THE SOFTWARE FOR THE AUTOMOTIVE VIDEO GENERATING HARDWARE PLATFORM

Final Report

Author: Xiaobo Qu
Student number: 00064763
School: HZ University of Applied Sciences
Supervisor teacher: Bert Verhage
Company: Innolux - TPO Displays Europe B.V.
In-company mentor: Bart Peters
Date of publication: 30-05-2013
Place of publication: Heerlen, The Netherlands
The software for the automotive video generating hardware platform

Final Report

Author: Xiaobo Qu
School: HZ University of Applied Sciences
Programme: Bachelor of Engineering in Mechatronics
Supervisor teacher: Bert Verhage
Company: Innolux - TPO Displays Europe B.V.
In-company mentor: Bart Peters
Date of publication: 30-05-2013
Place of publication: Heerlen, The Netherlands
Summary

The purpose of this project is to develop the software for the new stand-alone automotive video generating hardware platform which is under development in TPO Displays Europe B.V. at Heerlen, The Netherlands. This platform will be used in the reliability testing, EMC testing of the real product and also will be used for driving display modules in production.

There are two parts of the software development for this hardware platform. One of them is developing the embedded software implemented in C++ for an on the hardware platform. And the other is developing the GUI running on the PC, which is implemented in C#. And as an extra assignment the embedded software implemented in Verilog for the SPI Block of the FPGA on the hardware platform also need to be developed. Without these all of proper working software, there will be no working automotive video generating platform.

At the end of this project, the development of embedded software for the SPI Block was finished completely. And most of the functionalities were developed in the GUI and the embedded software in C++. But by changing the priority of the project, two functions in the GUI and one function in that embedded software in C++ are still under development.

Furthermore, there are some limitations in the online compiler was used for developing the embedded software in C++. So if the development cost is available, it would be better to introduce an IDE software application into this project.
Foreword

First of all, I want to appreciate my in-company mentor, Bart Peters, my colleagues Mark Janssen and Joep Schreurs for their tutor and help during this internship.

Inside this report you can find the introduction about the basic information, the main design assignment and the sub design assignments about this project. All the theoretical and technical parts of this project are dedicated in the chapter “Theoretical framework”. The “Method and materials” was written for introducing the method and the steps were taken and the materials were used for this project. And the details of design will be shown in the chapter “Design of sub systems” The chapter “Result” is for how the progress was made of the software, and it will be compared with the primary expectation in the chapter “Discussion”. And the chapter “Conclusion” reviews the results that have been achieved during the project. At the end, the advice could be take in the future is in the chapter “Recommendations” and the information were referred for this project was listed in the chapter “Reference”.
The software for the Automotive video generating hardware platform

### Abbreviations

- **ADC**: Analog-to-digital Converter
- **AMLCD**: Active-Matrix Liquid Crystal Display
- **AShell**: Apix Automotive Shell
- **CAN (Bus)**: Controller Area Network (Bus)
- **CLS**: Common Language Specification
- **CPHA**: Clock Phases
- **CPOL**: Clock Polarity
- **CS**: Chip Select
- **DAC**: Digital-to-analog Converter
- **FPGA**: Field-programmable Gate Array
- **GUI**: Graphical User Interface
- **HDL**: Hardware Description Language
- **IDE**: Integrated Development Environment
- **LC**: Liquid Crystal
- **LCD**: Liquid Crystal Display
- **MCU**: Microprogrammed Control Unit
- **MISO**: Master Input Slave Output
- **MOSI**: Master Output Slave Input
- **MSB**: Most Significant Bit
- **OLED**: Organic Light-emitting Diode
- **PLL**: Phase-Locked Loop / Phase Lock Loop
- **PWM**: Pulse-width Modulation
- **RAD**: Rapid Application Development
- **SCK / SCLK**: Serial Clock
- **SPI (Bus)**: Serial Peripheral Interface (Bus)
- **SS**: Slave Select
- **TFT**: Thin-film-transistor
- **TPO**: TPO Displays Europe B.V.
- **VHDL**: VHSIC Hardware Description Language
- **VHSIC**: Very-high-speed Integrated Circuits
# Table of Contents

Summary ........................................................................................................................... 0  
Foreword ........................................................................................................................... 0  
Abbreviations .................................................................................................................... 0  
1. Introduction ................................................................................................................... 1  
   1.1 Company ................................................................................................................. 1  
   1.2 Hardware platform ................................................................................................... 1  
   1.3 The main design assignment ................................................................................... 2  
   1.4 The sub design assignments ................................................................................... 2  
      1.4.1 The software of GUI .......................................................................................... 2  
      1.4.2 The software of MCU ........................................................................................ 2  
      1.4.3 The software of Serial Peripheral Interface (SPI) Block .................................... 2  
      1.4.4 All the connections ............................................................................................ 2  
2. Theoretical framework ................................................................................................... 3  
   2.1 The software of GUI ................................................................................................ 3  
      2.1.1 Microsoft Visual C# ........................................................................................... 3  
      2.1.2 C# ..................................................................................................................... 4  
   2.2 The software of MCU ............................................................................................... 5  
      2.2.1 mbed NXP LPC1768 ......................................................................................... 5  
      2.2.2 The mbed Compiler .......................................................................................... 6  
      2.2.3 C++ ................................................................................................................... 6  
   2.3 The software of SPI Block ....................................................................................... 7  
      2.3.1 Altera Cyclone® IV FPGA .................................................................................. 7  
      2.3.2 Verilog ............................................................................................................... 7  
   2.4 The others ............................................................................................................... 9  
      2.4.1 SPI protocol ...................................................................................................... 9  
      2.4.2 Apix2 controller ............................................................................................... 12  
      2.4.3 Liquid Crystal Displays (LCD's) ...................................................................... 15  
3. Method and materials .................................................................................................. 18  
   3.1 Method ................................................................................................................... 18  
      3.1.1 Introduction ...................................................................................................... 18
1. Introduction

1.1 Company

TPO Displays Europe B.V. (TPO) now is one part of Innolux Corporation. Innolux headquarters is in Jhunan Taiwan. They have development departments (in Taiwan, Japan and Heerlen, The Netherlands), fabrication facilities (in Taiwan and Mainland China) and sales departments (in Taiwan, Mainland China, Japan, Europe and the USA/Canada.)

Innolux produces Liquid Crystal Displays (LCDs), Organic Light-emitting Diode (OLED) Displays and Touch Screens in the range from LCD-TV to displays for the mobile market (Telecommunication and Automotive). In Heerlen they mainly develop displays for Nokia, Research in Motion, Acer, Audi, BMW, Porsche and Volkswagen.

The company started in 2010 from a merger of Chi Mei OptoElectronics, Innolux Display Corporation and TPO which itself was a joined venture between Toppoly and Philips Mobile Display Systems that started in 2006.

And now, Innolux is No.1 in the Small/Medium display market and No.3 in the mobile phone display market.

1.2 Hardware platform

Nowadays, in TPO, a new stand-alone automotive video generating hardware platform for driving automotive displays is under development.

This hardware platform is developed as a testing tool to deliver the sample for one driving automotive displays development project (See Figure 2). The staffs in TPO will use this platform in the reliability testing and EMC testing of the real product, and then they will improve the product according to the result of the testing. It also will be used for driving display modules in production.

The control function of this hardware platform requires implementation in embedded software implemented in C++ running on an ARM Cortex CPU on the platform. And on the hardware platform, there is a Micro-programmed Control Unit (MCU) named LPC1768 on the hardware platform. This MCU belongs to a series of ARM microcontroller development boards designed for rapid prototyping.\[1\]
And a hardware/software interface between CPU and the Altera Cyclone4 FPGA on the hardware platform requires definition and implementation. This Field-programmable Gate Array (FPGA) will take care of generating the test images and other display specific functions.

Next, a Graphical User Interface (GUI) running on a PC, implemented in C# requires the development to ease the use of the total setup. And without these all of proper working software, there will be no working automotive video generating platform.

1.3 The main design assignment

Developing and realizing the software for the entire system is the main question of this internship assignment. One of it is the software running on a MCU, and another one is the software of the GUI running on a PC.

1.4 The sub design assignments

According to the main question, there are several sub-questions in this project.

1.4.1 The software of GUI

The software of the GUI should be able to control the PC to send the commands to the MCU and receive the feedback from the MCU. So, the answers of how to setup the GUI in C# and achieve these functionalities should be found out.

1.4.2 The software of MCU

Then the software of the MCU in C++ should manage to let the MCU receive the commands from the PC and send the feedback to the PC. And it should act as link between all components from the hardware platform. So, the answers of how to program the embedded software in C++, implement them into the MCU and achieve these functionalities should be found out.

1.4.3 The software of Serial Peripheral Interface (SPI) Block

On the other hand, as an extra assignment, the hardware programming of the SPI Block in the FPGA also should be handled. This part should be able to receive the command from the MCU by SPI interface and store them in the specific register of the FPGA. So, the answers of how to program the SPI Block, implement the software into the SPI Block of the FPGA, and achieve these functionalities should be found out.

1.4.4 All the connections

As a consequence of this, the answer of how to build up the communication layer between all hardware components also has to be found out. Because most of them use their own interfacing and command structures. And the MCU is the overall heart that needs to supply each other component from the correct information and register settings. At the same time, the software of the GUI and the software of the MCU should make the PC and the MCU communicate with each other in reliable and structured way.
2. Theoretical framework

In this design assignment, there are a lot of professional and theoretical new knowledge and new concepts need to be known. In this chapter these knowledge and concepts will be introduced and explained, which are based on the sub design assignments which are introduced in the previous chapter.

![Figure 3](image.png) The architecture overview of the EMC/SA Testing Tool

2.1 The software of GUI

2.1.1 Microsoft Visual C#

In the Figure 4, it is the logo of Microsoft Visual C# which is the C# development tool developed by Microsoft. It includes an interactive development environment, visual designers for building Windows and Web applications, a compiler, and a debugger.[2] In the design assignment of the GUI part, Microsoft Visual C# was used as a developing tool for the embedded software of the GUI in C# running on the PC.

Microsoft Visual C# is part of an integrated development environment (IDE)[3] from Microsoft called Microsoft Visual Studio, which includes Microsoft Visual Basic, Microsoft Visual C++, and the JScript scripting language.

All of these languages provide access to the Microsoft .NET Framework, which includes a common execution engine and a rich class library. The .NET Framework defines a "Common Language Specification" (CLS), a kind of lingua franca that ensures seamless interoperability between CLS-compliant languages and class libraries. For the developers of C#, this means that even though C# is a new language, it has complete access to the same rich class libraries that are used by seasoned tools such as Visual Basic .NET and Visual C++ .NET. C# itself does not include a class library. [4]
2.1.2 C#

C# (pronounced C sharp) is the programming language which was used in this project for the programming the embedded software of the GUI running on the PC.

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, procedural, generic, object-oriented (class-based), and component-oriented programming disciplines. C# is one of the programming languages designed for the Common Language Infrastructure. [5]

It was developed by Microsoft Corporation[6], and it will immediately be familiar to the programmer who are used to program in C and C++. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++.[7]
2.2 The software of MCU

2.2.1 mbed NXP LPC1768

The mbed NXP LPC1768 Microcontroller is one of the major components on the stand-alone automotive video generating hardware platform.

This MCU which is based on the NXP LPC1768 in particular is designed for prototyping all sorts of devices, especially those including Ethernet, USB, and the flexibility of lots of peripheral interfaces and FLASH memory. It includes a built-in USB FLASH programmer, which makes the process become easier to the developer during the developing and programming the software of this MCU.

It is based on the NXP LPC1768, with a 32-bit ARM Cortex-M3 core running at 96MHz. It includes 512KB FLASH, 32KB RAM and lots of interfaces including built-in Ethernet, USB Host and Device, SPI, I²C, Controller Area Network (CAN), Analog-to-digital Converter (ADC), Digital-to-analog Converter (DAC), Pulse-width Modulation (PWM) and other I/O interfaces. The pinout above shows the commonly used interfaces and their locations. Note that all the numbered pins (p5-p30) can also be used as DigitalIn and DigitalOut interfaces.

This MCU provides enough free space for the embedded software developers. It provides a powerful and productive platform for building proof-of-concepts. For developers who are newer to a 32-bit micro-controller, mbed provides an accessible prototyping solution to get projects built with the support of libraries, resources shared in the mbed community. The developer can easily find lots of example software in this community and use them as the reference for their own developments and projects.
2.2.2 The mbed Compiler

The mbed compiler is developed by the mbed official website itself. In the design assignment of the MCU part, the mbed compiler was used as a developing tool for the embedded software in C++ running on the MCU.

The mbed compiler provide a lightweight online C/C++ IDE that is pre-configured to let users quickly write programs, compile and download them to run on the their mbed Microcontroller. The developer can use this compiler to develop software of their project, after they logged in the mbed website by using their own mbed account. As result of this there is a web application; the users don’t need to install any plug-ins to get it running. And the developers can save, share and commit their own code by the mbed compiler to the online terminal. Next time they login, they imidiately can continue with their work on their own project.

Furthermore, The Import Wizard of the mbed compiler allows developers to import programs and libraries published by other mbed users. This is useful for importing code that has been packaged as a reusable library component (e.g. a class for a peripheral), so the developers can quickly use and plug-in building blocks for their project.

2.2.3 C++

C++ (pronounced C plus plus) is the programming language which was used in this project for programming the embedded software of the MCU on this hardware platform.

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises both high-level and low-level language features. It was developed by Danish computer scientist Bjarne Stroustrup. The development of the C++ starts from 1978. That time, C++ was named C with Classes. In 1983, the language, C with Classes was renamed C++, as a pun involving the increment operator.

Now, C++ is one of the most popular programming languages over the world. And it’s implemented on a lot of different kinds of hardware platforms and operating systems. The area of the application of the C++ includes the software of systems, the software of applications, the driver of devices, the embedded software, entertainment software such as video games and so on. And nowadays, the engineer also will choose C++ in the hardware design as the initially described language.
2.3 The software of SPI Block

2.3.1 Altera Cyclone® IV FPGA

The Altera's Cyclone® IV FPGA also is one of the major components on the stand-alone automotive video generating hardware platform. Because some kind of reason which will be explained in a later chapter, the SPI Block in the FPGA was included into the design project as an extra assignment. So the introduction of the Altera's Cyclone® IV FPGA became quite important and necessary.

The FPGAs developed by Altera can be divided into several different levels. The Altera’s Stratix® (I, II, III, IV and V) FPGA which are specially for the high-end market, Altera’s Arria® (I, II and V) FPGA’s are developed for the midrange market, and the Altera’s Cyclone® (I, II, III, IV and V) devices designed for the low-cost market. The FPGA on the automotive video generating hardware platform is the Cyclone IV EP4CE6E22C8N, belongs to the “Cyclone IV E” devices.

FPGA’s are programmable digital logic chips. It is a semiconductor device that can be configured and programmed after finished manufacturing. It means that developers can reach almost any digital function by using programmable FPGA’s.

The developers can use a computer to describe a "logic function" that they want. They can draw a schematic, or just create a text file describing the function. Then the developers can use their own computer to program and compile these logic functions. And next they can download the bin file into their own own FPGA, which is created by the software likes the Altera Quartus II 12.1.

Furthermore, the developers can download that bin file as many times as they want, there is no limitation. If they want, they can add, adapt different functionality to the program or make changes each time in the program. If there are some mistakes made in their design, just fix them, recompile and download them again to the FPGA. There is no need to hardware changes or fixes on the PCB, no soldering or component replacement should be done. It adds great flexibility in making hardware designs, logical structures and circuits on a pcb.

2.3.2 Verilog

Verilog is a hardware description language (HDL) used to model and describe the electric systems. In the development of the SPI Block in the FPGA, Verilog is the language which was chosen to program the software of this part.

---

1 The Quartus II software offers a complete solution for system designers who design with Altera FPGA and CPLD devices. After the developers compile their design, they can use the Quartus II Programmer to program or configure your device, to test its functionality on a circuit board. – Altera Corporation, Quartus II Handbook Version 13.0, Volume 3: Verification, Chapter 18, November 2012
Compared to another HDL, named VHSIC\textsuperscript{2} Hardware Description Language (VHDL), Verilog is more similar with the C language. So it will be easier for the programmer who’s familiar with C to grasp it. And that is the reason why it was chosen as the programming language in this design assignment. Verilog has a basic preprocessor like the C language, and its control flow, keywords are equivalent, which including if/else statement, for loop, while loop, case statement and so on. And its operator precedence is compatible. On the other side, the differences on the syntactic Verilog and C language include the way of variable declaration, using begin and end instead of curly braces ("{" and "}") to demarcate the procedural blocks and other minor differences.

Although Verilog is a little similar with the C language, as HDL, Verilog is still quite different from a software programming language. Because it includes the ways to describe the hardware components including the time and signal dependencies (sensitivity). For example, there are two different assignment operators in Verilog. They are the blocking assignment (symbolize by "="), and the non-blocking assignment (symbolize by "\leq"). The difference between them is the non-blocking assignment allows the developers to describe a state-machine update without needing to declare and use temporary storage variables.

Verilog can be used to designing at several different levels of abstraction. And in those several levers, there are three of them are quite useful, they are behavioral level, register-transfer level and gate level. The behavioral level is which level describes a system by concurrent algorithms. The register-transfer level is the level used for specifying a circuit by operations and the data transfer between the registers. And within the gate level the characteristics of a system are described by logical links and their own timing properties.\textsuperscript{[18]}

\textsuperscript{2} VHSIC was a 1980s U.S. government program to develop Very-high-speed Integrated Circuits. A well-known part of the project's contribution is VHDL, a hardware description language. – Wikipedia, VHSIC(retrieved on: 26 May 2013), http://en.wikipedia.org/wiki/VHSIC
2.4 The others

2.4.1 SPI protocol

For connecting the MCU to the FPGA and Apix2 controller, the SPI protocol will be used in this design assignment part. The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link de facto standard, named by Motorola, which operates in full duplex mode.\[^{16}\] In the communication of the SPI protocol, the devices will be divided into the master and slave mode, and the master device will be in charge of initializing the data frame, and providing the clock.

The SPI protocol communication between two devices requires at least two wires, but in this architecture four wires to setup the communication, they are Serial Clock (SCK / SCLK), Master Output Slave Input (MOSI), Master Input Slave Output (MISO) and Chip Select (CS) or Slave Select (SS). And in the multiple slave devices SPI communication, the SPI protocol allows every single slave devices to use their own individual CS lines to communicate with the master device.

**Single slave**

In the Figure 9, there is a simple example about the SPI communication between two chips. As what was mentioned in the paragraph above, the SPI communication between these two chips requires four wires to be used. As you can see, in this figure, the wires are called SCLK, MOSI, MISO and SS, and one of the chips is called the SPI master, while the other one is called the SPI slave.

![Figure 9 SPI bus: single master and single slave](image)

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency supported by the slave device. Such frequencies are commonly in the range of 1–100 MHz.\[^{17}\] After the setting of the format of the SPI protocol (bit order, length of data words exchange, etc...) had been finished, the transmission will start. The clock will be generated by the master, and one bit of data will be transmitted on each clock positive or negative edge. In the transmission by the SPI protocol, there are only wires named MOSI and MISO used for data transfer, every one of them for each single direction.

In a normal SPI transmission, the master will transfer the logical low level for the chip select over the SS line, because for this chip the chip select line is active low.\[^{18}\] Sometimes, for example, in an analog-to-digital conversion, after the slave chip has been activated, the master needs to wait for a certain period. This wait is required before starting to generate the clock pulse, for conversions or setting data ready. Then during every single clock cycle, the master chip sends a bit data over the MOSI,
the slave will read it at the same time. And on the other hand, if the slave is also sending a bit data over the MISO, the master chip will read it at the same time.

![Image](image.png)

**Figure 10** A typical hardware setup using two shift-registers to form an inter-chip circular buffer

**Multiple slave**

As what was introduced in the above paragraph, one SPI master device can communicate with multiple SPI slave devices. It can be setup by adding the SS wires and connecting most signals in parallel, or chaining the slaves. In the development of the software in this project, the method used is the previous one. It’s called the daisy chain SPI configuration, and the second method called independent slave SPI configuration.

In the independent slave configuration (See **Figure 11**), there is only one independent SS line for each single slave device. This is also a way of multiple SPI slave devices configuration that usually is used. As result that all the MISO pins of the slave devices will be connected together in this kind of configuration method, these MISO pins are all required to be the tri-state\(^3\) pins.

And in the daisy chain SPI configuration (See **Figure 12**), the multiple slave devices share the same SCLK line, MOSI line and MISO line. It means the input pin of the first slave chip, which connects to the output pin of the mater chip, also connects to the second slave chip, third slave chip, etc. And the transmissions among these several slave

\(^3\) In digital electronics three-state, tri-state, or 3-state logic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit. This allows multiple circuits to share the same output line or lines (such as a bus which cannot listen to more than one device at a time). – Wikipedia, Three state logic(retrieved on: 27 May 2013), http://en.wikipedia.org/wiki/Three-state_logic
The software for the Automotive video generating hardware platform

chips are controlled by the different SS pins on the master devices. So this configuration requires some separate SS (chip select) line for each slave.

**Mode numbers**

In the SPI protocol format part of the SPI transmission, In addition to the clock frequency setting, it has settings of the transmission mode (listed in the Table 1). These should be set before the transmission starts. It is referred to the combinations of the polarity and the clock phases, which adds more flexibility to the SPI communication channel among all the master and slave devices.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Clock Polarity (CPOL)</th>
<th>Clock Phases (CPHA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1

For more easier to introduce and explain the transmission modes, the timing diagram is shown in the Figure 13.

**Figure 13** A timing diagram showing clock polarity and phase. The red vertical line represents CPHA=0 and the blue vertical line represents CPHA=1.

CPHA = 0 means the base value of the clock is 0, and the format of the transmission is sampling on the first clock edge, while CPHA = 1 means that the base value of the clock is 1, and the format of the transmission is sampling on the second clock edge. Whatever if the first clock edge of the clock cycle is positive edge or negative edge.

And at CPHA = 0, for CPOL = 0, data is sampled on the positive clock edge and is propagated on a negative clock edge. For CPOL = 1, data is sampled on the negative clock edge and is propagated on a positive clock edge. At CPHA = 1, for CPOL = 0, data is sampled on the negative clock edge and is propagated on a
positive clock edge. For CPOL = 1, data is sampled on the positive clock edge and is propagated on a negative clock edge.

Furthermore, the data must keep stable for a half cycle before the first clock cycle. And for all the SPI transmission modes, the initial clock value must be stable before the SS / CS line is selected.

2.4.2 Apix2 controller

The INAP375T / INAP375R Transmitter and Receiver, or also known as Apix2 controller is a very important component on the stand-alone automotive video generating hardware platform.

In this project, the software of the MCU should be able to communicate with the Apix2. At beginning the protocol planned to be used is the I2C, but later on, it was changed to the SPI protocol. The communication between the MCU and the Apix2 controller is a little bit more complicated compared to the one between the MCU and the SPI Block in the FPGA. So that is the reason why it should be introduced in this chapter.

APIX2 is Inova Semiconductors latest 3 Gbit/s Pixel Link, solving point to point connectivity solutions in the vehicle for DISPLAY and CAMERA applications.[19]

The configuration from a microcontroller can be handled though the SPI Slave interface in the Apix2 controller. For configuration and access from a microcontroller, there are two methods that can be used, one is the “Single Byte Read/Write” method, and the other is the “Burst Read/Write” method.

In the transmission and receiving function of the Apix2 controller, the address and data bits are written and read in Most Significant Bit (MSB) first order. And the difference between READ and WRITE access is determined by the MSB of the
address. For READ access, the MSB of the address needs to be set LOW, for WRITE access the bit needs to be set HIGH.

The Apix2 controller supports “Single Byte Read/Write” method as well as the “Burst Read/Write” method, in which the address will be automatically increased for each byte. The length of the transmission is controlled by the chip select pin named SPI_S_CS2#. The different between these two methods are shown in the Figure 17 and Figure 18 below.
On the other hand, due to the different flexible multiplexing of the data interface, the INAP375T provides several access paths to the APIX Automotive Shell (AShell)\(^4\), to overcome pin limitations.\(^{[23]}\) Sending data over the SPI protocol is always handled through the SPI Slave interface in the Apix2 controller. There are two ways that can be used to transmit the data to the AShell (please refer to Figure 19). One of them is using the “SPI_S_CS0” pin to let the data received by “SPI_S_SDI” directly forward to the AShell. And another one is using the SPI_S_CS2 which is used access to the device registers. Then the address of this writing option should be the address of the as0_data register or the as1_data register.

![Figure 19 SPI write access to the AShell\(^{[24]}\)](image)

Before initiating the transmission, the signal on SPI_S_STALL pin must indicate and being checked if low, this means the data path is not busy. If not and the transmission is started at this moment, the data will be lost.

And the Figure 20 below shows more details and requirements about the transmitting data to AShell by using the SPI_S_CS2 pin.

\(^4\) The AShell is an abstraction layer for the data communication. The AShell allows a secure and error free data exchange on the bi-directional full duplex communication channels of the APIX link. -- Inova Semiconductors, INAP375T Advance Information, Revision 1.1, page 4
2.4.3 Liquid Crystal Displays (LCD’s)

The kind of the automotive displays which for the new stand-alone automotive video generating hardware platform is developed for, are LCD’s. Although there is not a lot of professional knowledge needed for the software development, some brief introduction about the LCD’s still will be provided in this chapter, to get knowledge about the technology and working principles of these type of displays.
LCD’s consist of two glass plates (please refer to Figure 21), and the transparent conductive electrodes covered on the each plate. The distance between the substrates (cell gap) is in general 4.5 or 6.0 micron. And the cell is filled with the liquid crystal (LC) material. The LC material was discovered by the Austrian botanist and chemist, Friedrich Richard Reinitzer in late 1880s.

The term called LC material is the sort of material with a liquid crystalline phase. This phase is the phase between the solid phase and the liquid phase, as an additional phase exist in some materials. The LC material consists of long cigar-shaped organic molecules. The elastic coupling between these molecules is strong and consequently the orientation of the molecules is in identical directions.

On the two sides of the LC layer, there are two polarizing filters, one of these filters polarizing the light vertically and the other one polarizing the light on the horizontal direction. Actually, the LC layer do not generate light, it only transmit or reflect the external light through changing the direction of that long cigar-shaped organic molecules of the LC material. So there is always a fluorescent or LED back light behind the LC layer.

In this project, the new automotive displays are not only LCD’s, but are also Active-Matrix Liquid Crystal Displays (AMLCD’s).

As the Figure 22, the AMLCD’s with x*y pixels consists of x horizontal rows and y vertical columns (in the figure above x = 3 and y =4). In series with each pixel a non-linear switching elements is used. This non-linear element can be a diode, a transistor or a switch. For example, the Thin-film-transistor (TFT) is widely used into the AMLCD’s.

The non-linear elements are on one side connected to vertical columns and on the other side to the transparent electrode of the LCD cell, where is filled with the LC materials. The closure of the switches is accomplished by the selection voltage on the multiplexed rows.

During one of these x rows is selected, all switches belonging to the selected row will become conductive. And the capacitors will be able to be charged by the column
electrodes. The difference between the y column voltages and the common voltage at the counter electrode will determine the voltage across these capacitors on y columns. Then the switches will be opened again after the line selection time. And the capacitor of each pixel has a memory function for the video signal during the frame time.

In the same order all rows will be successively selected and the pixel capacitors of the entire matrix display will be charged. In this respect the AMLCD is a memory display that modulates the ambient light during the entire frame time. [29]
3. Method and materials

The design method is a necessary part for any design assignment. It can help the engineer to handle a design with high-efficiency and a good performance. On the other hand, in every design assignment, a lot of materials will be used for supporting the engineer to finish the researching, the measurement and the designing.

So, in this chapter, the method and the materials that were used for this internship assignment will be introduced.

3.1 Method

3.1.1 Introduction

The design method used for this project called Engineering Design Process. This methodology is which one TPO tends to use. These are the reasons that this methodology was chosen to create the design of the internship assignment.

Engineering Design Process is the formulation of a plan to help an engineer to build a product with a specified performance goal. This process involves a number of steps, and parts of the process may need to be repeated many times before production of a final product can begin. In the Figure 23 below, there is the overview about all the steps of this method.

This design method was followed during the whole development of the project, and also during each step in the development of every small assignment.

![Figure 23 The steps of the Engineering Design Process](image-url)
3.1.2 Define the problem

The engineering design process starts from **defining the problem**, when the designer asks question about the problems that he or she observe. Any kind of questions are based on three main questions. They are what is the problem or need, who has the problem or need and why is it important to solve.

3.1.3 Do background research

It is the consensus that learning from experience of others rather than blundering around and repeating their mistakes. So, for an engineering design project, it is necessary to **do background research** after the designer finished the problem definition phase. And research should be set in two major areas; they are users or customers and existing solution.

3.1.4 Specify requirements

After doing the background research phase, it is the **specify requirements** phase. One of the best ways to identify the design requirements for the solution is to analyze the concrete examples of a similar, existing product, noting each of its key features.

3.1.5 Create the alternative solution and choose the best one

After finished all the steps above, the designer should start to **create the alternative solutions**. There are always many good possibilities for solving design problems. If the designer focus on just one before looking at the alternatives, it is almost certain that you are overlooking a better solution. Good designers will try to generate as many possible solutions as they can. Then looking at whether each possible solution meets his or her design requirements. Some solutions probably meet more requirements than others. Reject solutions that do not meet the requirements. And the designer will **choose the best one** from all the alternative solutions.

3.1.6 Develop the solution, build the prototype, test and redesign

After the designer has chosen the best solution, he or she will start to **develop this solution**. The development involves the refinement and improvement of a solution, and it continues throughout the design process, often even after a product ships to customers.

After or during the process of the development of the chosen solution, one or several **prototypes will be built**. The prototype is an operating version of a solution. Often it is made with different materials than the final version, and generally it is not as polished. Prototypes are a key step in the development of a final solution, allowing the designer to test how the solution will work.

As we know, the **testing and redesign phase** also is necessary. The design process involves multiple loops and circles around the final solution. The designer will likely test his or her solution -- find problems and make changes -- test the new solution -- find new problems and make changes -- and so on, before settling on a final design.
3.1.6 Communicate the result

At the end of the engineering design, to complete one project, communicate the results to others in a final report, or a display board will be necessary. Professional engineers always do the same, thoroughly documenting their solutions so that they can be manufactured and supported.
3.2 Materials

3.2.1 Computer

The computer was used for programming the embedded software for the GUI in C#, the embedded software for the MCU in C++ and the embedded software for the SPI Block in the FPGA using Verilog.

At the same time, it also was used for searching the information on the internet and monitoring the signal on the USB port, which was sent out by the GUI or feedback from the MCU.

3.2.2 Power supply

The power supply was used for providing power to the automotive video generating hardware platform and the automotive displays.

The displays include BMW C1D65 F25LCI L6MU, BMW FOMO 10.2 Apix1 Mode and BMW FOMO 10.2 Apix2 Mode.

3.2.3 Tektronix TDS 3012 Oscilloscope

The Oscilloscope was used is the TDS 3012 100 MHz 2 Channel Digital Phosphor Oscilloscope from Tektronix, for measuring and monitoring the SPI protocol between the MCU and FPGA or between the FPGA and Apix2 in the debugging. Sometimes, it also would be used on measuring the SPI protocol between FPGA and Apix2 in the debugging.

3.2.4 Hardware platform

The hardware platform is the new stand-alone automotive video generating hardware platform which is under development in TPO, which was used for testing the software programmed for the GUI and the MCU from 4th March 2013 till the end of this internship project.

Before 4th March 2013, the hardware platform was not received yet. The software programmed just was tested with the MCU development board alone.
3.2.5 BMW Automotive LCD Display (C1D65 F25LCI L6MU)

This display is one of the last generation developments for automotive displays for BMW, made by Innolux. As the result of this the display chosen for this software development is an already available mass production product, to replace the BMW FOMO LCD Display (FOMO 10.2). For a starting period, the software was under development using this replacement (Apix1) display, till the FOMO 10.2 display was received and available by TPO.

Most of the time, it was connected with the hardware platform and used for debugging the software running on the MCU, and also for the SPI Block development in the FPGA.

3.2.6 BMW Automotive LCD Display (FOMO 10.2)

This display is the new generation of the automotive displays of BMW, made by Innolux. It was connected with the hardware platform and used for debugging the software running in the MCU, and also the software for the SPI Block in the FPGA.

3.2.7 ALTERA USB Blaster

The Altera USB Blaster was used for linking the PC and the FPGA, when the embedded software in Verilog, for the SPI Block in the FPGA, was being programmed by Altera Quartus II 12.1.

It also was used when the signal that passed the SPI Block was being monitored by the In-System Sources and Probes Editor\(^5\) of the Quartus programmer during the design debugging.

3.2.8 Other cables

These cables are including one LVDS cable and two USB cables.

\(^5\) The In-System Sources and Probes Editor in the Quartus II software extends the portfolio of verification tools, and allows you to easily control any internal signal and provides you with a completely dynamic debugging environment. Coupled with either the SignalTap II Logic Analyzer or SignalProbe, the In-System Sources and Probes Editor gives you a powerful debugging environment in which to generate stimuli and solicit responses from your logic design. – Altera Corporation, Quartus II Handbook Version 13.0, Volume 3: Verification, Chapter 16, June 2012
The LVDS cable was used for linking the hardware platform and the displays. One of the USB cables is for setting the USBSerial Port between the PC and the MCU, and the another USB cable is for connecting the PC and the MCU for downloading the bin file for the MCU.

Figure 31
4. Design of sub systems

In the Figure 32, it is the architecture overview of the primary solution. This solution would be improved on three parts during the test and redesign phase, which are the SPI Block of the FPGA, connection between the MCU and Apix2 controller and the development priority of some functionality in this project.

![Figure 32](image)

As the primary solution was planned, the connection among the PC, MCU and message LCD (for debugging) would be setup at the beginning. The connection between the MCU and message LCD would be setup according to the specification of that LCD (See Appendix B). And the connection between the PC and MCU would be selected as the USBSerial\(^6\) protocol.

Then the GUI, which is called Kleine_Friet, running on the PC in a draft version would be developed, it would be able to send commands of the video setting and picture generator to the MCU and receive the feedback from the MCU, then the Phase-Locked Loop (PLL)\(^7\) Calculation Function should implemented into the GUI. At the same time, the software running on the MCU for receiving the commands from the GUI, storing them into the registers and sending the feedback to GUI also would be developed.

After the above task was finished, the connection between the MCU and the FPGA would be enabled, the interface between them would be selected as the SPI protocol. Next, the software of the MCU should be improved for being able to send out the command in the specific register to the SPI Block in the FPGA.

---

\(^6\) The USBSerial interface is used to emulate a serial port over USB. The user can use this serial port as an extra serial port or as a debug solution. It's also a great solution to easily communicate between the user's mbed and a computer. – mbed.org, Handbook, USBSerial, Last modified 1\(^st\) Mar 2013, by Samuel Mokrani.

\(^7\) A phase-locked loop is a control system that generates an output signal whose phase is related to the phase of an input "reference" signal. -- Wikipedia, Phase-locked loop (retrieved on: 27 May 2013), http://en.wikipedia.org/wiki/Phase-locked_loop
On the other hands, the connection between the MCU and the Apix Controller also would be enabled. As planned, the I²C protocol would be choose to setup the communication between these two components. Then the software of the MCU should be improved again for being able to send out display initializing code for testing all the connections were built up before, with the last generation BMW automotive display in Apix1 mode.

After all the above parts were finished perfectly, the display would be updated to the new generation BMW automotive display (FOMO 10.2) in Apix1. Next, the code in the software of the MCU would be improved for matching the display configuration requirements of the FOMO displays, then test with that display. Next, the FOMO display would be refreshed to the Apix2 mode. And the code in the software of the MCU would be updated according the configuration requirements of the Apix2 mode and test with this display in the new mode.

After got success in the Apix2 mode testing, the software of the GUI and the MCU all would be improved again to enable on reading back the data from the Apix2 controller on the hardware platform and on the displays. Then the main question of this internship assignment would be completely answered.
4.1 The software of the GUI

As the main requirement is the basic functionality of the embedded software was given by the company. So what is quite important is the maintainability of the code in the further improvement or developments. The Table 2 below is parts of basic parameters the users of the testing tool need for controlling in the GUI. They are also the quite important specify requirements for the software development.

<table>
<thead>
<tr>
<th>Structure</th>
<th>1byte</th>
<th>1byte</th>
<th>2byte</th>
<th>3byte</th>
<th>4byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>data1</td>
<td>data2</td>
<td>data3</td>
<td>data4</td>
<td></td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Pattern Generator</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL control</td>
<td>20hex</td>
</tr>
<tr>
<td>Base frequency</td>
<td>1byte</td>
</tr>
<tr>
<td>PLL1</td>
<td>1byte</td>
</tr>
<tr>
<td>PLL2</td>
<td>1byte</td>
</tr>
<tr>
<td>PLL3</td>
<td>1byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Horizontal timing</th>
<th>21hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_PW</td>
<td>1.5byte</td>
</tr>
<tr>
<td>H_FP</td>
<td>1.5byte</td>
</tr>
<tr>
<td>H_DV</td>
<td>1.5byte</td>
</tr>
<tr>
<td>H_BP</td>
<td>1.5byte</td>
</tr>
<tr>
<td>H_FC</td>
<td>1.5byte</td>
</tr>
<tr>
<td>reserved</td>
<td>0.5byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertical timing</th>
<th>22 hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_PW</td>
<td>1.5byte</td>
</tr>
<tr>
<td>V_FP</td>
<td>1.5byte</td>
</tr>
<tr>
<td>V_DV</td>
<td>1.5byte</td>
</tr>
<tr>
<td>V_BP</td>
<td>1.5byte</td>
</tr>
<tr>
<td>V_FC</td>
<td>1.5byte</td>
</tr>
<tr>
<td>reserved</td>
<td>0.5byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern Select</th>
<th>23 hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern_img</td>
<td>1byte</td>
</tr>
<tr>
<td>Pattern_Red</td>
<td>1byte</td>
</tr>
<tr>
<td>Pattern_Blue</td>
<td>1byte</td>
</tr>
<tr>
<td>Pattern_Green</td>
<td>1byte</td>
</tr>
<tr>
<td>DAT1</td>
<td>2byte</td>
</tr>
<tr>
<td>DAT2</td>
<td>2byte</td>
</tr>
<tr>
<td>reserved</td>
<td>24hex-29hex</td>
</tr>
</tbody>
</table>

Table 2 The Register Overview FPGA SA/EMC Tool
The software for the Automotive video generating hardware platform

So as the primary solution was planned and according to the Table 2 which was mentioned above, the development of the GUI about the Video Setting Function (Figure 33), Picture Generator Function (Figure 34) and PLL Calculation Function (Figure 35) were implemented into the software of the GUI one by one.
For developing the Video Setting Function, the index number of the “25hex” was taken for arranging the registers which would be used in the video setting. The result of the updating is shown in the Table 3 below.

<table>
<thead>
<tr>
<th>Picture generator</th>
<th>25hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Picture</td>
<td>1byte</td>
</tr>
<tr>
<td>Color_R</td>
<td>1byte</td>
</tr>
<tr>
<td>Color_G</td>
<td>1byte</td>
</tr>
<tr>
<td>Color_B</td>
<td>1byte</td>
</tr>
</tbody>
</table>

Table 3

And during the developing of the software of the GUI, sometimes, the complicated calculation would need to be implemented in the code. So before started to program the software, a flowchart would be quite necessary, which could help the developer to quickly build up the structure of the complicated calculation in the software.

The software flowchart in the Figure 36 was draw for the development of the PLL Calculation Function. In this flowchart, “(Base) frequency” is the value of the basic frequency of FPGA, and “x” is the frequency would be output after the PLL calculation. The “m” and “d” are represent the different “Multipliers” and “Dividers” in the PLL calculator, respectively. And the “r” is represent “result” of the actual frequency the PLL calculator output at the end of the calculation. And \( \Delta r \) is the difference between the expected frequency and the results was got during the calculator. The result with the smallest difference will be output as the final output frequency, but it will not be shown in the GUI interface and also will not be transmitted to the MCU. Only the “Multiplier” and “Divider” of this result will be shown in the GUI and transmitted to the MCU as the parameters named “PLL1” and “PLL2” (please refer to Figure 35).

After finished the development of the Kleine_Friet with the Video Setting Function, Picture Generator Function and PLL calculation Function, most the attention of the development was paid on the software of the MCU and the SPI Block, also the connection between the MCU and the SPI Block or the Apix2 controller.

But on the GUI part, there are still several improvements were made on the interface of the Kleine_Friet. The power button was added and the background color was changed for the better user experience. Then for the Kleine_Friet can be used on the different PC, without any change in the “serial port connection” part in the code, the code and interface of the Kleine_Friet about this part were redesigned. The “serial port connection” part was updated into two separate functions. They are the Manual Serial Port Setting Function and the Automatic Serial Port Setting Function (please refer to Figure 37). And the more details about the Automatic Serial Port Setting Function part will be introduced in the Chapter 4.4.
The software for the Automotive video generating hardware platform

Start

(Base) frequency = 40;
(Output frequency) x = Input;
(Multiplier) m = 0, (Divider) d = 0;
(m1,2 = 0, d1,2 = 0);
(Result) r = 0, Δ r = 0,
Δ r1 = 0, Δ r2 = 0;

i = 1

i = i + 1; r = frequency * i; m1 = i

j = 1

j = j + 1; r = r / j;
d1 = j; Δ r1 = r - x;

i == 1 && j == 2 ?

Δ r1 = Min {Δ r1, |Δ r2|}?

d = d1; m = m1; Δ r = Δ r1

Δ r1 = Min {Δ r1, |Δ r2|}?

d = d2; m = m2; Δ r = Δ r2

i != 1 || i > 2 ?

|Δ r1| != |Δ r| && Δ r1 < Δ r

|Δ r1| != |Δ r|
The software for the Automotive video generating hardware platform

Figure 36 The software flowchart of the PLL Calculation Function

After the communications between the MCU and the SPI Block and between the MCU and the Apix2 controller was setup. The Readback Function of the Kleine_Friet was started to develop, which named “Apix2 Reading & Writing” on the tab page of the Kleine_Friet. There are three main parts in the Readback Function. They are list of registers reading, specific register reading and specific register writing. They can be found in the Figure 37 below.

Figure 37 The interface of the Kleine_Friet with the Apix2 Reading and Writing Function and the Automatic Serial Port Setting Function.
In the “list of registers reading” part, the user can read the values from a list of registers of the Apix2 controller, whose addresses were built into the as the default setting. And in the “specific register reading” part and “specific register writing” part, both of the value will read from the register and the value will wrote to the register are all can be represented into hexadecimal number and binary number on the interface of the Kleine_Friet.

<table>
<thead>
<tr>
<th>Reading &amp; Writing</th>
<th>26hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Reading</td>
<td>1byte</td>
</tr>
<tr>
<td>Register Reading / Writing</td>
<td>1byte</td>
</tr>
<tr>
<td>Register Address</td>
<td>1byte</td>
</tr>
<tr>
<td>Writing Value</td>
<td>1byte</td>
</tr>
<tr>
<td>reserved</td>
<td>26hex-29hex</td>
</tr>
</tbody>
</table>

Table 4

For developing the Readback Function, and also for the development of this function in the MCU software later, the index number of the “26hex” was taken for arranging the registers which would be used during the process of reading and writing the value from Apix2 controller. The result of the updating is shown in the Table 4 above. But by changing the priority of the project, the software development of the Readback Function in the GUI is not completely finished yet. It is still under development.

The software on the PC named HHD Serial port monitor was used for monitoring and checking the commands sent out from the GUI by USBSerial port during the debugging and testing phase. And later on, after the corresponding software for the MCU was finished programmed, this the HHD Serial port monitor also could monitor the feedback sent from the MCU to check the result of the development. And the LED lights on the MCU also were used to monitoring the process of the MCU software while it was running, for checking how the software responded to the commands sent from the GUI,
4.2 The software of the MCU

The software of the MCU in the primary solution could be divided into three main parts. They are the communication with GUI, the communication with FPGA (the SPI Block) and the communication with Apix2 controller. And during developing this primary solution, the Message LCD Menu Function and the File Reading / Writing Function were included into the MCU software.

The communication with GUI is the first part that was built up, after the development of Video Setting Function in the GUI was almost finished. An interrupt routine (See Code Sheet 1) was programmed in this part for receiving the command transmitted from the PC. The recognition and storing part was put into the infinite while-loop which is in the main routine, and it was enabled by one index register. And later on, all the commands transmitted by the GUI were all received by this interrupt routine.

```
void receive_pc( );   // Receive the command from the PC

int main(void)
{
    __disable_irq( );   // to avoid getting masked interupts
    while (serial.available( ) > 0 )  // Start to receive string to StringArray
    {
        input_string[i] = serialgetc( );
        i++;
    }
}
```

But after the development of Picture Generator Function was finished and the initialization data for the display was programmed into the MCU code, the software started to became a little messy. For making these functionalities became well-organized, the flag system was setup into the code of the software for ensuring the functionalities in the program could be under a unified control. The Code Sheet 2 is the latest definition of the flag system of the MCU software.

```
// Flag //
int lcd_inform_flag     = 0x01;  // Flag for showing the content of the information on the LCD
int menu_flag           = 0x00;  // Flag will be set on when the user enter the LCD menu
```
The software for the Automotive video generating hardware platform

```c
int enter_menu_flag = 0x00; // Flag will be set on after the user push the Enter Button
int exit_menu_flag = 0x00; // Flag will be set on after the user push the Exit Button
int up_flag = 0x00; // Flag will be set on after the user push the Up Button
int down_flag = 0x00; // Flag will be set on after the user push the Down Button
int state_flag = 0x01; // Flag for showing the statement of the value in the MCU
int action_flag = 0x00; // Flag for setting the operating action to the MCU
int apix2_act_flag = 0x00; // Flag for reading or writing the register of the Apix2
int led_flag = 0x00; // Flag for remember the statement of the LED on the MCU
```

**Code Sheet 2**

After the flag system was defined, the whole software of the MCU was started to improve. In the main routine, some if-statements were programmed into that infinite while-loop for checking the statement of flags. The real process of the communication between the MCU and the GUI, such as the recognition, storing and sending parts were all moved into the sub-function, which would only be executed when the conditions of the if-statement were matched. (See **Code Sheet 3**)

```c
int main(void)
{
    ...........
    while(1)
    {
        if(previous_i != i)
        {
            receive_command_gui();
            previous_i = i = 0; //Reset the index
        }
        // Sending the INIT Code in LSB fisrt
        if (((action_flag & 0x01) == 1 || (state_flag & 0x01) == 1)
        {
            send_init_spi();
            state_flag &= 0xFE; // Set "INIT code" flag off, equals to xxxx xxx0.
        }
        // Sending command to FPGA by SPI protocol
        if (((action_flag & 0x02) >> 1) == 1 || ((state_flag & 0x02) >> 1) == 1)
        {
            send_fpga_spi();
            state_flag &= 0xFD; // Set "FPGA code" flag off, equals to xxxx xx0x.
            last_menu_infterface(); // Back to the last infterface of the LCD menu
        }
        ...........
    }
}
```

**Code Sheet 3**

Then at the same time of the development phase of the SPI protocol (See Chapter 4.4) was finished, the development on the parts about the communication with the FPGA and communication with the Apix2 were finished. Some sub-functions were built up in the program. After that, for users can control the MCU to send the
commands to components on the hardware platform, or send test pictures to the displays discretionarily, the message LCD menu was built up. The four push buttons on the hardware platform were used as the “Enter Button”, “Exit Button”, “Up Button” and “Down Button”. Users can through control these four buttons to enter or exit the menu level by level, and select the items of the menu to send initialization commands to FPGA or displays, send the test picture to the displays, initialize the USB connection and generate the configuration file within the MCU. The primary specification designed for the interface of the menu is in the Appendix C.

On the other hand, based on the software was developed in the MCU, there are two independent program were developed and succeed with using into the stand-alone tests of the new automotive displays from BMW. One is for the EMC test, and the other one is for the GAMMA test. But by changing the priority of the project, the software development of the Readback Function in the MCU is not completely finished yet. It is still under development now.

During the development of the software running on the MCU, the LED lights on the MCU also were used to monitoring the process of the MCU software while it was running, also for checking how the software responded to the commands sent from the GUI. And the Oscilloscope was introduced briefly in the Chapter 3.2.3 was used for monitoring the signal output by the SPI protocol of the MCU during the testing phase and debugging phase.
4.3 The software of SPI Block

In the primary plan, there is no requirement of the software development about the SPI Block. But later, this part was added into the whole internship assignment as an extra design assignment, for the reason that it would be easier to develop the software of the SPI Block and the software of the MCU together by the same developer.

The functionality requirement of the SPI Block software is not quite complicated. The functions should be enabled into this block is receiving and recognizing the commands transmit from the MCU by SPI protocol and storing them into several specific registers.

So in the design, there are one top-module including three blocks and two sub-modules in this software.

The first block of the top-module (see Code Sheet 4) will be enabled by the negative edge of the clock generated by the MCU SPI protocol, and this block is for controlling (enable, disable and reset) the shift register and the counter were described in the sub-modules. The “Dout_c” is the output of the counter; the “clr” is connected to the “reset” pin of the counter; the “cntEn” is connected to the “enable” pin of the counter and the “ce” is connected to the “enable” pin of the shift register.

```
// Enable and disable the shift register and the counter
always @ (negedge CLK_MCU)
begin
  if( Dout_c == 0 ) begin
    cntEn <= 1'b1;
    clr <= 1'b0;
    ce <= 1'b1;
  end
  if( Dout_c == 31 ) begin
    clr <= 1'b1;
    cntEn <= 1'b0;
    ce <= 1'b1;
  end
  else if( Dout_c == 32 ) begin
    clr <= 1'b0;
    cntEn <= 1'b1;
  end
  else begin
    clr <= 1'b0;
    cntEn <= 1'b1;
  end
end
```

Code Sheet 4

In the top-module, there are two sub-modules. They connected to each other by the code in the Code Sheet 5 below. One of the sub-modules is described as the counter (see Code Sheet 6), and the other is described as the shift register (see Code Sheet 7). In the primary design, they are all just enabled by the negative edge of the clock generated by the MCU. But in the debugging, the running of these two sub-modules was not desirable enough. So the signal of the chip select was improved into the sensitive list of the “always” statement in these two sub-modules.
The software for the Automotive video generating hardware platform

---

Code Sheet 5

```vhdl
// Shift register
shift u0(CLK_MCU, CS_MCU, ce, Slave_Input, so);
// Counter
count8 u2(CLK_MCU, CS_MCU, clr, cntEn, Dout_c);
```

---

Code Sheet 6

```vhdl
// an 8-bit counter with clear and count enable controls
module count8 ( 
  CLK,
  cs,
  clr,
  cntEn,
  Dout_c);
  input  CLK;
  input  cs;
  input  clr; // clear counter
  input  cntEn; // enable count
  output [7:0] Dout_c; // counter value
  reg [7:0] Dout_c;
  
  always @(negedge CLK) begin
    if(clr & ~cs) begin
      Dout_c <= 1;
    end
    // The count function only will work when cntEn == 1 and cs
    else if(cntEn & ~cs) begin == 0
      Dout_c <= Dout_c + 1;
    end
  end
endmodule
```

```vhdl
// a 32-bit shift-left register with a negative-edge clock,
// a clock enable, a serial in and a serial out.
module shift ( 
  clk,
  cs,
  ce,
  si,
  so
);

  input clk;
  input cs;
  input si;
  input ce;

  // output LED1;
  output [31:0] so;
  reg [31:0] so;
  // The shift register function only will work when ce == 1
  and cs == 0
  always @(negedge clk)
  begin
    if (ce & ~cs) begin
      so <= so << 1;
      so[0] <= si;
    end
  end
endmodule
```
Then it’s the second block of the top-module (see Code Sheet 8). It is controlled by the negative edge of the clock generated by the FPGA itself. Its frequency is faster than the frequency of the clock generated by the MCU. So the block can keep working between the two negative edges of the MCU clock and also after the MCU clock stopped. This block is special for assigning the data from the output of the shift register (so) to the registers of the recognition part.

```
always @ (negedge CLK_4)
begin
  if ( Dout_c == 32 ) begin
    // Take the index value for reorganization
    data_recognize <= so[31:16];
    data_receive <= so;
  end
  else if ( Dout_c == 16 ) begin
    // Clear the transitional register
    data_recognize <= 16'h0;
    data_receive <= 32'h0;
  end
end
```

About the third block of the top-module (see Code Sheet 9), it is controlled by the negative edge of the FPGA clock as the previous block. This block is special for recognizing the commands received, then storing into the specific registers in the FPGA.

```
// Recognize and assign the value from the transitional register to the specific register
always @ (negedge CLK_4)
begin
  if (data_recognize == 16'h1) begin
    Base_frequency <= data_receive;
  end
  else if (data_recognize == 16'h2) begin
    PLL1 <= data_receive;
  end
  else if (data_recognize == 16'h3) begin
    PLL2 <= data_receive;
  end
  else if (data_recognize == 16'h4) begin
    PLL3 <= data_receive;
  end
  else if (data_recognize == 16'h5) begin
    H_PW <= data_receive;
  end
  else if (data_recognize == 16'h6) begin
    H_FP <= data_receive;
  end
  else if (data_recognize == 16'h7) begin
    H_DV <= data_receive;
  end
  else if (data_recognize == 16'h8) begin
    H_BP <= data_receive;
  end
  else if (data_recognize == 16'h9) begin
```
The several probes of the In-System Sources and Probes Editor of Quartus II 12.1 were used for monitoring the signal was received by the internal registers of the SPI Block during the testing and debugging. And also the Oscilloscope was used for monitoring the signal output by the SPI protocol of the MCU during the testing phase and debugging phase.
4.4 All the connections

About the connections part, as the primary solution was planned, the connection between the PC and the MCU, and the connection between the MCU and the message LCD was finished to setup at first, which are the connections are showed in the red circle of the Figure 38.

![Figure 38](image)

The connection between the PC and the MCU, and the connection between the MCU and the message LCD (red circle)

And of course, in the both of the GUI software and the MCU software, the USBSerial Port were all declared or set in their own definition part in the code, respectively. In the GUI code there are several lines code are special for opening and closing this USBSerial Port. And later of the GUI software development, the Manual Serial Port Setting Function and the Automatic Serial Port Setting Function were built into the GUI code, which made the GUI became much more friendly to the users.

![Figure 39](image)

The interface of the WMI Code Creator
About the design of the Serial Port Setting Function, the manual one is not difficult to set up based on the former version of the GUI software already been programmed before, but about the automatic one, the method of the developing is not quite apparent at the beginning of the development.

A lot of information was searched from the internet. And the tool named WMI Code Creator was mentioned in the most of the related website, which allows the developers to generate VBScript, C#, and VB .NET code that uses WMI to complete a management task such as querying for management data, executing a method from a WMI class, or receiving event notifications using WMI.

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Management;
using System.Windows.Forms;

namespace WMISample
{
    public class MyWMIQuery
    {
        public static void Main()
        {
            try
            {
                ManagementObjectSearcher searcher =
                    new ManagementObjectSearcher("root\CIMV2",
                        "SELECT * FROM Win32_SerialPort WHERE Caption = 'Mbed Virtual Serial Port (COM5)');

                foreach (ManagementObject queryObj in searcher.Get())
                {
                    Console.WriteLine("-----------------------------");
                    Console.WriteLine("Win32_SerialPort instance");
                    Console.WriteLine("-----------------------------");
                    Console.WriteLine("Caption: {0}", queryObj["Caption"]);
                }
            }
            catch (ManagementException e)
            {
                MessageBox.Show("An error occurred while querying for WMI data:");
            }
        }
    }
}

Code Sheet 10

Then code generated by this tool (See **Code Sheet 10**) was updated in several parts to fit the requirements of the Automatic Serial Port Setting Function (See **Code Sheet 11**). And after a little debugging, and fixed some bugs in the code, the development of this function was finished. And the LED lights on the MCU were used to monitoring the process of the MCU software while it was running.

using System;
using System.IO;

void AutoPortSetting()
{
    ProgressBar.Value = 0;
    ProgressBar.Maximum = 5;
    try
    {
        ManagementObjectSearcher searcher = new ManagementObjectSearcher("root\CIMV2", "SELECT * FROM Win32_SerialPort");

        foreach (ManagementObject queryObj in searcher.Get())
        {
            if (queryObj["Caption"].ToString().Contains("Mbed Virtual Serial Port"))
            {
                ProgressBar.Value = 3;
                MessageBox.Show("" + queryObj["Caption"].ToString() + " will be open");
                ProgressBar.PerformStep();

                if (!serialPort1.IsOpen)
                {
                    serialPort1 = new SerialPort();
                    serialPort1.PortName = queryObj["DeviceID"].ToString();
                    TextBoxPortname2.Text = queryObj["DeviceID"].ToString();
                    SerialPort1Preparation();
                }

                if (serialPort1.IsOpen)
                {
                    // Change the statement of the some items on the form
                    ChangSubInterFaceItemStatement(true);
                    // Change the ReadOnly statement of these textboxes
                    ChangeTextBoxReadOnly(false);
                    // Call the PictGen1stByte Function to initialize FPGA P.G Part
                    PictGen1stByte();
                    serialPort1.DataReceived += new SerialDataReceivedEventHandler(serialPort1_DataReceived);
                }
            }
        }
    }
    catch (ManagementException exception)
    {
        MessageBox.Show("An error occurred while querying for WMI data: " + exception.Message);
    }
    if (ProgressBar.Value == 0)
    {
        MessageBox.Show("Please check your USB connection");
    }
}

Next connection need to be setup is the SPI protocol. It should not only be built up between the MCU and the SPI Block, but also between the MCU and the Apix2 controller. Because there was no any requirements temporarily from the customer at that moment, which specify that \( \text{i}^2 \text{C} \) have to be enabled between the MCU and the Apix2 controller. So the more familiar protocol, the SPI one was chosen to connect the MCU and the Apix2. So as the result of it, the \( \text{i}^2 \text{C} \) was moved out of the design assignment.
The connection between the MCU and the SPI Block in the FPGA was not difficult to be built up. What should be done just wrote the right definition into the MCU code and set the format of the SPI protocol into the correct style. As the code shown in the Code Sheet 12, the SPI transmission was set for 16 bit data with 2000HZ, the clock would start from low state, and the data would be captured at the second edge.

```
SPI spi(p5, p6, p7);  // mosi, miso, sclk
DigitalOut cs(p16);   // chip select

cs = 1;  // Deselect the chip
// Setup the spi for 16 bit data, low steady state clock,
// second edge capture, with a 2000Hz clock rate
spi.format(16,1);
spi.frequency(2000);
wait_us(2000000);

cs = 0;

// BaseFreq send in MSB first
output_string[0] = 0x01;
output_string[1] = BaseFreq;
spi.write(output_string[0]);
wait_us(2000);
spi.write(output_string[1]);
wait_us(2000);

spi.format(8,1);
spi.frequency(320000);
wait_us(2000000);

cs = 1;
```

Code Sheet 12

And the connection between the MCU and the Apix2 controller actually also was not difficult to be built up. But as the result that the Apix2 is a rather new component and there are several configuration modes is available in it, so a long-term debugging phase was taken to succeed with getting communication between the MCU and Apix2.

The code shown in the Code Sheet 13 could be introduced separately into two parts, one is the Apix2 configuration part, and the other is the command transmission part. And according to the SPI configuration and transmission requirement in the specification of the Apix2 controller, the SPI transmission was set for 8 bit data with 320000HZ, the clock would start from low state, and the data would be captured at the first edge.

```
SPI apix_spi(p11, p12, p13);  // mosi, miso, sclk
DigitalOut apix_cs(p15);     // chip select
DigitalIn spi_s_stall(p14);  // SPI_S_STALL
```
void send_apix_spi();
void send_init_spi();
void spi_write_displays(const unsigned int*);

// Subfunction for The Variable for Apix //
void send_apix_spi()
{
    ..........  
apix_cs = 1;  // Deselect the chip
    // Setup the spi for 8 bit data, low steady state clock,
    // first edge capture, with a 320kHz clock rate
    apix_spi.format(8,0);
apix_spi.frequency(320000);
    wait_us(2000);

    /* Apix2 Initialize with the "Burst Write" Method */
apix_cs = 0; // No any delay before the data comes
    for (spi_index= 0; spi_index <= 34; spi_index ++ )
        {
            const unsigned int apix_value = INIT_APIX_0[spi_index];
apix_spi.write(apix_value);
        }
    wait_us(1500); // For waiting the SPI handler send out all the value
    apix_cs = 1;
    ...........
}

// Subfunction for Sending Out The Initial Code for The Display //
void send_init_spi()
{
    ..........  
apix_cs = 1;  // Deselect the chip
    // Setup the spi for 8 bit data, low steady state clock,
    // first edge capture, with a 320kHz clock rate
    apix_spi.format(8,0);
apix_spi.frequency(320000);
    wait_us(2000);

    spi_write_displays(INIT_A);
    spi_write_displays(INIT_B);
    wait_us(1000000);
    ...........
}

// Sub-subfunction for Sending Out The Initial Code for The Display
void spi_write_displays(const unsigned int* send_value)
The software for the Automotive video generating hardware platform

```c
{  
    wait(0.005);
    if (spi_s_stall) wait(0.005);
    else if (!spi_s_stall)
    {
        apix_cs = 0; // No any delay before the data comes
        wait_us(0.1);
        apix_spi.write(0xF7);
        for (spi_index= 6; spi_index >= 0; spi_index -- )
        {
            const unsigned int apix_value = send_value[spi_index] ;
            apix_spi.write(apix_value);
        }
        wait_us(15); // For waiting the SPI handler send out all the value
        apix_cs = 1;
    }
}
```

**Code Sheet 13**

In the Apix2 configuration part, the function called “send_apix_spi” in the **Code Sheet 13**, where the MCU send the data to configure the Apix2 controller, the software of the MCU used the “Burst Write” method (please refer to the introduction in the **Chapter 2.4.2**) for writing the configuration command to the Apix2. After the chip selected pin (apix_cs) was low active, the data could be transmitted directly. There is no any delay need to be put between them. But according to the monitoring of the SPI protocol, after the transmission finished, the SPI handler in the MCU still need some time to send out the data. So, the delay was put in the code before the chip select went high.

And in the command transmission part, the function called “send_init_spi” in the **Code Sheet 13**, which function that the MCU send the data to initialize the display and also. Something special is that there is one pin on the MCU was picked for checking the “SPI_S_STALL” signal of the Apix2. Because in the AShell mode (please refer to the introduction in the **Chapter 2.4.2**), the data only can be transmitted not only while “cs” pin is low, but also while the “SPI_S_STALL” pin is low. When the transmission started, the program would wait for 0.005 seconds first then checked this pin. If the “SPI_S_STALL” pin was high that time, the program would wait for 0.005 seconds again. And these two delays would be enough to the transmission according to results of the several tests.

During the testing and debugging phase, the Oscilloscope was used for monitoring the signal transmitted by the SPI interface on the MCU, including the MOSI pin, CLK pin and CS pin. And also the LED lights on the MCU also were used to monitoring the process of the MCU software while it was running.
5. Results

The results of this project are presented in the Table 5 below. By changing the priority of the project, the software development of the Readback Function and Configuration Function in the GUI are not completely finished yet. They are still under development.

<table>
<thead>
<tr>
<th>Item</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The software of the GUI in C#</strong></td>
<td></td>
</tr>
<tr>
<td>The Video Setting Function</td>
<td>Done</td>
</tr>
<tr>
<td>The Picture Setting Function</td>
<td>Done</td>
</tr>
<tr>
<td>The PLL Calculation Function</td>
<td>Done</td>
</tr>
<tr>
<td>Automatic Serial Port Setting Function</td>
<td>Done</td>
</tr>
<tr>
<td>The Readback Function</td>
<td>Not yet</td>
</tr>
<tr>
<td>The Configuration Function</td>
<td>Not yet</td>
</tr>
<tr>
<td><strong>The software of the MCU in C++</strong></td>
<td></td>
</tr>
<tr>
<td>Receive the commands from the PC</td>
<td>Done</td>
</tr>
<tr>
<td>Recognize the commands from the PC</td>
<td>Done</td>
</tr>
<tr>
<td>Store the commands received to the registers</td>
<td>Done</td>
</tr>
<tr>
<td>Send the feedback to the GUI</td>
<td>Done</td>
</tr>
<tr>
<td>Send the commands to FPGA</td>
<td>Done</td>
</tr>
<tr>
<td>Send the commands to Apix2 Controller</td>
<td>Done</td>
</tr>
<tr>
<td>Build up the message LCD menu (Extra)</td>
<td>Done</td>
</tr>
<tr>
<td>Initialize the USB connection (Extra)</td>
<td>Done</td>
</tr>
<tr>
<td>Load and save the configuration file (Extra)</td>
<td>Done</td>
</tr>
<tr>
<td><strong>The software of the SPI Block in the FPGA (Extra)</strong></td>
<td></td>
</tr>
<tr>
<td>Receive the commands from the MCU</td>
<td>Done</td>
</tr>
<tr>
<td>Recognize the commands from the MCU</td>
<td>Done</td>
</tr>
<tr>
<td>Store the commands received to the registers</td>
<td>Done</td>
</tr>
<tr>
<td><strong>All the Connection between all the components</strong></td>
<td></td>
</tr>
<tr>
<td>Between the PC and the MCU</td>
<td>Done (USBSerial protocol)</td>
</tr>
<tr>
<td>Between the MCU and the message LCD</td>
<td>Done (According to the specification of the LCD)</td>
</tr>
<tr>
<td>Between the MCU and the FPGA</td>
<td>Done (SPI protocol)</td>
</tr>
</tbody>
</table>
The software for the Automotive video generating hardware platform

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Between the MCU and the Apix2 controller</td>
<td>Done (SPI protocol)</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td></td>
</tr>
<tr>
<td>Develop the software of the MCU for the EMC Test (Stand-alone, without PC)</td>
<td>Done</td>
</tr>
<tr>
<td>Develop the software of the MCU for the GAMMA Test (Stand-alone, without PC)</td>
<td>Done</td>
</tr>
</tbody>
</table>

**Table 5** The results of the internship assignment
6. Discussion

The results of this assignment and the method that was used during this internship all will be discussed in this chapter.

6.1 Results

6.1.1 The software of the GUI

Compared with the expectation from the company, some of the functionalities in the software of the GUI already have been finished. But there are two functions still have to be implemented into this software, the Readback Function and Configuration Function. So the Kleine_Friet need to be further developed.

6.1.2 The software of the MCU

About the software of the MCU, the functionalities were required by the company on this part are almost finished. The only thing should be improved into the MCU is the functionality which should be able to support the Readback Function of the GUI. But if just use this software of the MCU within the stand-alone tests (without PC), it would perform perfectly.

6.1.3 The software of the SPI Block

The software of the SPI Block is completely finished. The functionalities that required by the company on this part already have been reached totally.

6.1.4 All the connections

As a result the requirement of I²C protocol was moved out of this assignment, the connections were setup (USBSerial protocol and SPI protocol) are already enough to the new requirements. The communication between the GUI and MCU, the MCU and SPI Block, and between the MCU and Apix2 controller all perform well.

6.2 Method

The method was used for this assignment is really helpful. Everything might need for the real design phase became well-prepared after the first three steps were finished. So this method helped a lot on each single step of the design and made the process went pretty well.
7. Conclusion

The GUI implemented in C# running on a PC was developed. It is able to send the commands control the PC to send the commands to the MCU. They connect to each other by USBSerial port, which is used to emulate a serial port over USB. But by changing the priority of the project, the Readback Function in the GUI is not completely finished. It is still under development. So the GUI still cannot receive the feedback from the MCU yet.

On the other hand, the embedded software implemented in C++ running on the MCU was also developed. Now, it is able to let the MCU receive the commands from the PC and send the feedback to the PC. And the communication between the MCU and all components from the hardware platform are only setup by the SPI protocol, and they perform pretty well. The I²C protocol which was planned to connect the MCU and Apix2 controller was moved out of this assignment, because the priority of the project was changed.

Furthermore, as an extra assignment, the embedded software implemented in Verilog running on the SPI Block of the FPGA was developed within this internship assignment. It is able to receive the command from the MCU by SPI interface and store them in the specific register of the FPGA.
8. Recommendation

The compiler was used for developing the embedded software of the MCU is an online compiler from the mbed official website. As an online application, it has some limitations on debugging the embedded software. It cannot stop the running software by the “block function” whenever the developer want, then check the value of some registers as many as needed. The only thing can do in the development of the MCU software is programming some code in the software, and using them to display the value of registers on the message LCD of the hardware platform.

So if the development cost is available, it would be better to introduce an IDE software application into this kind of embedded software development. Then the whole debugging phase of the software development would become much more efficient.
9. Reference

[1] Dan Ros; mbed NXP LPC1768 Getting Started (retrieved on: 25 May 2013)
   https://mbed.org/handbook/mbed-NXP-LPC1768-Getting-Started


   https://en.wikipedia.org/wiki/Microsoft_Visual_Studio


   http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

   http://en.wikipedia.org/wiki/C_Sharp_(programming_language)


[8] Dan Ros; mbed NXP LPC1768 Getting Started (retrieved on: 25 May 2013)
   https://mbed.org/handbook/mbed-NXP-LPC1768-Getting-Started

[9] Dan Ros; mbed NXP LPC1768 Getting Started (retrieved on: 25 May 2013)
   https://mbed.org/handbook/mbed-NXP-LPC1768-Getting-Started

    https://mbed.org/handbook/mbed-Compiler

    https://mbed.org/handbook/mbed-Compiler

The software for the Automotive video generating hardware platform


Bjarne Stroustrup (retrieved on: 26 May 2013)
http://en.wikipedia.org/wiki/Bjarne_Stroustrup

C ++ (retrieved on: 26 May 2013)

[15] Deepak Kumar Tala;
Verilog Tutorial (retrieved on: 26 May 2013)
http://www.asic-world.com/verilog/veritut.html

[16] Wikipedia;
Serial Peripheral Interface Bus (retrieved on: 27 May 2013)

Serial Peripheral Interface Bus (retrieved on: 27 May 2013)

[18] Wikipedia;
Serial Peripheral Interface Bus (retrieved on: 27 May 2013)

[19] Inova Semiconductors;
APIX2 – Automotive Pixel Link (retrieved on: 29 May 2013)
http://www.inova-semiconductors.de/en/products_apix2.html

[20] Inova Semiconductors;
INAP375T User Manual, Revision 1.2
November 30, 2012, pp.15

[21] Inova Semiconductors;
INAP375T User Manual, Revision 1.2
November 30, 2012, pp.16

[22] Inova Semiconductors;
INAP375T User Manual, Revision 1.2
November 30, 2012, pp.17

[23] Inova Semiconductors;
INAP375T User Manual, Revision 1.2
November 30, 2012, pp.101

[24] Inova Semiconductors;
INAP375T User Manual, Revision 1.2
November 30, 2012, pp.101
The software for the Automotive video generating hardware platform

[25] Inova Semiconductors;
   INAP375T User Manual, Revision 1.2
   November 30, 2012, pp.104

[26] HTO-course AMLCD, Philips Flat Panel Displays, Jean Lorteije,
   Chapter 1.1 Introduction Flat Screen Displays, Version 1
   January 1999, pp.10

[27] Wikipedia;
   Friedrich Reinitzer (retrieved on: 28 May 2013)
   http://translate.google.nl/?hl=nl&tab=wT#en/zh-CN/Friedrich%20Reinitzer

[28] HTO-course AMLCD, Philips Flat Panel Displays, Jean Lorteije,
   Chapter 1.1 Introduction Flat Screen Displays, Version 1
   January 1999, pp.10

[29] HTO-course AMLCD, Philips Flat Panel Displays, Jean Lorteije,
   Chapter 1.1 Introduction Flat Screen Displays, Version 1
   January 1999, pp.13

[30] Microsoft;
   Download Page of WMI Code Creator V1.0 (retrieved on: 28 May 2013)
The software for the automotive video generating hardware platform

Final Report – Appendixes

Author: Xiaobo Qu
School: HZ University of Applied Sciences
Programme: Bachelor of Engineering in Mechatronics
Supervisor teacher: Bert Verhage
Company: Innolux - TPO Displays Europe B.V.
In-company mentor: Bart Peters
Date of publication: 30-05-2013
Place of publication: Heerlen, The Netherlands
Appendix A
The electronic circuit diagram of the automotive video generating hardware platform
The software for the Automotive video generating hardware platform
Appendix B
The specification of the message LCD on the hardware platform
The software for the Automotive video generating hardware platform
Appendix C

The specification of the message LCD menu

USBSerial Statement

<table>
<thead>
<tr>
<th>Non-connection</th>
<th>Connection</th>
<th>lcd_inform_flag = 0x01</th>
</tr>
</thead>
<tbody>
<tr>
<td>No USBSerial</td>
<td>USBSerial</td>
<td></td>
</tr>
</tbody>
</table>

Operation Menu

The First Level

Button1: Enter the menu  
lcd_inform_flag = 0x02

The Second Level

> Initialize  
  Picture Select  
  Initialize  
  > Picture Select  
  Picture Select  
  > Save Settings  
  lcd_inform_flag = 0x03

The Third Level (Initialize)

> FPGA  
  Display  
  FPGA  
  > Display  
  Display  
  > USB  
  lcd_inform_flag = 0x06

The Third Level (Picture Select)

> Picture name 1  
  Picture name 2  
  > Picture name 2  
  Picture name 3  
  > Picture name 3  
  Picture name 1  
  lcd_inform_flag = 0x09