, all the MAC layer, IP layer and UDP layer is constructed. It works at the frequency of 125M, which is the standard gigabit Ethernet transmission rate. The inputs and outputs of the module is listed as below.

- `.clk(clk125m)` // input signal clock which is standard gigabit Ethernet transmission rate
- `.rst_n(locked)` // reset signal
- `.eth_start(eth_start)` // this signal control the valid time of the whole module
- `.data(rx_data)` // output data
- `.crc_en(crc_en)` // enable CRC checksum
- `.DATA_LENGTH(DATA_LENGTH)` // specify the data length
- `.to_mac(to_mac)` // this specify the destination MAC address
- `.send_data_en(send_data_en)` // this sinal enables sending data for CRC
According to the theories discussed in the former chapter, according to the user defined Ethernet port packet format, the MAC layer, IP layer and UDP layer are established for data transmission. Furthermore, CRC checksum is separated for a solo module for check data error.

### 5.6.3. CRC checksum module

It is obvious that this module is for checking data error by using CRC checksum algorithm. This is realized by directly write the HDL codes. The checking algorithm has been discussed in the former paragraph. The reference codes can be founded in the appendix.

CRC module key components:
- .\clk(clk125m), // input signal
- .\rst_n(locked), // reset signal
- .\send_data_en(send_data_en), // this signal comes from data_gen module for receiving data signal
- .\rx_data(rx_data), // input data
- .\crc_en(crc_en), // this signal enables CRC check option
- .\tx_data(tx_data), // output data
- .\dv_rgmii(dv_rgmii) // this signal controls the Ethernet PHY chip

### 5.6.4. Data-transmission module

After getting through all the three layers and passed the CRC checksum, the data packet is ready for sending out. The final sending module, which is directly connected to the Ethernet PHY chip, is composed of three parts.

- Sending signal control
- Data sending
- Double-speed sending realization

In order to improve the sending speed, the double-speed sending module is created. Commonly, data will be only sent at the positive edge of the clock, but in this double-speed sending module, data will be sent at both positive and negative edges of the clock.

At the positive edge, 4 bits of higher order will be sent. At the negative edge, the lower order 4 bits will be sent.
All the three parts are realized through mega functions.

- **myoddr: ctl module**
  This module is for realizing double-speed data transmission. Through this module, at the positive edge of
  the clock pulse, 4 high-order bits will be sent. At the negative edge of the clock pulse, 4 low-order bits
  will be sent.

- **myoddr: oc module**
  This module is for generating clock pulse. Data sending is based on this clock signal.

- **oddr: oddr module**
  This module is for operating sending data to the external Ethernet PHY chip.

Key components of the module:

- `.clk(clk125m), // input signal clock`
- `.din(tx_data), // input data`
- `.dv(rx_dv_1), // Ethernet PHY chip control signal`
- `.dout(ETH_TXD), // output data`
- `.dctl(ETH_TXCTL), // signal control for data sending`
- `.oclk(ETH_TXCK) // data send based signal clock`

### 5.7. Quartus II implementation

As is discussed in the chapter 4, all the former design is not changed but the data transmission module is
added at the end of the project. It grasped data from the VGA side, reconstruct the data format and turn to
data packages to go through UDP, IP and MAC layers and finally be sent to the Ethernet PHY chip.

After inserting this module at the end of the project, the project needs to be re-compiled to check if there
is any error existed or not, do re-routing, re-fitting and check the timing again. When the compilation
process is done, re-download the .sof file to the FPGA board through the USB blaster again.
5.8. PC side data receive

5.8.1. Environment and Software Preparation

In order to run the program successfully so that the PC side can receive the data which are sent from the FPGA board, the transmission environment should be pre-defined.

For FPGA board side, before final compilation, the MAC address of the destination PC should be specified and the IP address of the PC should be manually set. On the PC side, the “wireshark” software is used for receiving the data and saving.

Steps for testing:
1) Modify the .v code to specify the destination MAC address
   Destination MAC address: h089e01d0529a
2) Modify the PC’s IP address
   IP address of PC: 192.168.1.118

3) Re-compile the whole project and download the .sof file to the FPGA board
4) Connect the board with PC with network cable though the board’s Ethernet 1 port.
5) Switch the SW1 to upside. This switch is for controlling open-close the data transmission function on board.
6) Open the “wireshark” software and choose “local connection” option to receive data.
7) Save the data at local PC.

5.8.2. Data transmission results

Theoretically, according to the former data transmission design, the current system can hold high throughput data amount. It can output data at the rate of 60 frames per second and be received by the PC side. The data transmission rate is at about 15.58 MB/s. $(640*480*8*60/(8*1024*1024))=15.58MB/s$
Figure 70 Data received by the PC side through wireshark software
Chapter 6

6. Historical methods used to achieve data transmission

One difficulty in this project design is the data transmission part. How to efficiently read the data out from the FPGA board and be stored from the external machine like PC is a big problem. In this chapter, one method is listed to illustrate how we have tried to achieve the data transmission function at first and why it is not finally chosen for the project.

6.1. NIOS II processor with NicheStack TCP/IP stack

6.1.1. Design idea

The Nios II processor core is a soft-ware central processing unit that the user could program onto an Altera FPGA board [33]. The main design idea for creating such a processor is separated into two parts: hardware construction part and software function part. For the hardware part, the Quartus II offers a “Qsys” tool for the user to construct the hardware architecture of the design. In this case, an architecture of Ethernet data transmission processor is constructed. With the help of “Qsys” tool, different functional modules are already existed and the user can customize the structure of the system [34]. In this design case, several key components like clock, cpu, system_id peripheral, tir-stae conduit pin sharer, SDRAM, parallel I/O, controllers are included. The usage of these key components are much more like the megafunctons mentioned in the former chapters. User can call the component from the “component” library and configure the component according to the function of the design. Then, the user should allocate proper addresses for different components and specify the connection relationship of the components. From the below picture, the whole map of the hardware architecture can be seen clearly.
This “Qsys” tool helps the user to construct the hardware part of the Nios II processor more conveniently. For software side, another software called “eclipse HELIOS” is used to achieve the software function. In order to transmit the data through the Ethernet port, one protocol we choose to use is the NicehStack TCP/IP stack protocol [34]. By using the C programming, one simple socket server system can be established. The FPGA board and the PC can establish communication relationship through client socket and server socket. One microprocessor of Altera’s webserver project is migrated to the design. Details can be seen as below.
For this design, it can read out data from SDRAM of the FPGA and then transmit the specified data to the PC side successfully. The received data can be shown within the console window of the “eclipse” software.
6.1.2. Problems of the design

Generally speaking, this method of design can meet with the basic needs of data transmission and it can tolerate high throughput data demand. One key problem of this design is that it is very difficult to integrate the Ethernet data transmission part with the image detection part together in one FPGA board. In other word, the image processing and data transmission parts cannot be performed simultaneously. Because both the data transmission and image processing part uses SDRAM resources. The conflict of resource using is unavoidable. For image processing part, the SDRAM is used to buffer, reconstruct and process millions of pixel data in every cycle. For data transmission part, the SDRAM is used to run the whole NIOS processor. All these two parts will consume large resource of SDRAM and SDRAM cannot be both read and written at the same time. This confliction of hardware makes this method unacceptable for the design.
Chapter 7

7. Conclusion

7.1. Conclusion of the project

For this project design, it achieve the function of using hardware development board to do real-time data collection, feature detection and Ethernet data transmission. Main difficulties of the design are listed below:

- Integrate image processing and Ethernet data transmission in one project
- Construct a complex hardware system to build an all-in-one solution
- Long-time debugging for testing the correctness of the design

Highlights and key contributions of the design:

- Using a new method to develop the hardware project. By using CWB, the designer can write the algorithm in ANSI-C and avoid directly using hardware description language to design the system, which will decrease the designing period and easier for the designer to modify the system.
- Achieving real-time feature detection and Ethernet data transmission in one design. This largely increase the efficiency of the system and offers a new solution to IoT project.
- The system can tolerate high data throughput at 60 frames/second.

The result of the design is reasonable and acceptable. The system can capture image correctly, locate the number-plate area successfully and transmit the data out of the FPGA board to the external PC efficiently. Since the data throughput in this case is very huge, the system can also tolerate the throughput at 60 frames/second.

7.2. Further improvement

7.2.1. Detection algorithm

For the current stage, the number-plate detection algorithm is not robust enough to adapt to different situations. The algorithm can be improved to be more robust that if the number-plate is warped, partly covered or blurred, it can still locate the number-plate position correctly. On the other hand, if the algorithm can be improved to identify the content of the number-plate, it will be greater.

7.2.2. Data storage format

Currently, the PC side can receive the data through the help of “wireshark” software and all the pixel value can be saved in txt format. According to the transmission algorithm, each line of the picture is
transmitted as one frame of data. Thus, if one method can be thought to save the data at PC side more efficiently, it will help the future analyze of these data much easier.

Bibliography

[16] Terasic, “Terasic TRDB-D5M Hardware specification”
[19] Fernando Ubiña, “FPGA implementation of camera colour model conversion”
[23] RFC 768
[25] RFC 791
[26] RFC 2474
[27] RFC 3168
[28] RFC 6864
[34] Altera, “Using the NicheStack TCP/IP Stack - Nios II Edition”
Appendix

1. Detailed information of the DE2_115 board

1.1.1. DE2_115 FPGA board tool package

![DE2_115 FPGA board tool package](image)

1.1.2. The composition and modules of DE2_115 board
Figure XXX gives out the overall appearance of the DE2_115 board and describes the composition of the board, as well as makes the labels of the connectors and the key positions.
The DE2_115 board can support various design requirements according to the designer’s needs, including from simple design to complicated multimedia circuit design.
The DE2_115 board concludes the following hardware configuration:
- Altera Cyclone IV 4CE115 FPGA chip
- Altera series configuration chip – EPCS64
- On board USB Blaster download circuit which supports both JTAG modes and AS modes
- 2MB SRAM
- 2 chips for 64 MB SDRAM
- 8MB flash memory
- SD card slot
- 4 buttons
- 18 switchers
- 18 red LEDs
- 9 green LEDs
- 50MHz crystal oscillator for clock signal
- 24-bit CD quality CODEC chip which has input, output and microphone input port
- VGA DAC (8-bit high-speed three channel DACs) which has VGA output port
- TV decoder (NTSC/PAL/SECAM) and TV input port
- 2000 MegaHz Ethernet PHY with RJ45 connector
- USB master-slaver controller which has A type and type ports
- RS-232 transceiver and 9 pin connector
- PS/2 mouse/keyboard port
- IR transceiver
- 2 SMA ports for external clock’s input/output
- 1 GPIO for 40 pins with diode protection circuit
- 1 HSMC connector
- 16*2 LCCD modules

Other than these hardware functions, DE2_115 develop board also supports standard IO interfaces and one control panel for evaluating different modules. The control panel also provides a plenty of demo examples for verifying the advanced functions of DE2_115.

In order to guarantee the normal use of the DE2_115 board, the user must first be familiar with Quartus II software.

1.1.3. The system diagram of the DE2_115 board

Figure 5 shows the system diagram of the DE2_115 board. In order to provide the most convenience for the users, all the connectors are connected to Cyclone IV E FPGA devices. Thus, the user can complete any design through the debugging of FPGA.
The detailed information of the modules:

FPGA devices
- Cyclone IV EP4CE115F29 device
- 114,480 logic devices
- 432 M9K internal memory module
- 3,888 Kbits embedded storage
- 4 PLL

FPGA configuration
- Both support for JTAG mode and AS mode
- Provide configuration series – EPCS64
- Internal USB Blaster circuit

Storage configuration
- 128MB (32 Mx32bit) SDRA,
- 2MB(1Mx16) SRAM
- 8bit 8MB (4Mx16) Flash storage, configured at 8-bit operating mode
- 32Kb EEPROM

SD-card port
- Provide SPI mode and 4-bit SD mode for SD card insertion

Connector
- two 10/100/1000 ethernet port
- one HSMC
configurable we/O standard (volt: 3.3/2.5/1.8/1.5V)
- type A and type B USB ports
- Completely compatible for USB2.0 master-slave controller
- Support full speed and low speed data transmission
- Can be used as PC drive
  - 40 pin GIPO
  - VGA output port
  - VGA DAC (three channel high-speed video DACs)
  - RS-232 interface with stream control, which can support DB9 connector ports
  - Provide PS/2 mouse/keyboard connectors

Clock input
- Three 50MHz crystal oscillators
- One SMA external clock input

Audio CODEC
- 24-bit encoder/decoder
- Circuit input/circuit output and microphone input

Display output
- 16*2 LCD modules
- Switchers and 7-segment nixie tube
  - 18 switchers and 4 buttons
  - 18 red and 9 green LEDs
  - 8 7-segment nixie tube

Other characteristics
- Infrared remote receiver module
- TV decoder(NTSC/PAL/SECAM) and TV-in connector

Power
- DC power supply
- Switcher and voltage reduction adjustor LM3150MH

2. Source codes for number-plate detection algorithm

```c
process detect(){
    int red,green,blue;
    int maxhsv,minhsv,tmphsv,delta;
    int a,b;
```
int h, v, s;

// hsv transform
pixel[2] = input(input_row[2]);
pixel[1] = input(input_row[1]);
pixel[0] = input(input_row[0]);

red = pixel[2];
green = pixel[1];
blue = pixel[0];

a = input(OX);
// if(a==639)  
b = input(OY);  
if(b==0)  
{ OK=0;  
  maxmum=0;  
}  
tmphsv = min(red, green);  
minhsv = min(tmphsv, blue);  
tmphsv = max(red, green);  
maxhsv = max(tmphsv, blue);  
v = maxhsv;  
delta = maxhsv - minhsv;

if(delta==0)  
{  
  h=0;  
}  

s = delta * 100 / maxhsv;

if(red==maxhsv && delta!=0)
\[
\text{if (green==maxhsv&&delta!=0)}
\]
\[
\text{\{ h=120+(blue-red)*60/delta; \}}
\]
\[
\text{else if (blue==maxhsv&&delta!=0)}
\]
\[
\text{\{ h=240+(red-green)*60/delta; \}}
\]

if(h<0)
\[
\text{h+=360;\}
\]

// if (h>10&&s>10&&v>100)
if(h>20&&h<40&&s<55&&s>75&&v<3200)
\[
\text{\{ \text{count==0) \}}}
\]
\[
\text{\{st=a;\}}
\]
\[
\text{\{count++;\}}
\]
\[
\text{\{pixel[2]=255*16;\}}
\]
\[
\text{\{pixel[1]=255*16;\}}
\]
\[
\text{\{pixel[0]=255*16;\}}
\]

else
\[
\text{\{count=0;\}}
\]
\[
\text{\{pixel[2]=0;\}}
\]
\[
\text{\{pixel[1]=0;\}}
\]
\[
\text{\{pixel[0]=0;\}}
\]
if(a!=0&&count>maxmum)
    maxmum=count;

if(maxmum>threshold&&flag==0&&OK==0)
{
    ul=st;
    h1=b;
    cnt++;flag=1;
}
if(a==750&&flag==1)
    ur=ul+maxmum;
if(flag==1&&(count>threshold)&&(cnt>10))
{
    flag=0;
    height=cnt;
    cnt=0;
    maxmum=0;
    OK=1;
}
if(flag==1&a==799)
    cnt++;

if((flag==1||b==height+h1)&&(a==ul||a==ur))
{
    pixel[2]=255*16;
    pixel[1]=0;
    pixel[0]=0;
}
if((b==h1+2||b==height+h1+1)&&a>=ul&&a<=ur)
{
    pixel[2]=255*16;
    pixel[1]=0;
pixel[0]=0;
}
output_row[2]=pixel[2];
output_row[1]=pixel[1];
output_row[0]=pixel[0];

output(output_row[0]);
output(output_row[1]);
output(output_row[2]);

3. CRC checksum codes