CHALMERS

Ethernet to HDMI Gateway

A way to transfer video data over long distance

Master of Science Thesis in the Programme Integrated Electronic System Design

SIMON KEKKONEN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, February 2012

Ethernet to HDMI Gateway
A way to transfer video data over long distance

Simon KEKKONEN

Examiner: Sven Knutsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden February 2012

Acknowledgements

While pursing my Master of Science I have relied on the support, encouragement, friendship and guidance of many people. I sincerely would like to thank several persons, especially my family, for their help during my years of study.

I would like to thank my supervisor Sven Knutsson for his constructive comments during this project.

I would like to thank Björn Bergholm for giving me the opportunity to doing this project at Broccoli Engineering AB. I would also thank other employees at Broccoli that has helped me during the project.

Abstract

Today the demands on electronic products are constantly increasing. This project deals with transmission of video data, especially over long distance. A device that could be able to increase the length from about 7 m to about 100 m when sending video was to be designed in this project. The task was to extend the length of a HDMI cable. The idea was to convert the signal from HDMI to Ethernet frames and sent it as Ethernet frames over a longer distance and then convert back to HDMI. In this project the Ethernet to HDMI conversion was designed. The device was based on a FPGA and should be able to receive Ethernet frames with video data and sends out the video data as a HDMI signal.

The result is not so good, but it is easily seen from the results that the device receives Ethernet frames and then sends out the data to a screen.

Table of Contents

List of Abbreviations

BGA	Ball Grid Array
CEC	Consumer Electronics Control
CMOS	Complementary Metal–Oxide–Semiconductor
COL	Collision Detected
CRC	Cyclic Redundancy Check
CRS	Carrier Sense
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DCCP	Datagram Congestion Control Protocol
DDC	Display Data Channel
DDR2 SRAM	Double Data Rate Synchronous Dynamic Random-Access Memory
DVI	Digital Visual Interface
FCS	Frame Check Sequence
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
FTP	File Transfer Protocol
G	Pre-defined polynomial for CRC
g	Pre-defined polynomial for CRC, binary representation
GMII	Gigabit Media Independent Interface
GPIO	General Purpose Input / Output
GTXCLK	1 000 Mb/s Transmit Clock

HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HDTV	High-Definition Television
HID	Human Interface Device
HPD	Hot Plug Detect
HTTP	Hypertext Transfer Protocol
I^2C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IP	Internet Protocol
IPG	Inter Packet Gap
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
LSB	Least Significant Bit
M	Ethernet frame, from dest. address to and including the data field, polynomial representation
m	Ethernet frame, from dest. address to and including the data field, binary representation
MAC	Media Access Control
MDC	Management Data Clock
MDIO	Management Data Input/Output
MII	Media-Independent Interface
MSB	Most Significant Bit
PC	Personal Computer
PHY	Physical Layer
Q	Quotient when dividing $\frac{T}{G}$

R	FCS, polynomial representation
r	FCS, binary representation
RCA	Radio Corporation of America
RXCLK	Receive Clock
RXD	Receive Data
RXDV	Receive Data Valid
RXER	Receive Error
SCART	*Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs*, Radio and Television Receiver Manufacturers' Association
SCL	Serial Clock
SDA	Serial Data Signal
SFD	Start Frame Delimiter
SPI	Serial Peripheral Interface Bus
SVGA	Super Video Graphics Array
T	Ethernet frame, from dest. address to and including FCS, polynomial representation
t	Ethernet frame, from dest. address to and including FCS, binary representation
TCP	Transmission Control Protocol
TMDS	Transition-Minimized Differential Signaling
TXCLK	Transmit Clock
TXD	Transmit Data
TXEN	Transmit Enable
TXER	Transmit Error
UART	Universal Asynchronous Receiver / Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus
VGA	Video Graphics Array

VHDL	VHSIC (Very-High-Speed Integrated Circuits) Hardware Description Language
XST	Xilinx Synthesis Technology

Chapter 1

Introduction

In this chapter an introduction to the project will be given. It starts with a background where some different standards for sending video data are mentioned. Then a project description will explain the task for the project. The chapter ends with purpose, objectives and method.

1.1 Background

Today, consumer electronic is used by almost everyone. The demand for better products increases every day. When the products gets better, like TV-screens and DVD-players, the cables that carries the video and audio data also needs to get better. To transport data between two units without any distortion is important. To be able to transmit this data, a numbers of different signal types and cables has been developed from the 1950s until today. In table 1.1 a few of all the standards that have or are being used are shown.

The signals can be divided into two forms, these are analog and digital form. The three first standards in table 1.1 are analog. The two at the bottom are digital. Finally DVI (Digital Visual Interface) can handle both analog and digital signals. Some signal types can be transmitted by using different types of cables. Take for example DVI and HDMI (High-Definition Multimedia Interface). For the digital part DVI uses the same protocol as HDMI does. So the signal looks the same but the cables are very different.

A less clear distinction between signals/cables can be made, those that are used for consumer electronics and those that are used for PC (Personal Computer) systems. The two first in table 1.1, Composite video and SCART (*Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs*, Radio and Television Receiver Manufacturers' Association), are mostly used for television in home-use, while VGA (Video Graphics Array), DVI and DisplayPort are mostly used for PC systems. HDMI and to some extent DisplayPort are used both for TV in home-use and for PC-systems.

Composite video can be sent using a RCA (Radio Corporation of America) cable, which is quite small. All data are sent on one wire, which will not results in a very good picture. SCART can carry both video and audio data and is very common in Europe. SCART include both composite

Table 1.1: Shown some different signals standards [4].

Standard Name	**Year Introduced**	**Connector**	**Analog or Digital**	**Used For**
Composite video	1956	1 RCA, BNC, TV Aerial Plug, or Mini-VGA	Analog	Consumer electronics, including VCR and LaserDisc, 1970-1980s home computers like the Commodore VIC-20, 1980s-1990s video game consoles, some laptops.
SCART	1977	SCART 21-pin	Analog	Consumer electronics.
(Video Graphics Array) VGA	1987	VGA connector variants include DE-15/HD-15 (canonical), DE-9, RGB or RGBHV on separate BNC connectors, Mini-VGA, DVI/Mini-DVI/Micro-DV	Analog	Introduced with IBM x86 machines, but became a universal analog display interface. Display Data Channel was later added to allow monitors to identify themselves to graphic cards, and graphic cards to modify monitor settings.
Digital Visual Interface (DVI)	1999	DVI, Mini-DVI, Micro-DVI	Both	Recent video cards.
High-Definition Multimedia Interface (HDMI)	2003	19 pin HDMI Type A/C	Digital	Many A/V systems and video cards.
DisplayPort	2007	20-pin (external), 32-pin (internal)	Digital	Apple Inc. Lenovo, HP, and Dell systems and monitors ATI RV670 based graphics cards and NVIDIA G92 graphics cards.

video data signals and something called component video which has one wire for each of the three colours in the RGB colour model. To be able to fit all the signals into this cable the connector for SCART is quite big. These two signal types are commonly used for carrying video data, for example between a DVD-player and a television set and can also transfer audio data on the SCART cable.

The VGA standard is an analog signal that could be connected using a 15 pin D-sub connector. It can only send video. This signal type is manly used for PC systems and exist on almost every PC [1]. A standard that can send both analog and digital video is DVI. For the analog part of the DVI the VGA standard is used. The connector for DVI has the same size independent if analog

or digital data shall be sent, but the numbers of pins in the connector is different and the pins are placed differently in the connector. It is used for PC systems. Both the VGA connector and the DVI connector are quite big, but the DVI is the largest of the two.

The newest standard in table 1.1 is DisplayPort. It can carry both video and audio data. It can be used both for television in home-use or for PC systems. The connector is quite small. This has similar functionality to HDMI.

One standard that has become more used in recent years is HDMI. HDMI carries digital video and audio data. It can be used both for television in home-use and in PC systems. The connector is quite small [2].

All of the standards that are mentioned above are still in use, even if new and better standards has been developed, the old ones are needed for old equipment that can not handle the new standards.

In this project the focus will be on HDMI, which is a relatively new standard. But it experiences some limitations. One of them is that the length of the cable can not be that long. The maximum length of a HDMI cable is 15 m, but it is rare to find any cables longer than 7.5 m in a store [3].

To be able to extend the length between a HDMI source and a HDMI sink the medium the video data are sent on must be chanced. The medium that is used for this is Ethernet cables. The maximum length for Ethernet cables is about 100 m [5].

1.2 Project Description

This project will manly be about designing a FPGA (Field-Programmable Gate Array) that can handle a specific task. It involves HDMI signals and Ethernet signals.

The FPGA shall be able to receive Ethernet frames and save the data to a memory. Then it shall read from the memory, organize the data and send it out on the HDMI.

In this way it is possibly to send video data over long distance. It could be used in a already existing Ethernet network. There will also be possible to connect several screen to the network for displaying the same video image.

1.3 Purpose

The purpose with this project is to design a FPGA so that it can receive Ethernet frames with video data and save it to a memory and then read it and convert it to HDMI compatible signal and send it out.

The purpose with the prototype is to make it easy for the user to send video data from some Ethernet source to a HDMI compatible screen, regardless of the user wants so send video over long distance or to several screens on the network.

1.4 Objectives

The goal with the project is to get a working prototype according to the project description.

1.5 Method

The master thesis started with defining the task in more detail. When that was done research in the different parts of the project was done. Based on some of this research a development board was chosen.

The design was split into small parts that was developed separately. After each major step the design was tested and it was checked that it worked as intended, both in simulation and in hardware.

At the end all parts were combined so that all parts of the intended task was taken care of. The whole design was then implemented in the FPGA and tested.

Chapter 2

Overview of the System

This chapter describes how the entire system works and the main parts of the system. The chapter will also describe the development board and the development environment that was used.

2.1 Overview

The main task in the project is to design a device that can receive Ethernet frames and transform them into a HDMI signal that is sent out from the device. The device is based on a FPGA. Figure 2.1 shows a simple overview of the device with its input and output.

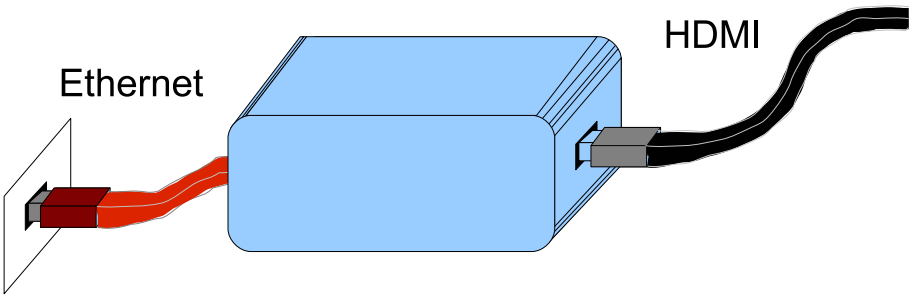


Figure 2.1: Simple overview of the device with its input and output.

The system is fed with Ethernet frames through a Ethernet cable. On the output side the systems is generating a HDMI signal through a HDMI cable.

In figure 2.2 a block diagram over the entire system is shown. The FPGA is divided into three large parts. These parts are Ethernet, HDMI and Memory. Each of them will handle its own task. The Ethernet module will handle communication to the outside world via an Ethernet controller but also communication with the memory. The HDMI module will handle communication with the outside world via a HDMI buffer and also to the memory. Finally the memory module will

handle communication with both Ethernet and HDMI and the DDR2 SRAM (Double Data Rate Synchronous Dynamic Random-Access Memory) memory on the board.

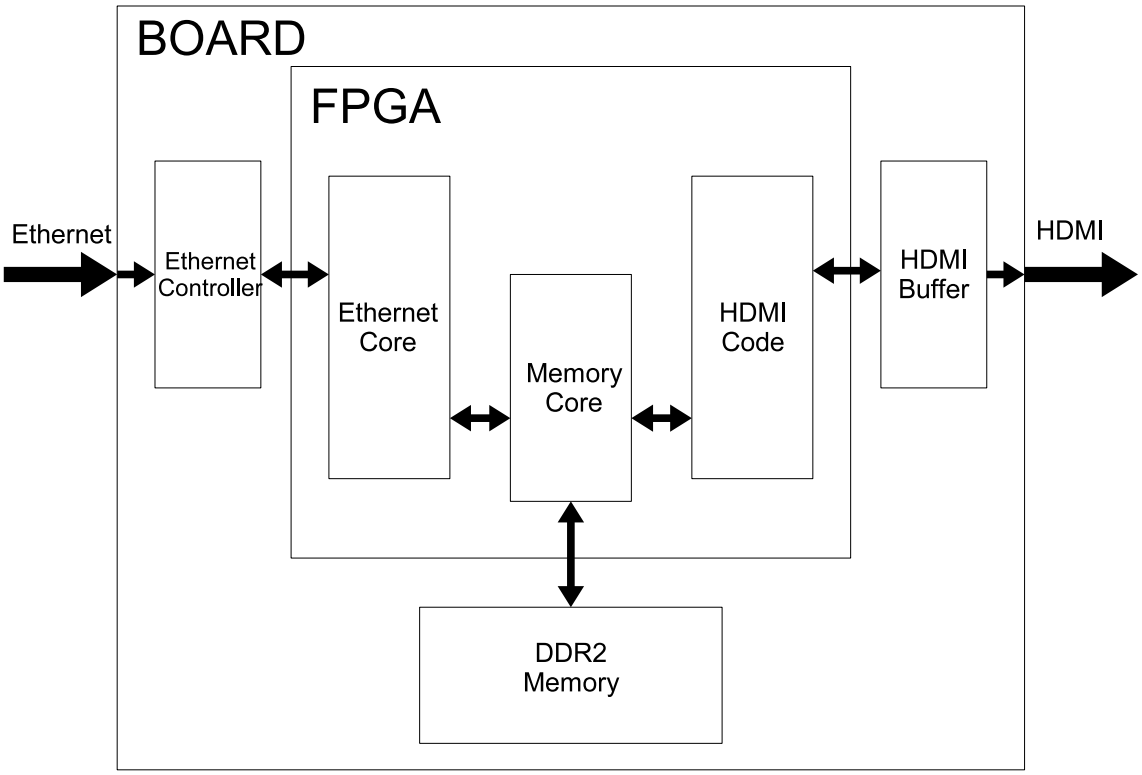


Figure 2.2: Block diagram over the entire system.

The gateway will be designed to transfer only video. As mentioned in chapter 1.1 HDMI can transfer both video and audio. This means that audio also could be included in the gateway but that is not implemented in this project.

2.2 Development Board

The development board that was used for the project was a board from Digilent called Atlys^{TM} Spartan-6 FPGA Development Board. The development board has the following features [6]:

- Xilinx Spartan-6 LX45 FPGA, 324-pin BGA (Ball Grid Array) package
- 128 Mbyte DDR2 16-bit wide data
- 10/100/1000 Ethernet PHY
- On-board USB2 (Universal Serial Bus) ports for programming & data transfer

- USB-UART (Universal Asynchronous Receiver / Transmitter) and USB-HID (Human Interface Device) port (for mouse/keyboard)
- Two HDMI video input ports & two HDMI output ports
- AC-97 Codec with line-in, line-out, mic & headphone connectors
- Real time power monitors on all power rails
- 16 Mbyte x 4 SPI (Serial Peripheral Interface Bus) Flash for configuration & data storage
- 100 MHz CMOS (Complementary Metal–Oxide–Semiconductor) oscillator
- 48 I/O's routed to expansion connectors
- GPIO (General Purpose Input / Output) includes 8 LEDs, (Light-Emitting Diode) 6 buttons & 8 slide switches
- Ships with a 20 W power supply and USB cable

The development board has both Ethernet and HDMI connections, so no extra connection was necessary to add to the board before the project could start. The Ethernet controller on the board is a Marvell Alaska Tri-mode PHY and is also called 88E1111 or just Ethernet PHY (physical layer) chip [7]. HDMI does not need to have any controller but it has a buffer circuit, and on this board it is a circuit called TI TMDS141 [8].

2.3 Development Environment

The FPGA that was used was a Spartan-6 from Xilinx, so a development environment from Xilinx was used. Xilinx provides a free development environment called ISE® WebPACK™ 13.2 (Integrated Software Environment). It is a fully featured front-to-back FPGA design solution. It offers HDL (Hardware Description Language) synthesis and simulation, implementation, device fitting, and JTAG (Joint Test Action Group) programming [9]. With this program it is easy to manage projects with VHDL (VHSIC (Very-High-Speed Integrated Circuits) Hardware Description Language) or Verilog code. Simulation of the code can be done with ISim. Synthesis is done with XST (Xilinx Synthesis Technology). Finally implementation is done which includes translating, maping and place & route. When the previous steps are done the hex code can be downloaded to the FPGA using iMPACT.

But ISE® WebPACK™ 13.2 has some limitations. One of them is that when the lines of code are more then 50 000 the simulation time is increased dramatically. So simulating large projects, will not work.

To speed up design time IP (Intellectual Property) cores have been used. Both cores from Xilinx and OpenCores have been used. Also example files with code from Xilinx have been used.

The project has been developed on a PC, HP ProBook 6550b, using Windows 7 Professional. The screen that has been used during the testing of sending HDMI signals is a Samsung SyncMaster BX2331.

Chapter 3

Theory Behind Ethernet

This chapter deals with the Ethernet and starts with a introduction to Ethernet followed by a short explanation of the Ethernet frame in general.

3.1 Introduction to Ethernet

Ethernet is used for communication, that is send and receive data between devices. It is the most widely used local area networking technology in the world today. Since 1973, when the first tests with Ethernet started, new and updated versions have been developed. The official Ethernet standard today is the IEEE (Institute of Electrical and Electronics Engineers) 802.3 which is controlled by the IEEE. So when referring to Ethernet it is the 802.3 standard you are referring to.

The Ethernet uses something called CSMA/CD (Carrier Sense Multiple Access with Collision Detection). To explain this in a short way, an illustration can be useful. Think of a group of people sitting in a dark room and can only hear but not see one another. Before you can speak it must be quiet for a period of time. This is what is called carrier sense. When it is quiet, all have the equal chance to say something. This is multiple access. Finally, if two people starts to speak at the same time they must stop talking and wait a random period of time before that try to talk again. This is collision detection [10] [11].

3.2 How is the Ethernet Frame Constructed?

Ethernet uses packets to send and receive data. A data packet is called a frame. In figure 3.1 an Ethernet frame is shown with the 802.3 standard.

A frame consist of some different parts. The first part is the preamble. It consist of 7 bytes with 1 and 0 alternative in this form “10101010”. The preamble is used for getting the involved circuits to become synchronized with each other. The second part is called SFD (Start Frame Delimiter)

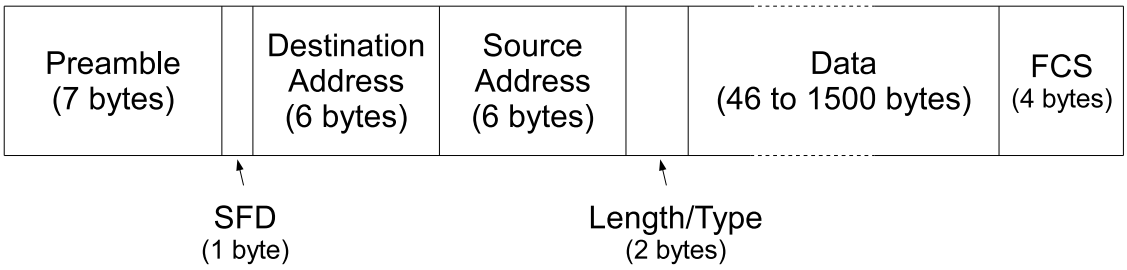


Figure 3.1: Show how the Ethernet frame is constructed.

which is only one byte long and has the form "10101011". The SFD indicates where the first bit of the frame is located. The SFD stops with two consecutive ones to indicates this.

Next are the address fields, with destination address and source address. The destination address is the address where the message is sent to. The source address is the address from which the massage is sent. Each one of these fields is 6 bytes.

Type or length field is the next field of the frame and it consist of 2 bytes. As the name indicates it can take two meanings. First if the value is less than or equal to 1 500 decimal (05DC in hexadecimal), then the it is considered to be a length field. In this case, it indicates how many bytes the data field consists of which is the next field. Secondly, if it is greater than or equal to 1 536 decimal (0600 in hexadecimal), then it is considered to be a type field. The type field was used in the original Ethernet to indicate which type the upper layer had, but now this field is mostly used for length, because type is indicated in a higher layer [12].

Next is the data field which can be between 46 bytes and 1 500 bytes long. If the data that shall be sent is shorter than 46 bytes zero padding shall be used. This means that extra bytes is filling the rest of the data field with bits of zeros so that the length gets to be at least 46 bytes. The extra bytes will be removed by the recipient when it receives the frame.

The last field is the FCS (Frame Check Sequence) which is 4 bytes long [10] [11]. This part is used for checking that the transfer of bits has been error free. The method that is used to verify that there are no errors in the frame is called CRC (Cyclic Redundancy Check).

3.2.1 How to Calculate the CRC

The calculation of the value that shall be placed in the FCS part will be explained. The destination address, source address, length/type and data are the basis for the CRC, and this part of the frame is called m. When calculating the CRC, polynomials are used to represent the binary numbers. So the frame is called M when using the polynomials. The transmitted frame, from destination address to and including the FCS, is called T and looks like in equation (3.1).

$$T = M \times x^{32} + R \tag{3.1}$$

Where R is the FCS, which in this case is 32 bits.

The main idea with CRC is that the reminder of $\frac{T}{G}$ shall be zero when no errors are detected, where the G is the pre-defined polynomial which in the 32 bits case looks like in equation (3.2). This polynomial is defined by the IEEE 802.3 standard.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (3.2)$$

So the R needs to be calculated before the frame is ready for transmission. R is calculated according to equation (3.3), but before starting the calculation the 32 first bits in the frame M must be complemented.

$$\frac{M \times x^{32}}{G} = Q + \frac{R}{G} \quad (3.3)$$

By taking the frame M and multiplying it with x^{32} and then dividing it with the pre-defined polynomial G, this will result in a quotient Q and a reminder R. This reminder R, which is the CRC, from this calculation is what should be placed in the FCS field. Before it is added to the frame, the 32 bits of the CRC are complemented. Now the frame is ready to be sent, according to equation (3.1).

When the frame is received, it needs to be checked for errors. So to check if there where any errors during transmission, the received frame is divided with G and the remainder is checked. If the reminder is zero then the frame that was received was error free. So $\frac{T}{G}$ shall have no reminder. Let's check this, equation (3.4).

$$\frac{T}{G} = \frac{M \times x^{32} + R}{G} = Q + \frac{R}{G} + \frac{R}{G} = Q + \frac{(R + R)}{G} \quad (3.4)$$

In modulo 2 when any binary number is added to itself it is zero, so there is no reminder, so the result looks like in equation (3.5).

$$\frac{T}{G} = Q \quad (3.5)$$

A short example of how to calculate the FCS will be given below. Until now only polynomials has been used. The cofficients of the polynomials represent the binary value, ether '1' or '0'. When the value is in binary form it is symbolized by letter case symbols, for example the frame that is notated M in polynomial is notated m in binary form. Example how the CRC is calculated will follow. This example have a frame that is 8 bit long and the pre-defined value g has 4 bits and the FCS has 3 bits. As mentioned before shall the 32 first bits of the frame be complimented before calculating the CRC, in this example no bits has been complimented. When the CRC has been calculated the 32 first bits shall be complimented before the CRC is added to the frame, this is not done in the example that fill follow.

- Frame that shall be sent, M: $x^7 + x^5 + x^4 + x^2$, m: "10110100"
- Pre-defined polynomial, G: $x^3 + x^2 + 1$, g: "1101"
- Number of bits in the FCS field: 3

- Starts by multiplying M with x^3, $M \times x^3 : x^{10} + x^8 + x^7 + x^6 + x^5$ and binary representation m is shifted 3 bits to the left and inserted zeros, m + "000": "10110100**000**"
- Then dividing the $M \times x^3$ with G using modulo-2 arithmetic [13] as shown in equation 3.3. In equation 3.6 the example frame:s CRC is calculated using polynomial division. In equation 3.7 the example frame:s CRC is calculated using binary division.

$$
\begin{array}{r|l}
 & x^7 + x^6 + x^3 + 1 \\
\hline
x^3 + x^2 + 1 & x^{10} + x^8 + x^7 + x^5 \\
 & -(x^{10} + x^9 + x^7) \\
\hline
 & x^9 + x^8 + x^5 \\
 & -(x^9 + x^8 + x^6) \\
\hline
 & x^6 + x^5 \\
 & -(x^6 + x^5 + x^3) \\
\hline
 & x^3 \\
 & -(x^3 + x^2 + 1) \\
\hline
 & x^2 + 1
\end{array}
\tag{3.6}
$$

$$
\begin{array}{r|l}
 & 10110100\mathbf{000} \\
\hline
1101 & \\
 & 1101 \\
\hline
 & 01100100\mathbf{000} \\
 & 1101 \\
\hline
 & 00001100\mathbf{000} \\
 & 1101 \\
\hline
 & 00000001\mathbf{000} \\
 & 1101 \\
\hline
 & 00000000\mathbf{101}
\end{array}
\tag{3.7}
$$

- The result of the division is summed up here: $\frac{x^{10}+x^8+x^7+x^5}{x^3+x^2+1} = (x^7 + x^6 + x^3 + 1) + \frac{x^2+1}{x^3+x^2+1}$
- The reminder is, R: $x^2 + 1$, r: "101". Which is the CRC.
- So the frame that will be sent looks like this, T: $x^{10} + x^8 + x^7 + x^5 + x^2 + 1$, t: "10110100101"

When the frame is received it needs to be checked. Example on how the check the received frame will follow.

- When the frame is received, it is checked by dividing it with G according to equation 3.5. In equation 3.8 the received example frame is divided using modulo-2 arithmetic in polynomials.

In equation 3.9 the received example frame is divided by using binary representation.

$$
\begin{array}{r|l}
x^3+x^2+1 & x^7+x^6+x^3+1 \\
\hline
 & x^{10}+x^8+x^7+x^5+x^2+1 \\
 & -(x^{10}+x^9+x^7) \\
\hline
 & x^9+x^8+x^5+x^2+1 \\
 & -(x^9+x^8+x^6) \\
\hline
 & x^6+x^5+x^2+1 \\
 & -(x^6+x^5+x^3) \\
\hline
 & x^3+x^2+1 \\
 & -(x^3+x^2+x^1) \\
\hline
 & 0
\end{array}
\tag{3.8}
$$

$$
\begin{array}{r|l}
1101 & \overline{10110100\mathbf{101}} \\
\hline
 & 1101 \\
\hline
 & 01100100\mathbf{101} \\
 & \;1101 \\
\hline
 & 00001100\mathbf{101} \\
 & \quad 1101 \\
\hline
 & 00000001\mathbf{101} \\
 & \qquad 1101 \\
\hline
 & 00000000\mathbf{000}
\end{array}
\tag{3.9}
$$

- The result of the division is summed up here: $\frac{x^{10}+x^8+x^7+x^5+x^2+1}{x^3+x^2+1} = (x^7+x^6+x^3+1)$
- Since there is no reminder term, no errors was found in the transmission.

Each byte of the frame is transmitted with the LSB (Least Significant Bit) first, except the FCS field. The FCS is transmitted with MSB (Most Significant Bit) bit of the first byte first and the LSB bit of the last byte as the last bit [10].[1]

[1] In some special cases an extra field is added last in the frame, called extension field, if you want more information on this read source [11].

Chapter 4

Theory Behind HDMI

The chapter deals with HDMI. It starts with a short introduction to HDMI followed by an explanation of the cable that is used when transferring video data. Then TMDS (Transition-Minimized Differential Signaling) will be described.

4.1 Introduction to HDMI

HDMI is a interface for transferring video and audio data between two devices. It is commonly used in home settings for connecting for example a DVD player to a TV-set. It are also used in offices to connect for example a computer to a monitor.

To minimize the electromagnetic interference a thing called TMDS will be used. It is trying to minimize the number of transitions in the cable. This will be described more in detailed later in the chapter.

In 2003 HDMI started to be used. According to Steve Venuti, president of HDMI Licensing, LLC, 2 billion HDMI-enable products has been shipped since the launch in 2003 [15].

In this project the HDMI Specification 1.3a has been used. There is a newer version, but to be able to get the documentation for that one it is necessary to become a HDMI Adopter. To become a HDMI Adopter is quite expensive.

Four different colour depths are supported by HDMI, which are 24-, 30-, 36- and 48-bits per pixel. In this project 24 bits per pixel was used, which means the three colours red, green and blue are represented by 8 bits each. With this colour depth 16 777 216 different colours can be generated. This is called truecolor.

An nice feature with HDMI is that DVI uses the same protocol as the HDMI uses for video. So a DVI-to-HDMI or HDMI-to-DVI adapter will not need any signal conversion and as a result of that, no loss in video quality will appear. However, DVI is not capable of transferring audio.

4.2 HDMI Cable

When sending the data one wants to minimize the electromagnetic interference in the copper cables. In order to do this the number of transitions of the voltage levels must be minimized, i.e. minimize the transitions from 0 to 1 and 1 to 0. To be able to do this TMDS is used. In figure 4.1 a block diagram over the HDMI is shown.

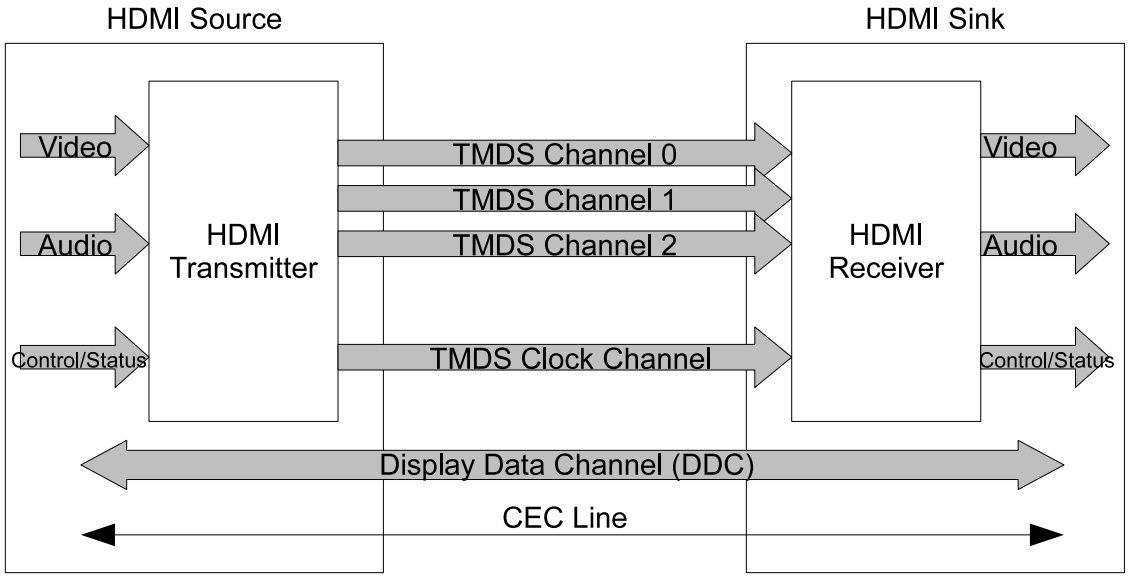


Figure 4.1: Block diagram over HDMI [2].

On the left-hand side is the HDMI source, which generate signals. The video and audio data are fed to the HDMI transmitter, which process the signals and converts them to TMDS. Some control and status signals are fed to the HDMI transmitter as well. The HDMI transmitter encode the signals so that can be sent on the HDMI cable.

In the middle is the cable which consists of 19 wires. Each of the three channels and the clock channel has 3 wires. One is for shield and the other two are for data. The two wires for data are inverted to each other. The same applies to the clock signal. In figure 4.2 a time-diagram are showing how it can look when sending data over HDMI using TMDS. Two wires are used for DDC (Display Data Channel) which uses I^2C (Inter-Integrated Circuit) bus to let the devices communicate, for example learn what format on the audio and video that are supported by the screen. A CEC (Consumer Electronics Control) wire is used for high-level control functions between all of the various audiovisual products in a user's environment. Four signals are not shown in the figure, but they will be described here. One wire is used for determine when a monitor is connected or not, which is the HPD (Hot Plug Detect) signal. The remaining three wires are used for +5 volt, ground and one is reserved. On the Atlys development board that was used, the CEC and HPD signals where not connected to the FPGA.

On the right-hand side in figure 4.1 the HDMI sink is shown. The HDMI receiver decode the signals from the HDMI cable and divides it up in video and audio data and control signals [2].

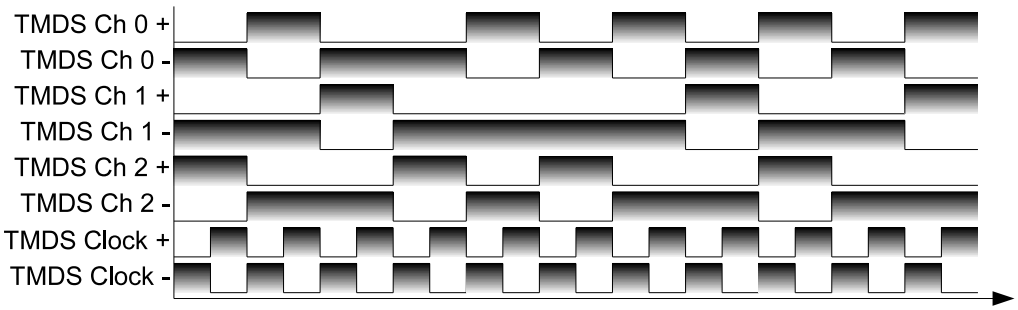


Figure 4.2: Time-diagram for data transmission over TMDS.

4.3 What Is TMDS And How Is It Used?

TMDS stands for Transition Minimized Differential Signaling as mentioned above. It uses a coding algorithm that is minimizing the transitions in the cable. Each channel has two wires for data transmission which are inverted to each other. Through the three channels video, audio and auxiliary data are transmitted in different time periods.

The data is transmitted on three TMDS channels as mentioned before. When video data is sent the pixel is divided in three parts that is sent in the three channels. All three components that builds up a pixel are sent in parallel. Each pixel is represented by 24 bits.

To minimize the number of transitions an advanced algorithm is used that convert 8 bits to 10 bits with less transitions. With 10 bits it is possible to have 1 024 different combinations. To represent the 256 different colours, 460 combinations are used. A colour can be represented by at most two combinations and some colours has only one combination. Four combinations are used for control signals. The rest of the combinations are reserved and forbidden, that is 560 [16].

4.3.1 How Does It Work With the Different Periods

Three different modes are used when transmitting data over the TMDS channels. These are Video Data Period, Data Island Period and Control Period. The actual pixel data are sent in the Video Data Period. Audio and auxiliary data are sent in the Data Island Period. When no video, audio or auxiliary data are sent the Control Period is active. There must be a Control Period between two other periods that are not Control Periods. In figure 4.3 is an example of how the different periods can be placed. This example uses a screen resolution of $1\,280 \times 720$.

The Control Period is used for synchronization or contains a preamble which indicates what type of data that will follow, ether Video Data Period or Data Island Period. Each Video Data Period starts with a pre-defined Video Leading Guard Band and this is then followed by the actual data. The Data Island Period starts with a pre-defined Leading Guard band which is then followed by audio or auxiliary data. The auxiliary data include data that describes the source [2]. The two signals HSYNC and VSYNC that are shown in figure 4.3 are signals that are sent in the Control Period. The VSYNC indicates that a new image frame has started and the HSYNC indicates that a new line has started.

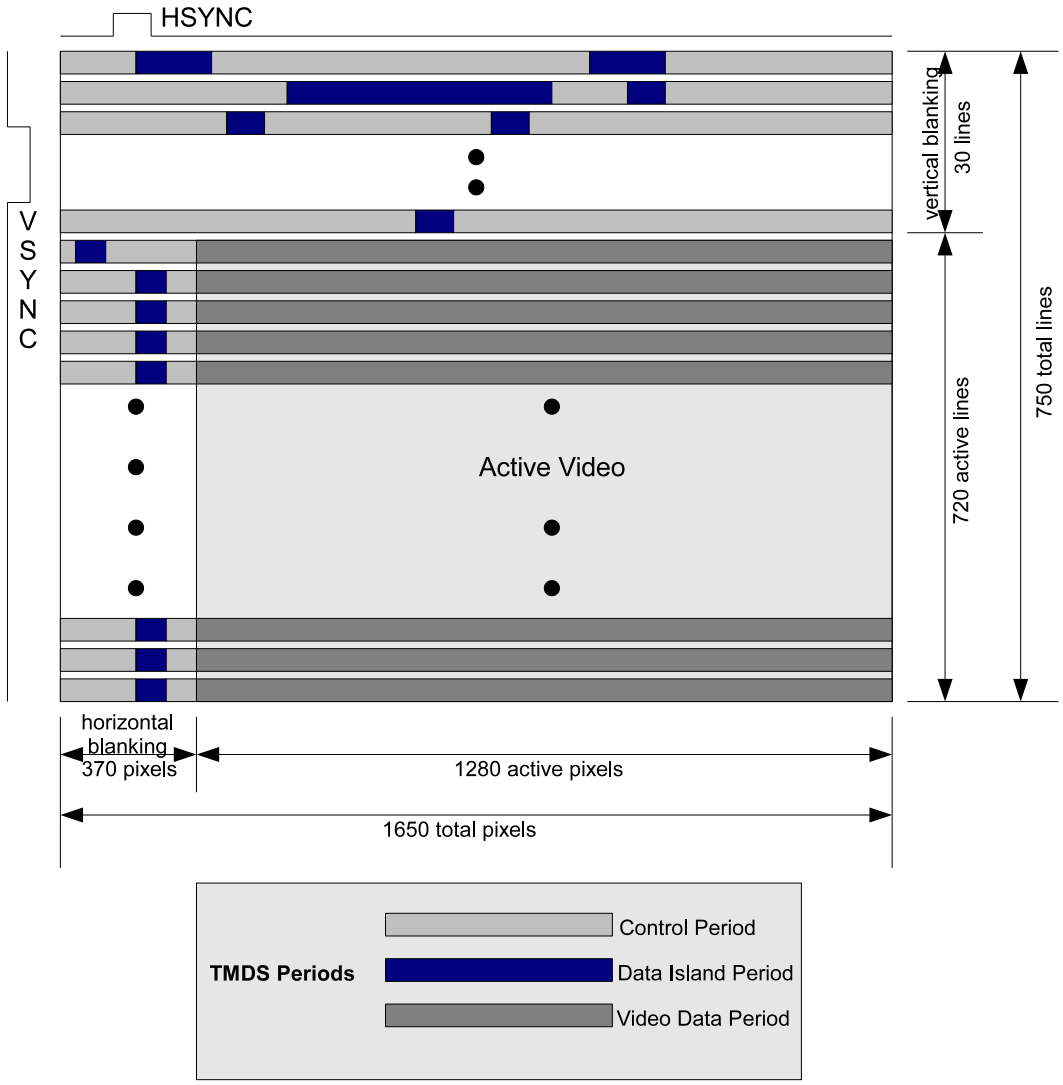


Figure 4.3: Three different periods are used when transmitting TMDS [2].

Chapter 5

Design of Ethernet

This chapter will deal with the different design aspects of Ethernet and the VHDL code that has been designed for the Ethernet.

5.1 Ethernet Hardware And Software

As mentioned earlier there is a Ethernet controller available on the board. In figure 5.1 it is shown how the Ethernet controller is connected to the FPGA [6].

To be able to communicate with the Ethernet controller some kind of Ethernet core is necessary. There exist some free cores, for example from OpenCores and Xilinx. For this project the OpenCores Ethernet core was used.

The core from OpenCores is written in Verilog and uses the Wishbone bus protocol to communicate with a top module.[1] The documentation for the core was deficient and it was difficult to understand how the core was working.

Xilinx also has an Ethernet core but it is mostly used for communication with a soft processor on the FPGA. Also some special programs are necessary to do this, because of that this core from Xilinx was not used.

5.1.1 Ethernet Core to Ethernet Controller

Communication between the Ethernet core and the Ethernet controller are done using a number of signals, that can be seen in figure 5.1.[2] All of these signals are not used in this project.

This protocol is called MII (Media Independent Interface) and can be divided into two parts, the data interface and the management interface. The data interface is used for receiving and

[1] The core is called Ethernet MAC 10/100 Mbps and the head author is Igor Mohor.

[2] Since only a shorted version of the datasheet for the Ethernet controller was available[7], the assumption that the signals behave as they are described in the IEEE 802.3 standard where made [11]. The complete datasheet from Marvell for the Ethernet controller is under non-disclosure agreement according to Atlys manual [6].

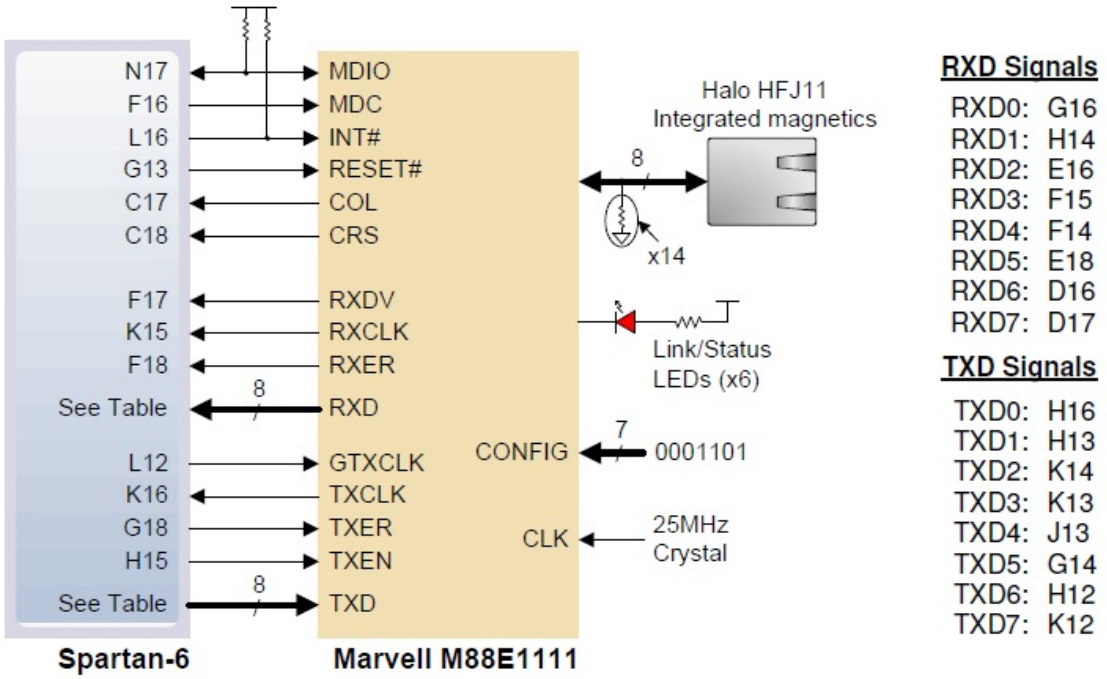


Figure 5.1: The FPGA is connected to the Ethernet controller by a number of wires [6].

transmitting Ethernet frames. The management interface is used for setting and reading control and status registers on the PHY. MII can handle two data rates, 10 Mb/s and 100 Mb/s.

The management interface uses two wires, MDIO (Management Data Input/Output) and MDC (Management Data Clock). The MDC is giving the PHY a timing reference for the MDIO signal. The data are sent on the MDIO signal which is bidirectional.

The rest of the signals belong to the data interface except INT# and RESET#. RESET# is just for reseting the PHY. INT# is for some interrupt signal that can trigger the PHY.

To be able to handle the CSMA/CD that is mentioned in chapter 3.1 two signals are used for this, COL (Collision Detected) and CRS (Carrier Sense). These signal are only used in half-duplex mode. The half-duplex mode means that the frames are sent on the same physical cable in both directions, but at the different times. In full-duplex mode different physical cables are used for transmission and receiving. In this case it is not needed to use this two signals. But in half-duplex mode they are needed. CRS is active when ether transmission or receiving is in process. The COL is active when a collision is detected. That is when both reception and transmission are processed at the same time.

When frames are received, three signals and one data bus is used for transferring the data to the FPGA. The RXDV (Receive Data Valid) signal is indicating when data is valid on the data bus. RXCLK (Receive Clock) is to synchronize the data transfer. If an error occurs during receiving the RXER (Receive Error) signal will indicate this. Finally RXD (Receive Data) is an four bit wide data bus that transfers the actual frame data.

Transmitting frames is working almost the same way as receiving frames when considering the signals between the FPGA and PHY. TXEN (Transmit Enable) indicates when data is enabled on the bus. When an error occur it is indicated by TXER (Transmit Error). To get a time reference TXCLK (Transmit Clock) is used. Finally the data are sent on TXD (Transmit Data) and as for receiving only four bits are used.

As can be seen in figure 5.1 an additional signals is marked, which is GTXCLK (1 000 Mb/s Transmit Clock). Also the TXD and RXD buses are eight bits wide. These extra signals are used when another interface called GMII (Gigabit Media Independent Interface) is used which only can handle a data rate of 1 000 Mb/s [11].

The PHY has eight pins connected to a Halo HFJ11 connector. That is where the Ethernet frames are being transmitted on the Cat5 Ethernet cable. Six LED:s are connected to the PHY to get some feedback on what is being processed at the moment. The LED:s indicate the data rate that is used and which duplex mode that is used and also if transmission or reception is in process. There also are seven pins that are used for setting some special parameters in the PHY. These can not be changed, since that are soldered on the board. Finally the PHY is provided with a 25 MHz clock.

5.1.2 Ethernet Test

To test that the Ethernet core was sending frames, a program called Wireshark was used [17]. With this program it is possibly to log Ethernet frames that are sent over the network. It shows the MAC (Media Access Control) destination address, MAC source address and the format of the frame, for example TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). It also shows at what time it was sent and all the data, among other things. In figure 5.2 a printed screen from Wireshark is shown.

To check that the FPGA was really sending some frames it was checked by Wireshark in the following way. By letting the FPGA transmit Ethernet frames that had a MAC address and MAC destination and had a UDP header and a IP header, IP will be explained later in the report, and some data that was pre-defined. Then logging the Ethernet traffic on the network and finding the frame that was sent from the FPGA, in this way it was possible to verify that the frame was sent without any errors.

5.1.3 Ethernet Frame with Image Data

The Ethernet frame is the basis of the communication. The data in each frame is dependent on what it is used for. In figure 5.3 a structure with several layers is shown. This communication protocol is called Internet Protocol Suite. The lowest layer is the link layer. This layer includes the frame header and the frame footer as shown in figure 3.1. The layer above that is the internet layer which in this case is IP. The transport layer includes the UDP. The highest layer is the application layer which includes the data [10].

In this project the Ethernet frame is defined according to the figure 5.3, but each layer can use one of many different formats. For example in transport layer you can use TCP or DCCP (Datagram Congestion Control Protocol). In the highest layer, the application layer, different

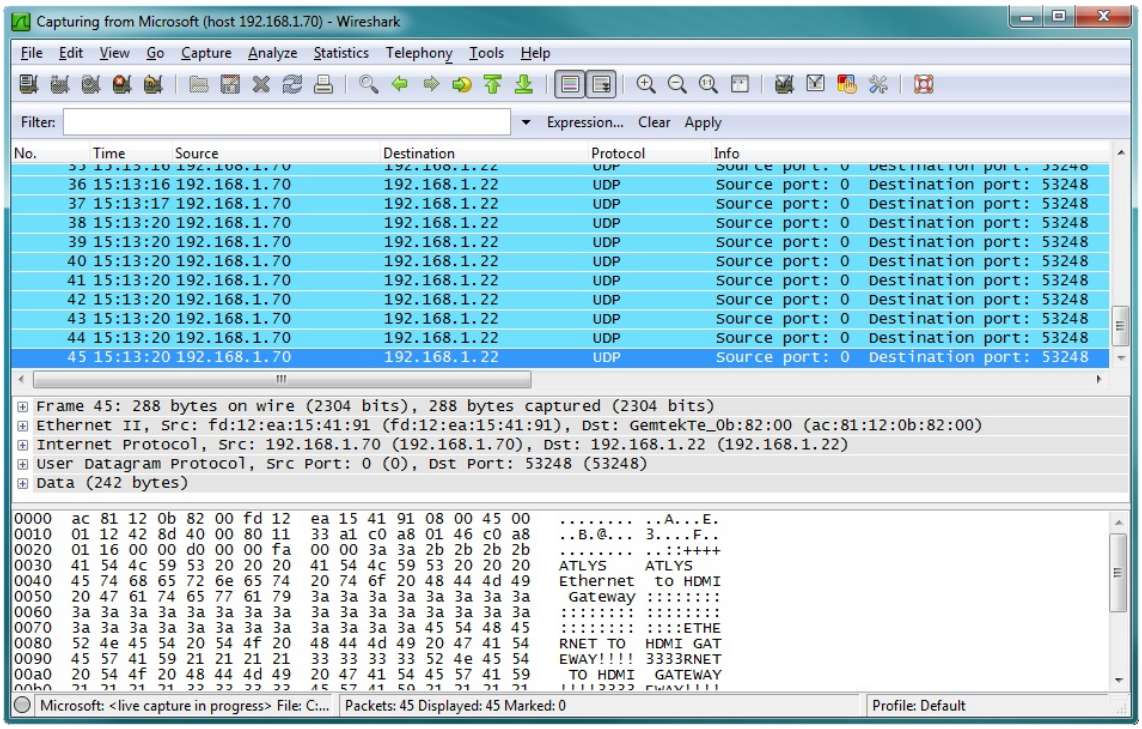


Figure 5.2: Shows the main window of WireShark.

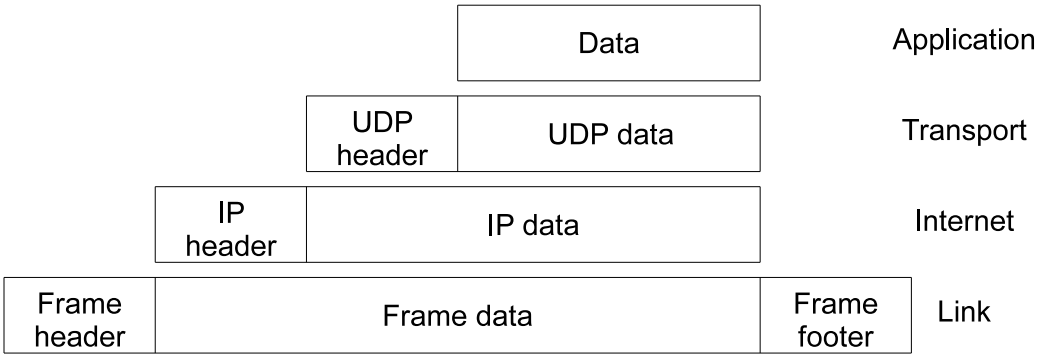


Figure 5.3: Shows the different layers in the Internet Protocol Suite and what they include in this project.

protocols can be used, for example HTTP (Hypertext Transfer Protocol) or FTP (File Transfer Protocol). In the internet layer other formats than IP can be placed.

In this project Ethernet frames with video data shall be revived. It would be good to use IP in the internet layer and UDP in the transport layer to send the data. Since UDP does not require any acknowledgements from the receiver it does not need to be so complex as for example TCP that needs acknowledgements and other functions to make it work.

The UDP frame has four fields as a header. The first field is for source port followed by destination port. This addresses some specific function that can be used. But in this project it does not matter since this is a special case. Each of these fields are two bytes long. The next field is the length, which indicates the length of the both the UDP header and its data. The next field is the checksum that is calculated both om the UDP header and its data. The length and the checksum is both two bytes long each. The last field is where the data is placed [14].

To be able to use UDP it needs to be placed in the IP:s data field. There exist different versions of IP, the newer version is called IPv6 but in this project IPv4 is used. The IP frame consists of a IP header and IP data. The header consists of 20 bytes where 8 are used for source and destination address. The rest of the field is for setting parameters like version, header checksum and length among other things.

The raw data shall be placed in the application layer. So the video data will be placed in the application layer. To know where the data that are being sent should be placed on the screen, some additional data needs to be added. This is because a image frame cannot be sent in one Ethernet frame, it needs to be divided into several frames.

According to figure 3.1 the data field in the MAC frame was 46 to 1 500 bytes. But now the IP header and the UDP header have taken some space in the data field. The IP header consists of 20 bytes and the UDP header consists of 6 bytes. This means that the data field in the application layer can only handle 1 474 bytes maximum.

The Ethernet frames were constructed according to figure 5.4. Where the data field is 242 bytes long. Two bytes are used for indication where the pixels are going to be placed on the screen. The other 240 bytes are used for sending 80 pixels. Each pixel consists of 3 bytes, so $3 \times 80 = 240$ bytes. How the screen is divided will be explained in chapter 6.2.

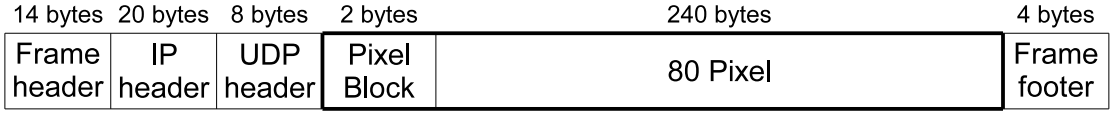


Figure 5.4: Shows how the frame that contains the image data looks like.

How the data in the application layer was structured was not according to any standard. It was a design decision made in the project. If more pixels where sent on each Ethernet frame the efficiency of the transmission on the Ethernet would be higher. But on the other hand. it will be difficult to handle to large amount of data in the FPGA since it has a limited size.

The total length of the Ethernet frame is 288 bytes. The first 42 bytes are the same as in figure 5.3, which are the Frame, IP and UDP header. The two following bytes are used to identify which block of pixels the image data belongs to. The image data are placed in the following bytes. Last in the frame is the Frame footer which is the FCS.

Between every frame there needs to pass some time, this is called IPG (Inter Packet Gap). For 100 Mb/s it is 96 bits that are needed between frames. Which means 0.96 μs between two frames.

5.2 Design of Ethernet - VHDL Code

The sending and receiving of the actual Ethernet frames is done by the on-board Ethernet controller as mentioned before. To be able to communicate with the controller is quite difficult, so a pre-designed core was used. To use a core also speeds up design time. A core from OpenCores was chosen. It was a free core which had a relatively easy protocol to communicate with it, which was Wishbone.

To be able to communicate with the Ethernet core via the Wishbone[3] protocol a FSM (Finite State Machine) was used. The FSM was written in VHDL and contains 103 states. It contains an initialization phase for the Ethernet core. Then it waits for the user to set some switches or push some button. To get it to receive Ethernet frames continuously a switch must be set. Then it will go to a state were it waits until it captures anything on the network. In figure 5.5 a flowchart over the FSM for the Ethernet module is shown.

Every time a frame has been received it is indicated by a led on the board. The frame is checked for its MAC destination address to see if it is sent to this device, and then it also checked if it contains pixel data. If it has the correct MAC destination address the FSM machine will indicate this for a short moment, then it will return to the wait state and wait for the next frame to receive. This is the function of the Ethernet FSM.

[3]More information about the Wishbone protocol can be found in the source [21].

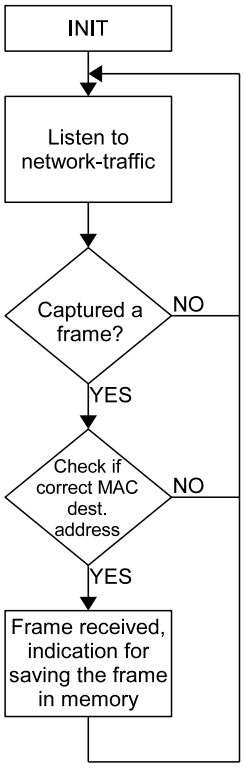


Figure 5.5: Flowchart Ethernet.

Chapter 6

Design of HDMI

In this chapter will deal with the design of the different aspects of HDMI and the VHDL code that has been designed for the HDMI.

6.1 HDMI Hardware

The HDMI does not have any driver which the Ethernet has. Because of this the signals can be generated directly from the FPGA. But there is a buffer circuit between the FPGA and the cable. That circuit makes it possible to send HDMI data up to 1 m, because the cable introduces both distortion on the TMDS channels and increasing capacitance on the DDC channel. The TMDS141 include both differential drivers and differential receivers. So the same circuit can be used both for transmitting and receiving HDMI video and audio data [8].

In figure 6.1 it is shows how the FPGA is connected to the TMDS141 and further to the HDMI connector which is of type A.

The HDMI cable was connected to the J2 connector on the board which is a HDMI output. As can be seen from figure 6.1 only 10 wires are connected between the FPGA:n and the HDMI cable. The TX bus which has 8 bits is used for three data channels and one clock channel. Two wires are used for the DDC channel which is the SCL (Serial Clock) and SDA (Serial Data Signal), but these wires are not used in this project.

To check that the HDMI signals are sent in a correct way, a screen was connected to the other end of the HDMI cable. To verify that the output was as expected a visual inspection of the image on the screen was made.

6.2 Screen

In this project two different resolution has been used, which are the following:

- HDTV 720p (1 280 × 720):

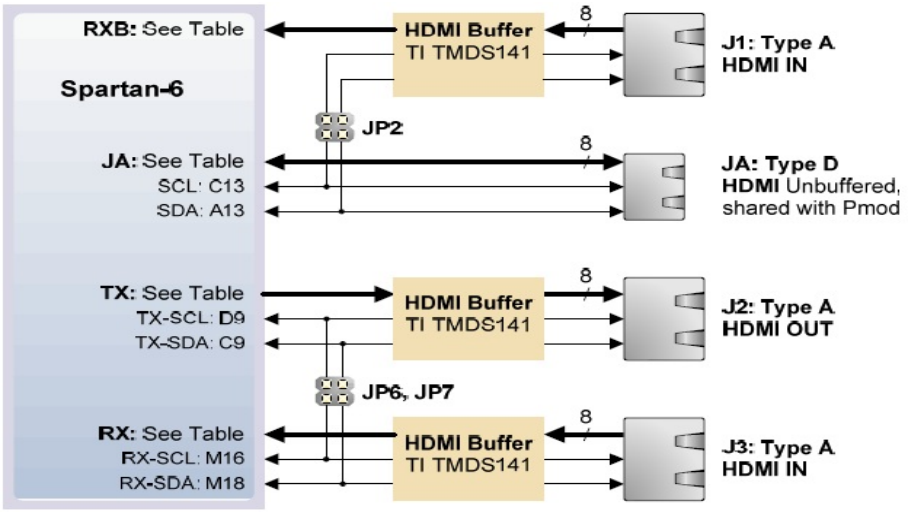


Figure 6.1: The FPGA:n is connected to the HDMI connector via a HDMI buffer circuit [6].

  - Number of video pixels: 921 600
  - Refresh Rate: 60 Hz
  - Pixel Rate: 74.25 MHz
  - Serial Data Rate: 742.5 Mb/s
- SVGA (800×600):
  - Number of video pixels: 480 000
  - Refresh Rate: 60 Hz
  - Pixel Rate: 40 MHz
  - Serial Data Rate: 400 Mb/s

The larger of the two is the HDTV 720p (High-Definition Television) which is becoming more and more used today. The smaller resolution is the SVGA (Super Video Graphics Array). It is not an official standard resolution but it is used in this project to be able to compare different things to the larger resolution.

From figure 4.3 it is clear that when sending video data for the HDTV 720p it needs more than $1\,280 \times 720$ pixels, it needs $1\,650 \times 750 = 1\,237\,500$ pixels. Since the screen shall be refreshed 60 times each second a pixel rate of $1\,650 \times 750 \times 60 = 74{,}25$ MHz in required. As mentioned before colour depth of 24 bits is being used which means that each colour is represented by eight bits. These are sent in parallel and are converted to 10 bits each, so to send a pixel takes 10 bits in length. So the serial data rate is $1\,650 \times 750 \times 60 \times 10 = 742{,}5$ Mb/s [18].

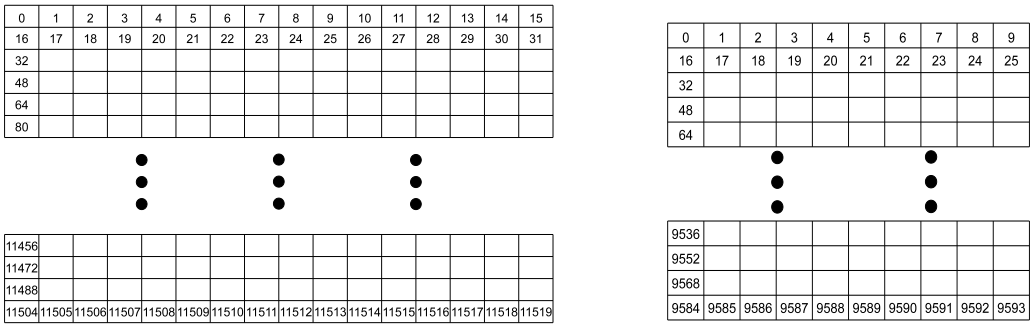
Since both resolutions has a refresh rate of 60 Hz it takes $\frac{1}{60} = 16.7$ ms to send a image frame to the screen regardless of resolutions, but since the larger resolution contains more data it needs to

sent more data for each image frame and therefore it uses a faster pixel clock. As mention earlier the pixel clock for the HDTV 720p is 74.25 MHz and for the SVGA it is 40 MHz.

6.2.1 The Screen Is Divided Into Smaller Blocks

The screen is divided into smaller regions so that data from the Ethernet frame can be placed at the correct place on the screen. Each Ethernet frame contains 80 pixels so a good way is to divide the screen into a blocks of 80 pixels.

In figure 6.2 it is shown how the screen is divided fo the two resolutions. The smaller screen resolution has the same index as the larger one, the only difference is that the six columns that a located to the right are neglected.



(a) 1 280 × 720 resolution.

(b) 800 × 600 resolution.

Figure 6.2: Index of the different field of the screen.

The screen is divided into 16 columns which results in $720 \times 16 = 11\,520$ blocks for the higher resolution. When the lower resolution is used there are only 10 columns which results in $600 \times 10 = 6000$ blocks. But as noted earlier, indexes for the lower resolution are not continuous. It is skipping six positions at a regular pattern. So the index stops on 9593 and not what as would be expected on 5999.

Whichever of the lower or higher resolution, each block consists of 80 pixels. This is what is sent in each Ethernet frame, so this is very convenient.

6.2.2 Displaying Images On the Screen

When playing a video it is often 24 image frames each second that are displayed. To be able to accomplish this it is necessary to send $24 \times 16 \times 720 = 276\,480$ Ethernet frames each second or $24 \times 10 \times 600 = 144\,000$ Ethernet frames for the lower resolution. Will this be possible?

Each Ethernet frame is 288 bytes long which is 2304 bits. With a data rate of 100 Mb/s[1] it will

[1]There exists some different definitions of what 1 Mb (Megabit) represent in bits. In this text it will follow this conversion 1 Mb = 1 000 kb = 1 000 000 bits, for simplicity. Another way to calculate this is using this conversion 1 Mb = 1 024 kb = 1 048 576 bits.

take 23.04 μs to transmit a Ethernet frame. The time between two frames is 0.96 μs which means that a block of 80 pixels can be delivered to the FPGA every 23.04 + 0.96 = 24 μs. This results in $\frac{10^6\,\mu s}{24.0\,\mu s} \approx 41\,666$ Ethernet frames can be sent each second. The numbers of image frames that can be displayed each second for each resolution is given below:

- $1\,280 \times 720$: $\frac{41\,666}{11\,520} \approx 3.62$ image frames per second
- 800×600 : $\frac{41\,666}{6\,000} \approx 6.94$ image frames per second

From this result it is obvious that it will not be possible to display 24 image frames each second but rather around 3 to 7 frames per second depending the resolution. If the data rate for the Ethernet is increased to 1 000 Mb/s it would be possible to send image data at the required frame rate. Assuming that the IPG is the same and it take the same time to send the Ethernet frame it will be possible to send $\frac{10^7\,\mu s}{24.0\,\mu s} \approx 416\,666$ Ethernet frames per second. Which results in $\frac{416666}{11520} \approx 36.2$ image frames each second for the 1280×720 resolution.

6.2.3 Display Methods

To display the pixels on the screen, different methods can be used. One of these is interlaced video and the other is progressive scan. When using interlaced video the image frame is displayed by first paint the odd lines and then the even lines. So the screen is generated in two steps, first the odd lines and then the even lines. When progressive scan is used the screen is generated in one step. Line by line is displayed for the entire image frame. In this project progressive scan has been used and that is most commonly used in newer equipment.

6.3 Design of HDMI - VHDL Code

The sending of HDMI video data has been done using example files from Xilinx that were written in Verilog. These files just send out a simple test image on the cable to the screen. They where originally designed for DVI but works equally well for HDMI.

The code has been modified to fit the task for this project. Some parts has been rewritten to VHDL, so they can easily be understood. There is a top module that compiles several subcomponents that handle different tasks.

The component that handles the image generations has been completely rewritten. Now it is a FSM that have the task to collect and generate the pixel data at an exact time.

As mention before is the screen divided into blocks of 80 pixels. A counter is used to indicate where on the screen that is updating right now. For example when the screen is updating the pixels on the first row and the first 80 pixels is the counter 0 and that is the same as the index for the screen. When pixels 81 to 160 on the first row is updated, is the counter 1 and so on. This counter is called block counter.

The FSM is written in VHDL and consists of 7 states. In figure 6.3 the flowchart for the HDMI module is shown. It starts by requesting pixel data from the memory. When it gets the pixel data from the memory it will wait for the right place on the screen to place the data. Since the screen is updating the screen pixel by pixel, it is necessary to wait for the screen to update the block of

pixels that just has been collected from the memory. When it is the right time to place the data it is sent out to be transformed to a TMDS signal. Before it leaves this state the current position of the pixel block counter is stored in a register. In the next state the pixel block counter will be increased by one and a request is sent to the memory for the next pixel data. Then it waits again for the data from the memory. This is how the FSM machine is working for collecting and generating pixels to the screen.

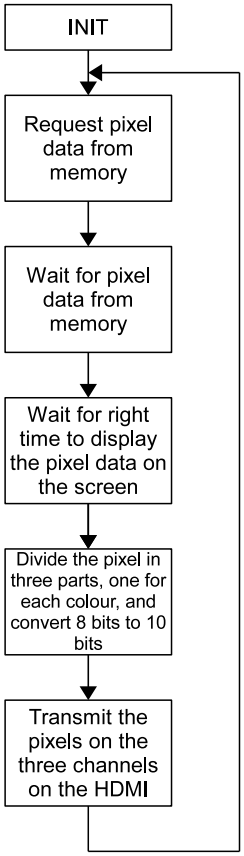


Figure 6.3: Flowchart HDMI.

Chapter 7

Design of Memory

This chapter will start with a short introduction to memory. Then will this chapter deal with the design of the different aspects of Memory and the VHDL code that has been designed for the Memory. The chapter ends with describing how the different parts are connected to each other.

7.1 Introduction to Memory

Memory is used for storing data. This is necessary in this case where a lot of data needs to be stored. The on-board DDR2 SDRAM memory will be used for this purpose.

In this project the DDR2 SDRAM memory is used for storage of pixels that build up an image frame. Since the memory can store 128 MByte it can without problem store several image frames. But only one image frame is saved.

7.2 Memory Hardware

In figure 7.1 the connection between the FPGA and the memory is shown. There are a lot of signals that are needed but the two most important ones are the address input to the memory and the data which is a bidirectional port[1].

To communicate with the memory a core has been used [19]. The core is from Xilinx and was generated with the help of a wizard[2].

In figure 7.2 it is shown how the pixels are arranged in the memory. It does not matter if the resolution is high or low since they have the same index. Each row in the memory have 64 bits or 8 bytes. This means that $2\frac{2}{3}$ of a pixel can be stored in a row. In three rows eight pixels can be

[1] For an explanation of the signals om the DDR2 memory, read the source [20].

[2] Some guidance from Joel William's homepage `http://tristesse.org/` has been helpful when generating the core and also communicate with it.

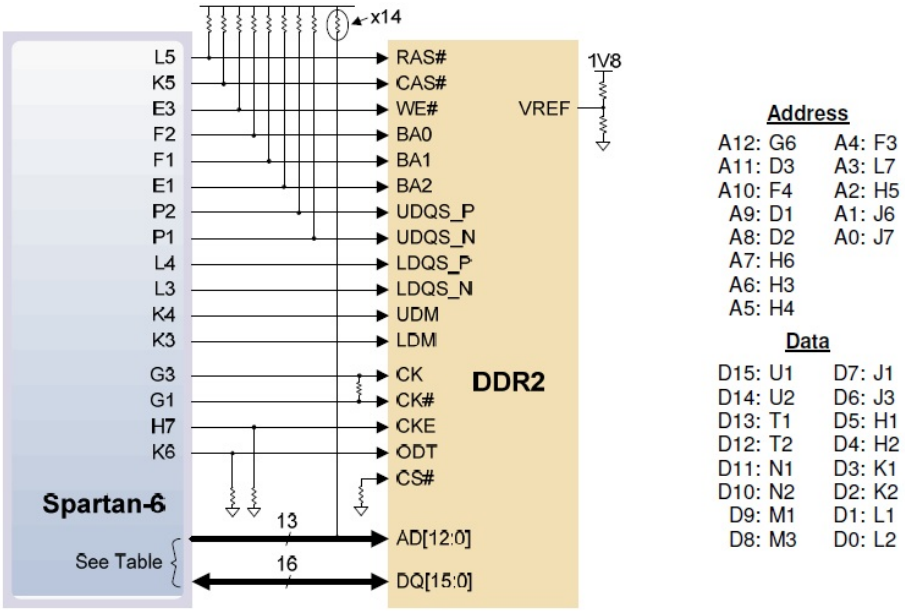


Figure 7.1: The FPGA is connected to the DDR2 memory [6].

stored. To storage 80 pixels requires $\frac{80}{2\frac{2}{3}} = 30$ rows. So when storing entire Ethernet frames with pixels to the memory it will be in every 30th row.

The memory was initially tested by saving random data from frames that where captured on the network to the memory. Then reading from the memory and putting the data into a frame that was send on the network. Checking was done by inspecting the frame with Wireshark.

7.3 Design of VHDL Code That Handle the Memory

Since an entire image frame can not be stored in a register in the FPGA, it is stored in a memory instead. To communicate with the memory, a core was used to speed up the design time and to be able to use it in an easier way.

To control the core a FSM was used. The FSM was written in VHDL and consists of 13 state. A flowchart of the FSM for the memory module can be seen in figure 7.3. The FSM starts with a initialization state where it waits for the clock to be synchronized. Then it waits for a read or write signal. When a read signal is activated, it starts to send a read request to the core with the pixel block counter as an address in the memory. When the core has collected the data from the memory it will send it out so it can be collected by a register in the FSM. A signal is indicating that data has been read from the memory and is available for the HDMI FSM to catch for a brief period of time. Then it returns to the wait state and waits for new commands.

When a write signal is activated together with data and address on the inputs to the FSM it starts to request a write to the core. Then it loads the data in to the core, and also the address in

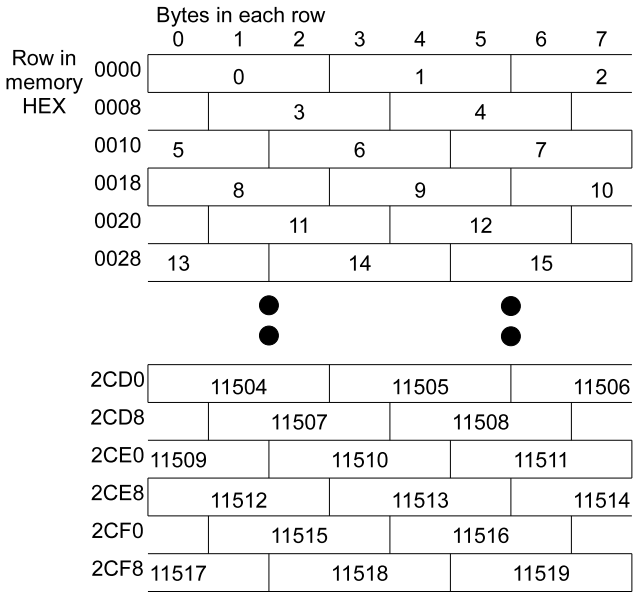


Figure 7.2: Shows how the pixles are arranged in the memory.

which the data shall be placed. When all the data has been written to the memory it returns to the waits state and waits for new commands.

7.4 VHDL Code That Connects All Different Parts

To connect the three components, a top module written in VHDL was designed that made the appropriate connections between them. Also connections to the external pins of the FPGA was made in this code. In figure 7.4 is the top module and its connections shown, both internal signals and signals that are connected externally. The clock distribution was managed from this code as well. In figure 7.5 the code structure is shown. At the top is the code that combines all the different parts. Underneath are the FSM and cores shown for the different parts.

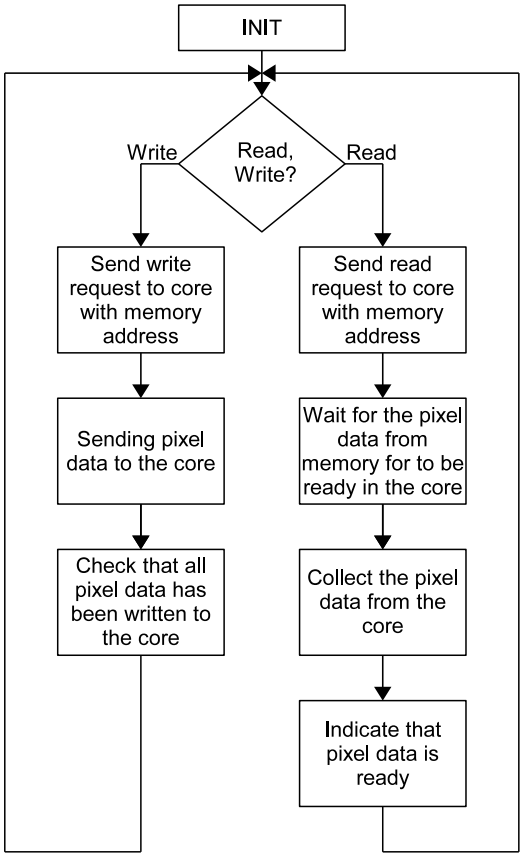


Figure 7.3: Flowchart Memory.

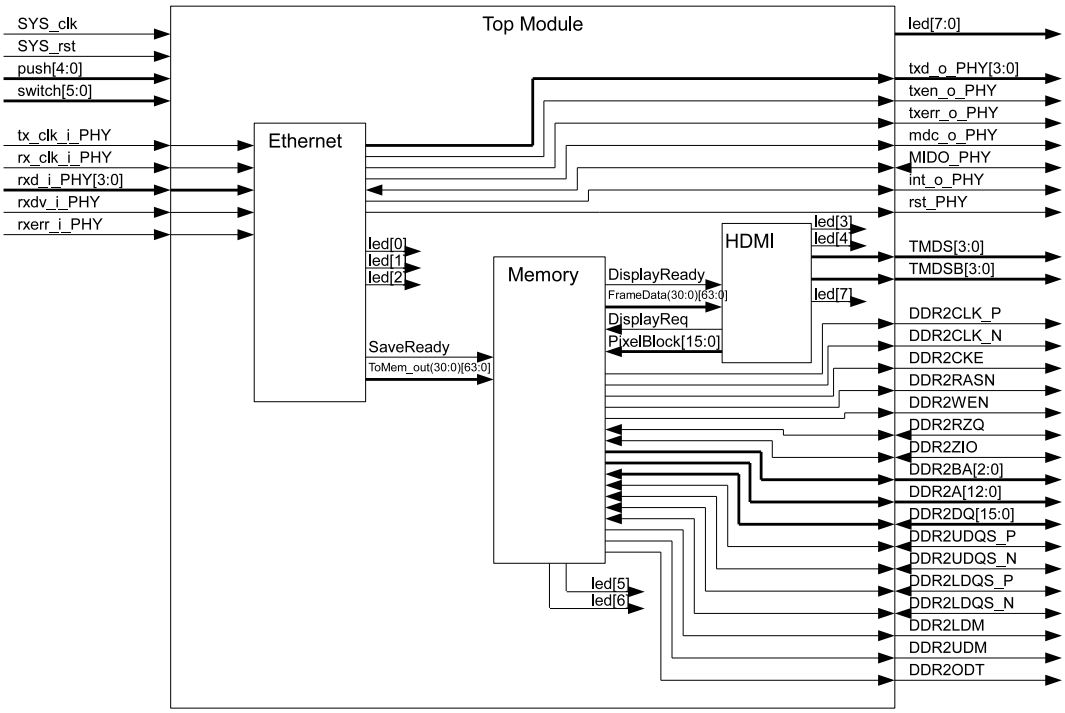


Figure 7.4: Shows how the different parts of the design are linked together.

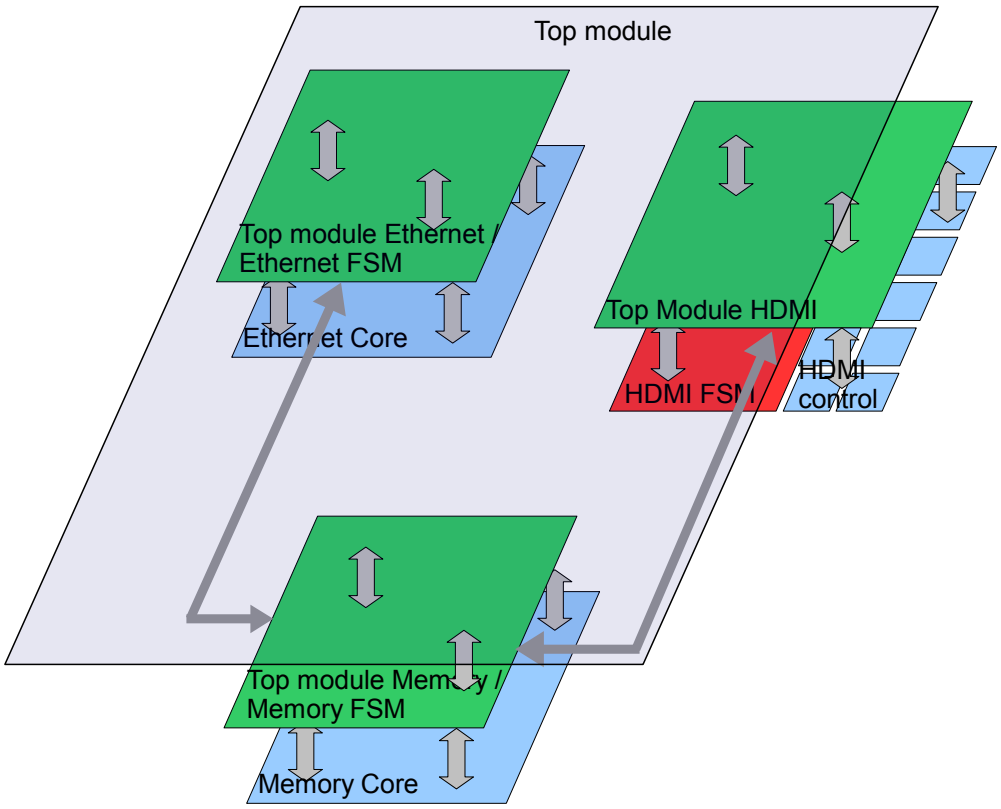


Figure 7.5: Code Structure.

Chapter 8

Result

In this chapter will the result of the project be presented. It starts with the HDTV 720p resolution and then goes on to SVGA resolution.

8.1 Result With HDTV 720p Resolution

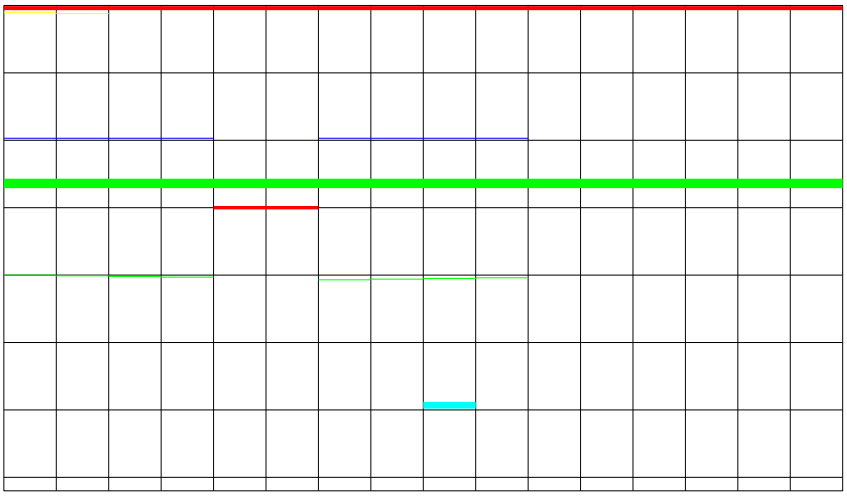
In figure 8.1 the expected and the real results are shown for the HDTV 720p resolution. In figure 8.1 there are a number of different colours on different places. There is a red line at the top of the screen and a green line at around of the middle of the screen that covers the entire screen width. The red line is on two rows and the green line on 14 rows. These two lines are sent from Ethernet frames and saved to the memory, read from the memory and sent using HDMI.

The other lines on the screen are saved to the memory from the FPGA. This means that they are hard coded into the memory from inside the FPGA. When the FPGA is started there is only random data in the memory. So by setting some switches on the board, different pre-defined data can be saved to the memory. In this case the memory is saved with black pixels except on certain places in the memory which will be seen on the screen. These are some yellow pixels in the upper left corner, a blue line almost in the middle. Two red block of pixels side by side in the middle of the screen and a green line also on the middle and finally one block of pixels in the bottom right corner that are cyan.

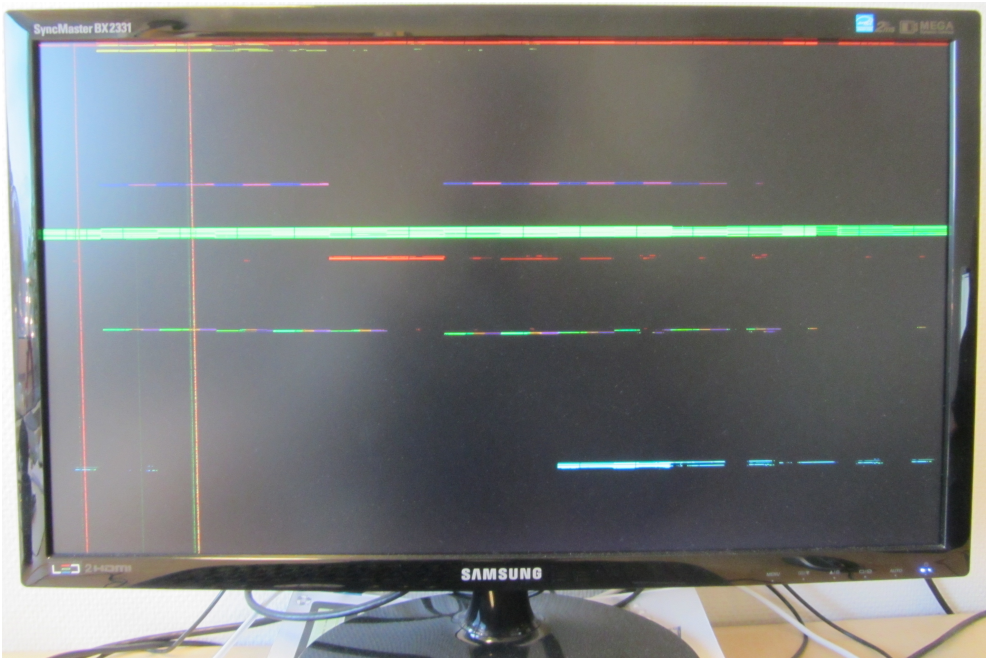
The line that looks blue is not actually completely blue, of the 80 pixels in the block the first 40 pixels are blue followed by 40 pixels in purple. The same thing goes for the green line that are located a little bit below middle of the screen. It consists of three different colours, which are green (40 pixels), brown (16 pixels) and purple (24 pixels). This could be seen more clearly in figure 8.1 b.

I figure 8.1 b the real result is shown. From the figure it is obvious that it does not look exactly as expected.

To investigate the difference further figure 8.2 will be of help. If looking at point 1, it is shifted one block of pixels to the right. Also in point 2 there seams to be some smudges that appear. The



(a) Expected result.



(b) Real result, photo.

Figure 8.1: HDTV 720p (1280 × 720) resolution.

same thing appear on point 3 that it is shifted and the smudges in point 4.

If one examines the code is does not give the shift or the smudges. Before the pixels are sent

to the screen it is checked that the pixels that are about to be sent out belongs to the column of blocks. When changing the optimization level the result is changed. So it is probably due to ISE removing some parts of the code in the optimization that the smudges appears.

It can also be that the memory does not have enough time to read from the memory and get the data ready before it shall be sent to the screen.

The shifted block to the right could be because of a error in writing or reading to the memory. But the writing has been tested by writing some data to the memory and then read the data and send it out as an Ethernet frame again. That has been checked to be correct.

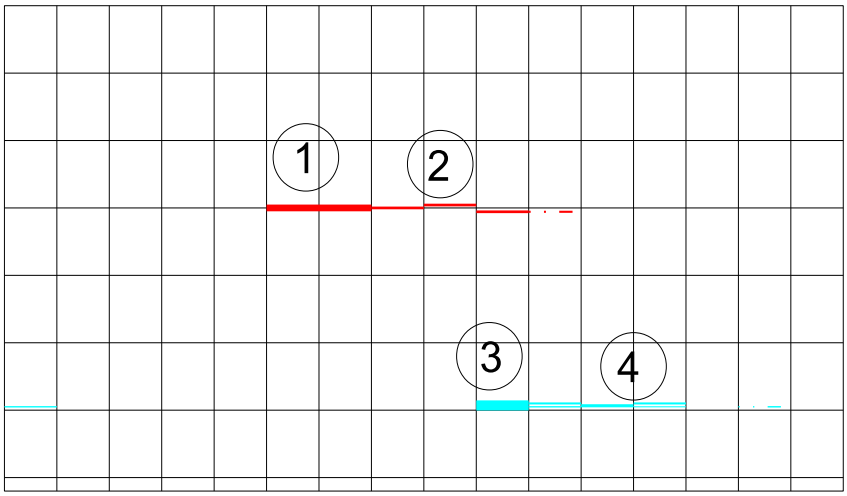


Figure 8.2: HDTV 720p (1280×720) resolution, real result, structural image.

8.2 Result With SVGA Resolution

We now turn our attention to the SVGA resolution to see how the result looks like when using a smaller resolution.

In figure 8.3 the expected result is shown. It uses the same colour as the HDTV 720p. The red line at the top and the green line at the middle is data from Ethernet frames and the other are generated from the FPGA itself.

In figure 8.4 b is the real result shown. I looks like the expected result but there are some differences.

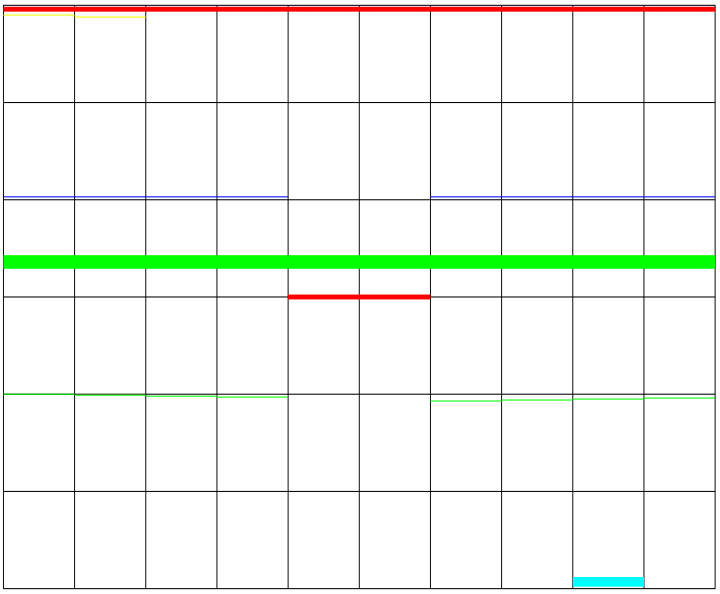
To highlight these difference and explain why they appear figure 8.4 will be of help. The first thing that can be seen is that the ratio between the hight and width is not preserved. This depends on the screen that is used. If looking at point 1 the red block of pixels are shifted one block to the right. One thing that is different from HDTV 720p is that the smudges is only one block of pixels for SVGA as can be seen at point 2.

If looking at point 3 the block is shifted one block to the right and also appears on the first column on the screen. That is what point 4 is indicating. Smudges appears in one block after the block of pixels at point 4.

Before any pixel data is sent out is checked that the pixels belongs to the column that are about to be sent out. This must be some problem when the ISE generated the file that is sent to the FPGA.

One thing that can be seen here is that the smudges are only one block in comparison to the HDTV 720p where the smudges where on several blocks. This could indicate that the memory do not have enough time to read the data and sent it to the screen. Now when the pixel clock is slower the FPGA has more time to read from the memory and the smudges are less.

When watching the screen live it is easy to see that 80 pixels are updated to the screen every time data is received from Ethernet frames since the Ethernet frames are sent with 10 ms interval.



(a) Expected result.

(b) Real result, photo.

Figure 8.3: SVGA (800×600) resolution.

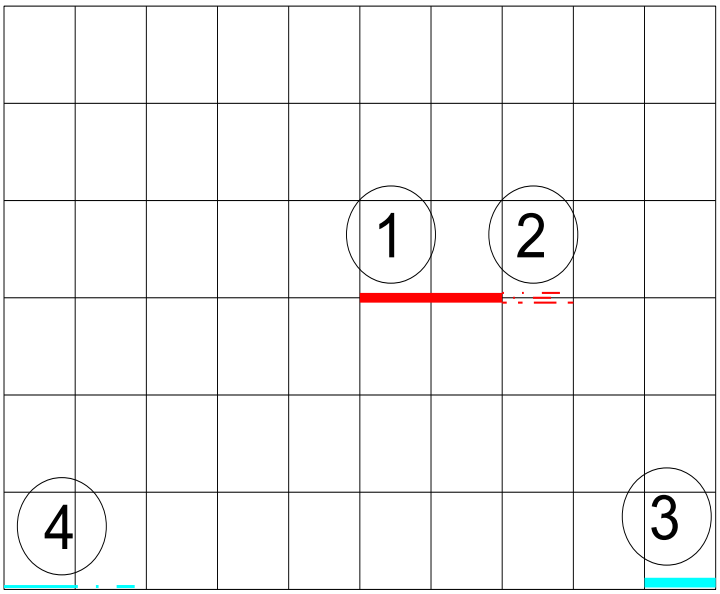


Figure 8.4: SVGA (800×600) resolution, real result, structural image.

Chapter 9

Discussion

Since the Ethernet core that was used in the project was from OpenCores and it was very difficult to understand how it worked it took a long time before the core started to work. A lot of test was necessary to get it to work. Maybe another core should be used that was more documented and had some example code files.

The blocks of pixels was first chosen to be much longer than it is now. But the FPGA had trouble handling such large quantities of data, so the length of data was chosen to 80 pixels in each block and also then chosen for each Ethernet frame that was sent to the FPGA.

Pixels can be lost in the transmission. One scenario is when the FPGA reads a Ethernet frame that is not including any data that is wanted, that is pixel data, an Ethernet frame can be missed since the FPGA is busy processing the other Ethernet frame. Then the pixel data from the missing Ethernet frame is not saved to the memory and the FPGA will not ask for it. Another scenario is if pixel data from an Ethernet frame is being saved to the memory, the memory is busy doing that. In that case the FPGA can not read from the memory and display the data on the screen, this will result in black pixels in that specific place.

The memory is quite large so more than one image frame could be stored in it. Now only one image frame is stored in the memory. But it would be possible to store several image frames in the memory. Then it is required that the Ethernet frames has the same speed as the refresh rate as the screen because otherwise will the a series of frames be displayed repeatedly. If the screens refresh rate is higher then the Ethernet frames can deliver pixel data, some old image frames will be displayed then some new ones then old ones again and so on.

The only real cost of the project was the development board. The software that was used was Xilinx's WebPACK which was free of charge. The WireShark was also free of charge. Then of course was a workplace with a computer and some cables was necessary. But if that is excluded was it only the development board that was needed to be bought. The board costed about $200 for academical use, otherwise it cost about $350.

9.1 Further Work

Here will some example of what could be considered to investigate further:

- Investigate how to solve the problem with the smudges on the screen
- Include that the gateway also can send audio
- Do the other part of the gateway, the HDMI to Ethernet gateway
- Increase the Ethernet rate to 1 000 Mb/s

Bibliography

[1] Polsson, Ken (2011). *Chronology of IBM Personal Computers*. `http://pctimeline.info/ibmpc/ibm1987.htm` [2011-10-19]

[2] HDMI Licensing, LLC (2006). *High-Definition Multimedia Interface Specification Version 1.3a*.

[3] Herrman, J. (2011). How To Extend Your HDMI Cables, *Popular Mechanics*.

[4] Wikipedia (2011). *List of video connectors*. `http://en.wikipedia.org/wiki/List_of_video_connectors` [2011-11-16]

[5] B&B Electronics (2011). *Cat5e Cable Wiring Schemes*. `http://bb-elec.com/tech_articles/Cat5eCableSchemes.pdf` [2011-04-18]

[6] Digilent (2011). *Atlys^{TM} Board Reference Manual*. `http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS` [2011-10-13]

[7] Marvell (2009). *88EE1111 Product Brief Integrated 10/100/1 000 Ultra Gigabit Ethernet Transceiver*.

[8] Texas Instruments (2007). *HDMI Hider TMDS141*

[9] Xilinx (2011). *ISE WebPACK Design Software*. `http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm` [2011-10-17]

[10] Spurgeon, C. E. (2000). *Ethernet: The Definitive Guide*. Sebastopol: O'Reilly Media, Inc.

[11] IEEE Computer Society (2008). *Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*.

[12] Wireshark (2011). *Ethernet (IEEE 802.3)*. `http://wiki.wireshark.org/Ethernet` [2011-12-19]

[13] Kounavis, Michael E., Berry, Frank L. (2005). *A Systematic Approach to Building High Performance, Software-based, CRC Generators*. Intel Corporation.

[14] Techwriters Future (2011). *UDP - User Datagram Protocol*. `http://ipv6.com/articles/general/User-Datagram-Protocol.htm` [2011-12-20]

[15] HDMI Licensing, LLC (2011). *HDMI® FOUNDERS ANNOUNCE INITIATIVE TO BROADEN INDUSTRY PARTICIPATION IN HDMI SPECIFICATION DEVELOPMENT*. `http://www.hdmi.org/press/press_release.aspx?prid=130` [2011-11-07]

[16] Digital Display Working Group (1999). *Digital Visual Interface DVI*.

[17] Lamping, Ulf (2011). *Wireshark User's Guide for Wireshark 1.7*. `http://www.wireshark.org/docs/wsug_html_chunked/` [2012-02-03]

[18] Feng, Bob (2010). *Implementing a TMDS Video Interface in the Spartan-6 FPGA*. Application Note: Spartan-6 Family.

[19] Xilinx (2011). *Spartan-6 FPGA Memory Interface Solutions User Guide*. Spartan-6 FPGA Memory Interface Solutions.

[20] Pedroni, Volnei. A. (2007). *Digital Electronics and Design with VHDL*. Burlington: Morgan Kaufmann.

[21] OpenCores (2010). *Wishbone B4 - WISHBONE System-on-Chip (SoC)Interconnection Architecturefor Portable IP Cores*.

Updated 19 February 2012.