CHALMERS

Wireless transmission of HDMI signals

Bachelor Thesis

SVEN ERIKSSON
MIKAEL LARSSON
JACOB ROSÉN

Department of MC2
Chalmers University of Technology
Gothenburg, Sweden 2013

— THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK —

Wireless transmission of HDMI signals

Bachelor Thesis

by

Sven Eriksson, Mikael Larsson, Jacob Rosén

Supervisor:

Zhongxia He

Examiner:

Vessen Vassilev

Department of MC2
Chalmers University of Technology
Gothenburg, Sweden 2013

Wireless transmission of HDMI signals
SVEN ERIKSSON (svene@student.chalmers.se)
MIKAEL LARSSON (mikalar@student.chalmers.se)
JACOB ROSÉN (jacobro@student.chalmers.se)

MCCX02 - Bachelor thesis at Microtechnololgy and Nanoscience
Bachelor Thesis No. MCCX02-13-09

Supervisor: Zhongxia He
Examiners: Vessen Vassilev

Department of Microtechnololgy and Nanoscience
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Göteborg, Sweden 2013

Abstract

A growing trend of physical separation of digital content and human interfaces is driving the demand for wireless solutions for streaming high bit rate data from one unit to another. This gives the end user freedom of placing equipment wherever wanted without any thought of problems with cables.

The goal is to serialize and later deserialize HDMI signals to enable it to be sent over a radio link. A Xilinx Spartan-6 LTX is used as a hardware platform. To limit the scope of the project it was decided that communication would only go one way. This prevents the negotiation between the HDMI source and monitor from taking place. The negotiation has to be implemented on the hardware platform.

A system design for serialization and sending of the serialized HDMI data was successfully created. Most of the different parts of the system have been verified to work individually, but not together. A fully working prototype is not far away.

Sammandrag

En växande trend av att fysikt separera gränssnitt och lagring påverkar behovet för trådlösa alternativ för att strömma höghastighetsdata från en enhet till en annan. Detta ger slutanvändaren frihet i att placera sin utrustning fritt utan att behöva begränsas av kablar.

Målet är att serialisera HDMI-signaler för att kunna skicka dem över en radiolänk. En Xilinx Spartan-6 LTX har använts som hårdvaruplattform. För att begränsa projektets omfattning bestämdes det att kommunikationen över radion bara kommer gå åt ett håll. Detta gör så att den förhandling som normalt äger rum mellan en HDMI-källa och en mottagare inte kan ske. Förhandlingen måste istället implementeras i hårdvaruplattformen.

En systemdesign för serialisering och överföring av HDMI-signaler skapades. De flesta av delarna i designen har, genom tester eller simuleringar, verifierats att fungera ensamma. En fullt fungerande prototyp är därför inte långt borta.

Acknowledgements

First we would like to thank Zhongxia He helping us through this project with ideas and support during the project. We would also like to thank Jens Kjellerup from Chalmers School of Entrepreneurship who gave us inspiration for applications of wireless transmission of video data. The students Stefan Buller, Joel Magnusson and Torbjörn Rathsman helped us by providing a working LaTeX template for the first pages of this report. They did also read through our report and gave valuable comments.

The Authors, Gothenburg, July 30, 2013

Contents

1 List of abbreviations

ASIC	Application Specific Integrated Circuit
CEC	Consumer Electronics Control
DDC	Display Data Channel
DVI	Digital Visual Interface
EDID	Extended display identification data
FPGA	Field-programmable gate array
HDL	Hardware description language
HDMI	High Definition Multimedia Interface
HPD	Hot Plug Detect
HSync	Horizontal Synchronisation
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
I/O	Input/Output
IP-core	Intellectual Property Core
ISE	Integrate Software Environment
JTAG	Joint Test Action Group
MGT	Multi-Gigabit Transceiver
RGB	Red, Green, Blue
PC	Personal Computer
SCL	Serial Clock Line
SDA	Serial Data line
TMDS	Transition-Minimized Differential Signalling
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits
VSync	Vertical Synchronisation

2 Introduction

The amount of digital content available to both consumers and professional users is constantly growing. The rate of growth is not likely to decrease while the production and distribution of material keep getting faster. Following this comes a demand of fast and easy access to, and ability to present, content at any time. The rapid advancement in the technical field also tends to render hardware outdated within a couple of years. This is why it is usually a bad investment to build all of the physical information infrastructure into buildings as it doesn't provide future flexibility regarding communication standards and placement of equipment. Infrastructure is getting even more important as the trend is to separate content storage, processing and human interfaces. For example, a home user may want to display an online video in the living room using the internet to connect with the workstation in the office, controlling it all from a keyboard sitting on the users knee.

Driven by the telecommunication industry, the capacity of wireless transmission is almost continuously increasing. For example, microwave based point-to-point transceivers can now reach multi-Gbit/s data rates. These improvements open great opportunities of transferring real-time multimedia wirelessly. A wireless infrastructure is better in meeting many of today's requirements such as fast setup, ease of replacement and the possibility of more independent placement.

A widely used standard for transmission of real-time digital video and audio is HDMI (High Definition Multimedia Interface). The standard defines both the physical layer and how pixel, audio and auxiliary data are sent. Using a pair of microwave radios in combination with appropriate HDMI interfaces one can transfer this kind of data wirelessly. With the use of high speed manipulation, additional data may also be sent along with HDMI. This technology applied to the scenario described above is illustrated in figure 1. Another interesting application is integration into hand held units, e.g. tablets and mobile phones, making it possible to easily stream a video to any receiver regardless of if it has been used before.

As many radios only work with serial data while the HDMI send parallel data streams, some sort of interface must be used between the HDMI source and the radio transmitter and then again between the radio receiver and the display. The aim of this project is to make an interface capable of turning the parallel data from the HDMI signal into serial data and then back to parallel data in the form of a HDMI signal. The possibility to incorporate extra features into the solution will also be investigated.

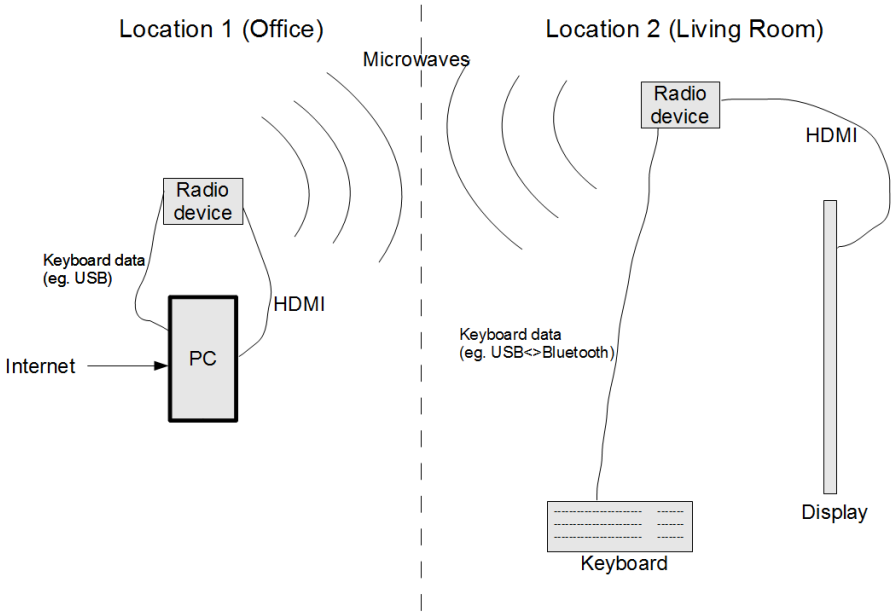


Figure 1: A home application of multi-Gbit/s radio

2.1 Purpose

The purpose of the project is to develop an application utilizing and demonstrating the capabilities of certain high end microwave radio equipment. The purpose is broken down into a primary and secondary goal.

- Primary: Design and implement an interface between HDMI and a microwave radio accepting and providing serial data at 2.5 Gbit/s.
- Secondary: Incorporate one or more commercially viable extra features, such as the ability to embed other signals in the serial data stream or manipulation of HDMI-video, in the solution.

2.2 Method and Planning

The group decided to break down the problem into smaller tasks to be solved in parallel. The interfaces between the parts were not extensively defined due to the nature of the project, crucial information such as the number of channels that had to be serialized was to be determined as work progressed. The work within each part would be iterated as seen i figure 2. The literature

used in this project consisted of the two data sheets for the hardware, the HDMI specification, specifications of other protocols and various user guides from Xilinx.

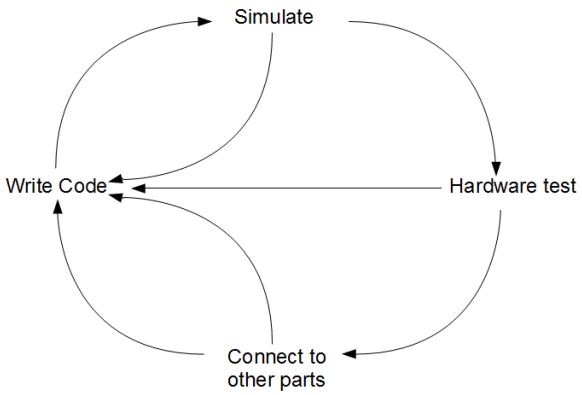


Figure 2: Iterative workflow

The project was initially divided into three tasks. Communication with HDMI devices and with the radio equipment were needed to complete the project. Incorporation of extra features was a stretch goal if we had the time. Each task includes both sender and receiver end versions.

1. Communicate with connected HDMI device. Receive or transmit HDMI data.
2. Incorporate extra feature.
3. Communicate with radio equipment. Serialize and deserialize data.

The two main tasks were worked on in parallel with the focus to get a working prototype done before the end of the project. Figure 3 illustrates how the tasks are related.

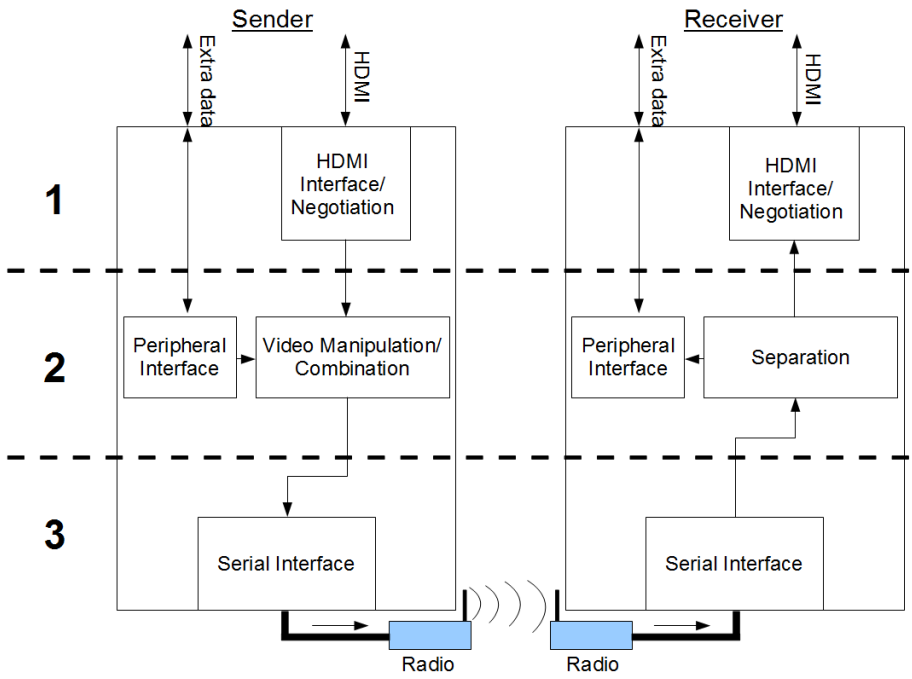


Figure 3: Principal view of the separate subsystems, during development a coaxial cable will be used instead of the radios.

2.3 Hardware Platform and Development Tools

2.3.1 The FPGA compared with ASIC and microprocessor

An FPGA Field-programmable gate array) is an IC (integrated circuit) with the ability to be configured and reconfigured after manufacturing. It is common to use some form of HDL (Hardware description language) to describe the intended functionality and then download the design onto an FPGA. There are several HDLs to choose from. This is not an issue as one design might use different languages through the use of modules. A module is a piece of logic written in one of the many HDLs, it must have both inputs and outputs ports. A module responds to changes in the inputs and calculates the outputs. Every module can instantiate and connect others, creating a hierarchical structure [1].

HDL can also be used to describe the functionality of ASICs (Application Specific Integrated Circuit). ASICs are much cheaper per unit but requires much higher initial investment. If the design of an ASIC isn't working as intended, one must manufacture new units with a new initial investment. Development using FPGAs is an inexpensive way of testing a design before

larger productions. If the productions are small or the product might need future updates it is a good idea to use FPGAs [2].

The FPGA's logic can be programmed to do several tasks simultaneously, unlike a processor which can only perform a single instruction at a time. FPGAs are more expensive than processors. The FPGA's logic is programmable while the processor has predetermined instructions that the developer must use.

2.3.2 Selection of Hardware Platform

An FPGA was chosen to work with because of its ability to process data in parallel and as it can be reconfigured. MGTs (Multi Gigabit Transceivers) are needed for the communication with a radio. The FPGA must contain atleast two MGTs. One for serial input and one for serial output.

The following hardware platforms was used by the project group during the project.

- Enclustra Mars MX2, Xilinx Spartan-6 LXT FPGA Module [3]
- Enclustra Mars Starter, Base Board for Mars FPGA Modules [4]

2.3.3 Software- and Hardware-Tools

The HDLs that were used in this project were VHDL (VHSIC Hardware Description Language) and Verilog. New modules were designed using VHDL. Already existing modules used in the project were written in Verilog. As the compiled programming file for an FPGA is very platform-dependent, the group used development tools from Xilinx as they are the manufacturers of the FPGA.

The software Xilinx ISE (Integrate Software Environment) was used to generate a programming file for the FPGA from the HDL code. The processes necessary for this translation are called synthesizing, mapping, placing and routing.

With the Xilinx ISE comes a simulation software called iSim. This can be used to verify the functionality of a design. iSim provides the ability to trace inputs, outputs and internal signals over the time period simulated. ModelSim is another simulation tool that was used. It works in a similar way as iSim and some of the group members had experience in working with it.

A standard PC (Personal Computer) was used for both code writing and the translation from HDL to a programming file. The generated programming file was used to program the FPGA using a program called ChipScope and a JTAG (Joint Test Action Group) connector. Using ChipScope the internal signals of the FPGA could be monitored.

2.4 Limitations

Some simplifications and limitations of the final product where made during the project.

2.4.1 One way communication

The HDMI system normally uses two way communication in order for the two devices to agree to some parameters they should use, e.g. resolution and color encoding. Two way communication is more difficult to implement than one way communication over the high speed serial I/O (Input/Output). When using one way communication between the sender and the receiver the FPGAs must handle negotiation with the HDMI devices. In order to solve this, the sender FPGA will only accept signals all HDMI devices shall support.

2.4.2 Maximum speed

The hardware is working with a serial data rate of 2.5 Gbit/s. This is lower than the maximum rate of 4.95 Gbit/s of the HDMI specification version 1.3. Because of this the resolution of the video must be limited. This is done during negotiations with the HDMI source.

3 The High Speed Serial I/O Interface

As the demands for higher data rates are growing in virtually all electronic products and on all levels within each system, the industry is currently experiencing a shift from parallel to high speed serial I/O solutions. The use of serial interfaces brings several benefits such as smaller connectors, lower electromagnetic interference, and better noise immunity. Many of these benefits also lead to a lower production cost [5]. In this context, data rates from around 1 Gbit/s up to roughly 10 Gbit/s are considered high speed or Multi-Gigabit.

On today's market, there are mainly three providers of FPGAs with MGTs, these are Altera, Lattice and Xilinx. Using any of these products one can perform real-time processing for applications like image processing, automotive safety systems or wireless communication systems. The Xilinx Spartan-6 LXT used in this project contains several MGTs that can be utilized by the serial interface.

3.1 8b/10b encoding

When transmitting serial data from one device to another, there are issues that have to be dealt with. The following issues are addressed by the 8b/10b (eight-bit-ten-bit) encoding scheme [6].

1. The frequency of the serial data stream may fall below the lower bandwidth boundary of the transmission medium.
2. Transitions may occur too rarely for the receiver to align its sampling clock.
3. The receiver does not know where the word-boundaries are.

The first two are caused by the risk of having data composed of many consecutive ones or zeros. The third issue is caused by the fact that all possible words can be made up by parts of two different words, making it impossible to spot the beginning and end in the stream of ones and zeros. This phenomenon is illustrated in figure 4.

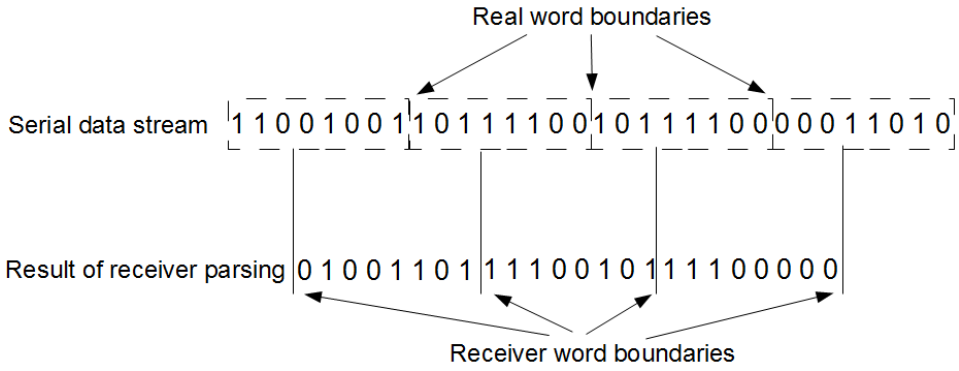


Figure 4: Illustration of word alignment issue

To prevent the transmission of long rows of ones or zeros and in that way overcome the first two issues, the 8b/10b scheme adds two bits to each 8 bit

data word. Every 8 bit word has two unique 10 bit representations. Most of these have the same number of ones and zeros. Some 10 bit words have four ones in the first representation and six in the other. Enabling the transmitter to choose from these representations, a minimum frequency as well as enough transitions for clock recovery to work can be maintained.

As the encoder adds two extra bits, there are some 10 bit words that can never occur while transmitting encoded 8 bit words. These words, called K-characters, can be used for special purposes. In this project one such K-character is used to solve the third problem described earlier in this section. The receiver aligns its word boundaries to that character and then stay properly aligned to the following words.

3.2 Design- and Verification-Method

The serial interface was designed using the software described in section 2.3.3. Additionally, to access the MGT in HDL code, a so called IP-core (Intellectual Property Core) had to be instantiated. An IP-core can be seen as a protected or "black-box" module accessible to the user but not editable. This was instantiated and configured with the help of the Core Generator built into the ISE. The Core Generator is a software accessible from within the Xilinx ISE that enables the user to generate custom IP-cores through a wizard.

To impose stimulus to the data inputs during simulation, a so called test bench was written. A test bench is a simulation-only module that can read from the outputs and write to the inputs of the design being simulated. In this case, the transmitter output was connected to the receiver input. This connection, working as a virtual loop-back cable, enabled simultaneous simulation of both transmitter and receiver.

When the transmitter was working correctly in simulation, the design was also verified on the hardware platform. As the MGT pins of the FPGA are connected to the HDMI connector on the Mars starter board, this verification was done by connecting the custom made adapter board shown in figure 5 to that HDMI connector. One of the coaxial cables was then connected to the oscilloscope.

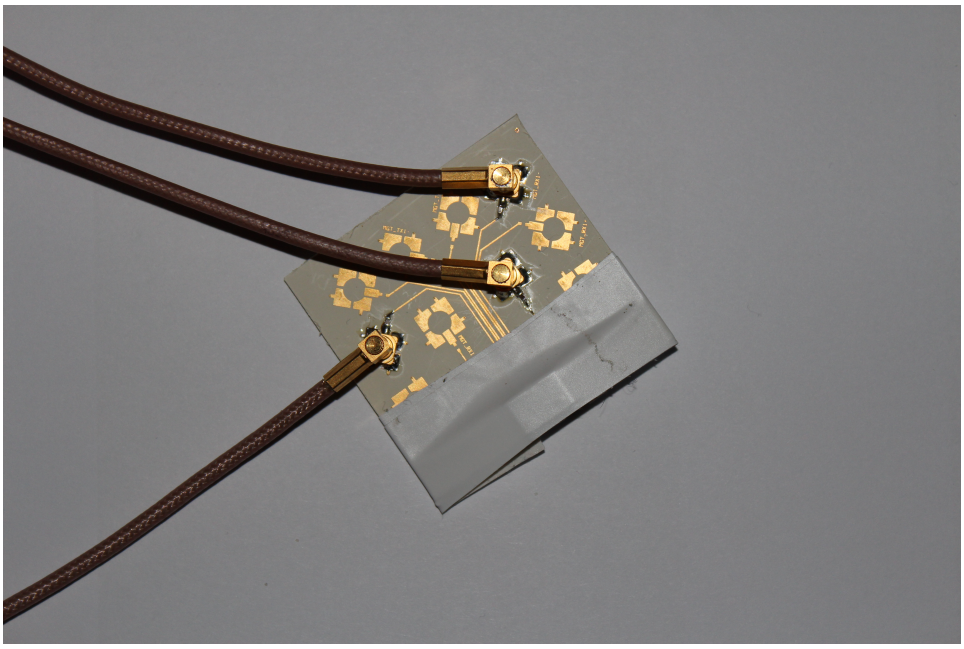


Figure 5: HDMI pins to coaxial connector adapter board

3.3 Implementing a High Speed Radio Interface

The microwave radio that is to be interfaced works at a fixed data rate of 2.5 Gbit/s and does only accept serial data. This means that the only possible data transfer between the FPGA and the radio is a serial data stream at this rate. As the HDMI usually is sent over several wires, these channels have to be serialized to be sent over the radio link. In the receiving end, these have to be deserialized again. Another important function is to handle differences in the data rate of the HDMI data and the serial data stream. The various modes of HDMI use different data rates that may be lower than 2.5 Gbit/s. On top of this, the interface should also take care of all the problems, e.g. word alignment, addressed by the 8b/10b encoding scheme, preferably by implementing exactly that. The functionality to implement is listed below.

1. Sender rate matching and transmission of K-character
2. Serialization
3. 8b/10b encoding
4. 8b/10b decoding and word alignment
5. Deserialization

6. Receiver rate matching

An explanatory overview of the transmitter design can be seen in figure 6.

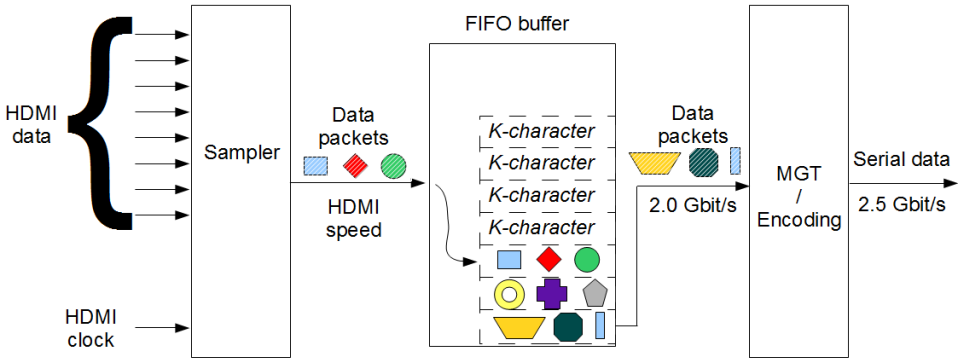


Figure 6: An explanatory view of the transmitter design

Starting from the right in figure 6, the output is a serial data stream at 2.5 Gbit/s. The *MGT/Encoding* block contains functionality for both serialization and 8b/10b encoding. As the encoding extends each 8 bit word to a length of 10 bits, the input rate must be 2.0 Gbit/s to achieve the desired output rate. Hence, one process takes data from the FIFO (First in-First out) buffer and writes it to the MGT at a rate of 2.0 Gbit/s. The buffer is fed with data by another process that samples the HDMI signals and writes their values to the buffer. The rate matching functionality of the system is needed when the incoming HDMI data is slower than the buffer output of 2.0 Gbit/s. When the FIFO buffer is read from at a higher rate than it is filled, it will eventually run empty. In that case, the output takes on a default value. This default value is chosen to be a K-character such as described i section 3.1. These special characters are in this way used to fill out the bandwidth during serial transmission and can later be detected and sorted out in the receiver.

As the receiver's function mainly is to revert the operations made by the transmitter, its structure is also similar to that in figure 6.

Figure 7 contains a view of the actual implementation of the serial interface. The figure shows the so called top module that contains all the other software modules used in the design. Descriptions of the various modules are found in the subsequent sections.

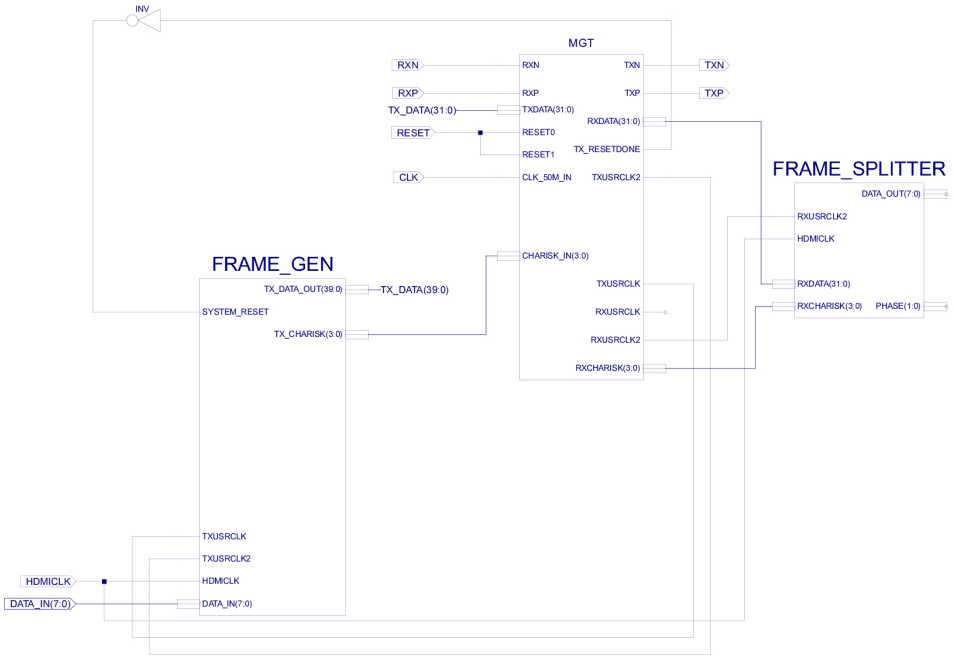


Figure 7: Top module of serial interface

3.3.1 The Frame Generator

The FRAME_GEN module is responsible for the sender rate matching and the transmission of K-characters for alignment. This module is fed all three clocks used in the serial interface.

HDMICLK_IN	Provided by HDMI interface. DATA_IN is sampled on each positive edge
TXUSRCLK	Provided by MGT. Running at the byte-rate, e.g. 250 MHz
TXUSRCLK2	Provided by MGT. Running at 32-bit-word-rate, e.g. 62.5 MHz

To transmit data using the MGT in the configuration used for this project, a 32-bit vector has to be written to its TXDATA-port on the positive edge of TXUSRCLK2. As the width of the DATA_IN-bus is 8 bits, this can be sampled a maximum number of four times during one TXUSRCLK2-cycle. Due to this, four internal vector signals are used to store the sampled input data during the TXUSRCLK2-cycle. Using a simple counter, the first sample is written to "vec0", the second to "vec1" and so on. On the positive edge

of TXUSRCLK2, these four vectors are concatenated to form a 32-bit word which is written to the TXDATA-port of the MGT. After each concatenation and transmission to the MGT, the vectors are reset to their default value of a K-character. If the HDMICLK_IN runs too slow to fill the whole 32-bit word in one TXUSRCLK2-cycle, the receiver will spot this through the detection of one or more K-characters. This technique enables the radio to run at a fixed rate higher than the HDMICLK_IN. It also makes it possible for the receiver to calculate the rate of the transmitter's HDMICLK_IN.

To avoid undefined behavior that might occur if the internal vectors are concatenated at the same time the DATA_IN is sampled, the mechanism shown in figure 8 was added.

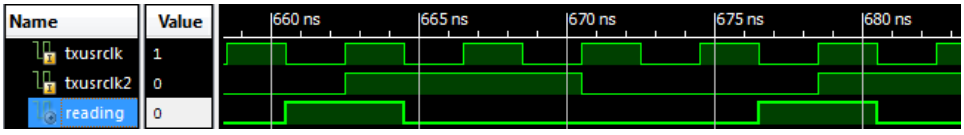


Figure 8: Relationship between MGT-clocks and the reading-signal

The figure shows how an internal signal called "reading" is driven high just before the positive edge of TXUSRCLK2, and then driven low again just after that same edge. This signal is controlled by a process running on the negative edge of TXUSRCLK, the topmost clock in figure 8. As TXUSRCLK2 is edge-aligned to every fourth positive edge of TXUSRCLK, it is possible to use a simple counter to drive "reading" high on the last negative edge of TXUSRCLK before data is written to the MGT. The sampling process only writes to one of the four internal vectors if "reading" is low. If it is high at the time of a positive HDMICLK_IN edge, the sample is written to a temporary vector. On the next sampling occasion, both the current DATA_IN and the value stored in the temporary vector is written to two of the four standard vectors.

3.3.2 The Multi-Gigabit Transceiver

The MGT contains functionality for both the serialization, deserialization and 8b/10b encoding and decoding. Figure 9 shows the contents of the *MGT* module previously shown in figure 7. This contains the module called *gtp* and supporting clock modules. GTP is Xilinx's name for this version of their MGT. The *gtp* module shown here is just a wrapper for the actual instance of the GTP IP-core which contains more inputs and outputs. The wrapper is auto-generated by the Core Generator and allows the user to choose which pins are relevant for their application. All the dynamic inputs and all the relevant outputs are shown in figure 9, but there are some configurations

done inside the wrapper.

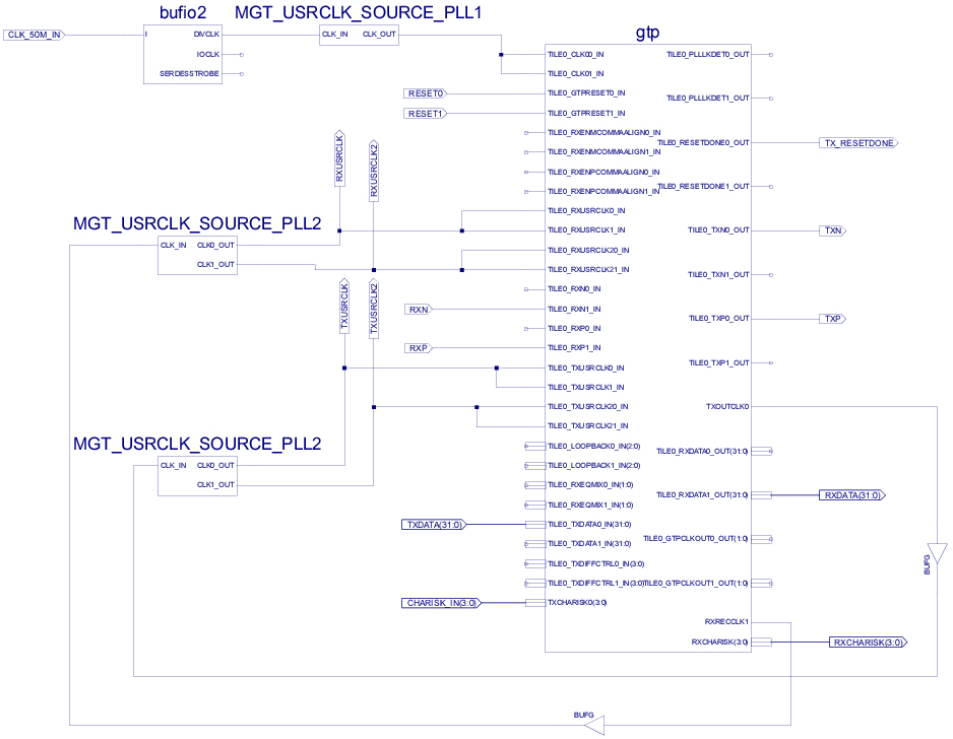


Figure 9: Contents of the MGT module

One instance of the GTP actually contains two transceivers, zero and one, many of the inputs and outputs come in pairs. In this design, transceiver zero is used for transmission and transceiver one is dedicated for receiving, allowing the same software module to be used in both the sender and receiver implementation. This was done to speed up the design process as both sender and receiver share some key resources such as clocking. It also makes simultaneous simulation of both transmitter and receiver easier.

One of the few things that must always be provided for the MGT to work is correct clocking. The first clock to provide should keep a rate corresponding to the line rate divided by 20. In our case this gives us 2.5 GHz / 20 = 125 MHz. On board the hardware platform is a 50 MHz clock connected to the FPGA. The *MGT_ USRCLK_ SOURCE_ PLL1* is a wrapper for a clock divider accessible through the standard library IEEE. It takes as input the 50 MHz clock and outputs a new clock running at the correct rate of 125 MHz. This is connected to both the CLK00 and the CLK01 port of the MGT.

On the right side of the GTP module is an output port called TXOUTCLK0.

This contains an automatically generated clock signal at byte-rate, i.e. 10 serial bits are sent during one cycle. If 8b/10b encoding is enabled, this corresponds to 8 bits worth of data. In this design, with a target line rate of 2.5 Gbit/s, this clock runs at 2.5GHz/10=250MHz. On the left side of the GTP module there are two input ports called TXUSRCLK and TXUSRCLK2. These are clocks used by the transmitter and must be provided by the user. TXUSRCLK should run at the same byte-rate described earlier, and TXUSRCLK2 should run at one fourth of this rate. These two clocks should also be positive-edge aligned. To generate the slower of the two, TXOUTCLK0 is connected to a clock divider module similar to the one described earlier in this section. This module outputs two clocks, one at the same rate as the input clock and one at the input rate divided by 4. These are connected to the two transmitter-clock inputs on the GTP.

An almost identical solution is used for the RXUSRCLK and RXUSRCLK2 which are used by the receiver. The only difference is that RXRECCLK1 is used instead of TXOUTCLK0. This is because the RXRECCLK1 contains a clock that is not only at the correct byte-rate, but kept aligned with the incoming serial data.

In figure 9 there are also two instances of *BUFG* and one of *bufio2*. These are so called buffers, they are necessary for the electrical routing inside the FPGA but do not affect the logic function.

3.3.3 The Frame Splitter

The *FRAME_SPLITTER* module is responsible for the receiver rate matching. This means that it should reverse the operations made by the frame generator described in section 3.3.1. The frame generator in the transmitter allows for slower input than output by adding K-characters to the serial data stream. These K-characters have to be removed by the frame splitter before data is sent to the HDMI interface. It is also necessary to create an output clock with the same rate as the HDMICLK_IN. The technique used to create this clock is described in section 3.3.4.

Table 3.3.3 contains the inputs to the *FRAME_SPLITTER* module.

HDMICLK_OUT	Provided by the clock generation module
RXUSRCLK2	Provided by MGT. Recreated and running at 32-bit-word-rate, e.g. 62.5 MHz
RXDATA	Provided by MGT. Received and deserialized data
RXCHARISK	Provided by MGT. Indicates the positions of identified K-characters in RXDATA

The MGT provides its received and deserialized data through the 32-bit RXDATA-bus. This should be read on the positive edge of RXUSRCLK2.

As the function of the frame splitter is to allow for an output slower than 32 bits per RXUSRCLK2-cycle, the data has to be buffered. Using a simple counter variable called "readcounter", an internal 96-bit vector called “datavec” is successively filled with the contents of the RXDATA-bus. After three RXUSRCLK2-cycles, “datavec” is completely filled. On the next edge of RXUSRCLK2, the first 32 bits of this vector are overwritten and the procedure is infinitely repeated.

The MGT also detects the K-characters in the incoming data stream and indicates their positions within the RXDATA through the RXCHARISK-bus. Each bit in this four bits wide bus corresponds to one of the four bytes in RXDATA. If RXCHARISK, for example, holds the value “0110”, this means that the second and third byte in RXDATA have been identified as K-characters. This information is read at the same time as RXDATA and is also buffered by the frame splitter in a vector called “Kvec”.

The output of the frame splitter is driven by HDMICLK_OUT. On each positive edge of this clock, data stored in “datavec” should be written to the DATA_OUT-bus. As illustrated in figure 10, a simple counter variable called "readcounter" is used to step through “datavec” and output a new 8-bit byte every HDMICLK_OUT-cycle.

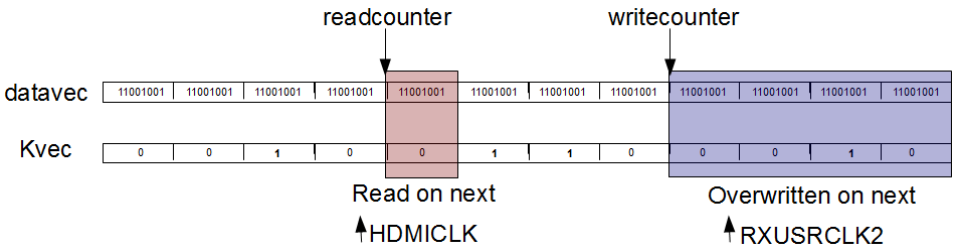


Figure 10: The Frame Splitter vectors

To avoid outputting potential K-characters, the "readcounter" process checks the “Kvec” and skips the bytes marked with ones. The skipping of K-characters is illustrated in figure 11.

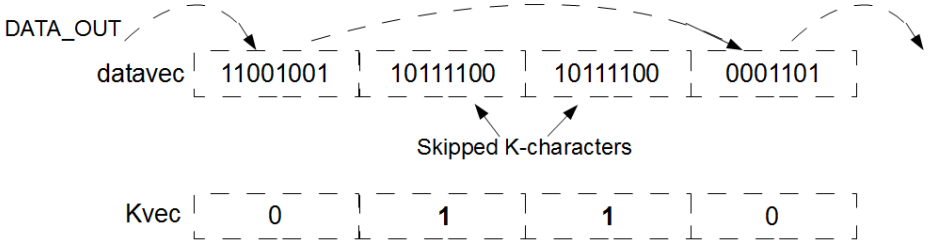


Figure 11: Bytes marked as K-characters in Kvec are skipped in the receiver

The technique used to avoid collision between the"readcounter" and the "writecounter" is presented in section 3.3.4.

3.3.4 Generation of Output Clock

The *HDMICLK_GEN* module shown in figure 7 is responsible for the generation of the output clock, or HDMICLK_OUT, used by the frame splitter. The average rate of this clock must be approaching the rate of the HDMICLK_IN used by the frame generator. If HDMICLK_OUT is slower than HDMICLK_IN, the"writecounter" shown in figure 10 will eventually catch up with the slow "readcounter". This will result in undefined behavior at the point of collision and subsequent data loss. If HDMICLK_OUT is faster than HDMICLK_IN, the fast "readcounter" will eventually catch up with the "writecounter", resulting in similar problems. One solution to this is to use a predetermined rate for HDMICLK_IN. This approach is used by the clock generation module.

The use of a predetermined rate is not on its own a complete solution. This is due to the fact that HDMICLK_IN and HDMICLK_OUT are generated by different oscillators that can never share exactly the same frequency. Over time, even a small difference will lead to the unwanted behavior described earlier. To address the problem of clock generation, the frame splitter is equipped with a 2-bit output-bus called PHASE. On each positive edge of the HDMICLK_OUT, the distance between the "readcounter" and the "writecounter" is calculated. The nominal distance is four bytes, as indicated in figure 10. If the distance gets greater than a preset value, PHASE is set to two. Similarly, if it gets too small, PHASE is set to zero. If the distance is between these boundaries, PHASE is set to one.

Inside the clock generation module, three clocks are generated. These are based on the 50 Mhz oscillator aboard the hardware platform. One of the

clocks is generated in a way that its rate is as close to the predetermined rate of the HDMICLK_IN as possible. One is generated to be certainly faster than HDMICLK_IN and one to be certainly slower. The difference between these three clocks is minimized. The PHASE-bus from the frame splitter is connected to the clock generation module. Based on the value on the PHASE-bus, one of the three clocks is output to serve as HDMICLK_OUT. By selecting the slower clock when the distance between the counters is too small and the faster clock when the distance is too big, a simple control system is implemented.

3.4 Verification of High Speed Radio Interface

An early version of the serial interface was tested on the hardware platform. The version of the Serial Interface tested contained just the MGT and supporting clock modules. During this test, the 8b/10b functionality of the MGT was not enabled. The data transmitted was not dynamically changing but was put directly into VHDL-code. Figure 12 shows the output from the MGT on the oscilloscope.

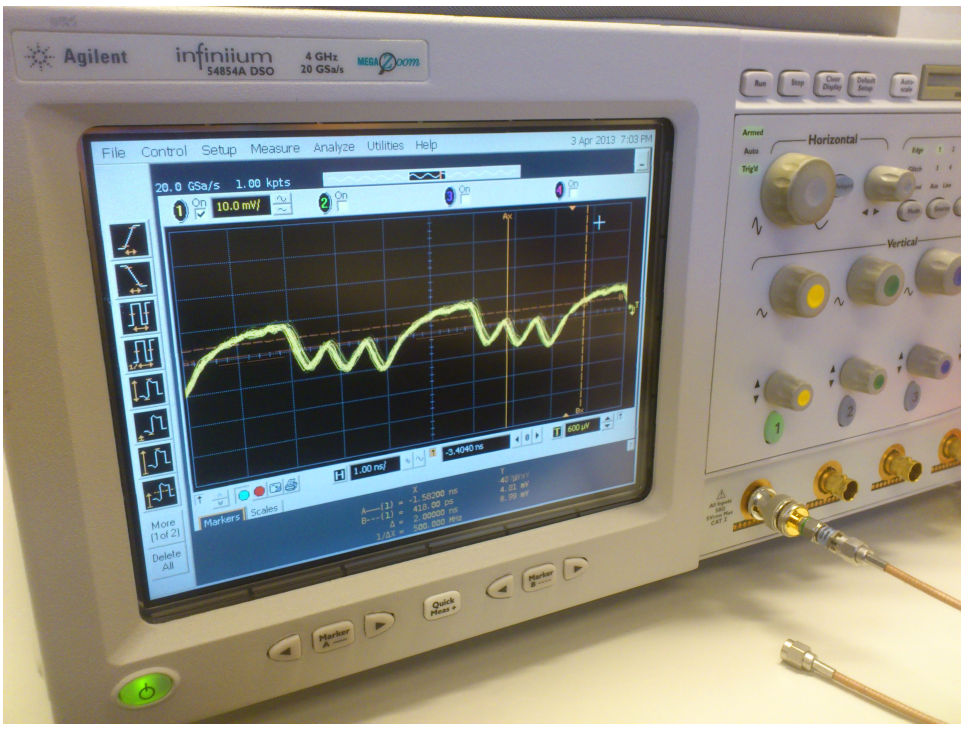


Figure 12: Verification of transmitter function

The hard-coded data consisted of the 8 bits "00010101". These were re-

peatedly output to the oscilloscope. As can be seen in figure 12, the inverse of the transmission data is shown on the screen of the oscilloscope. As the polarity of the output as well as the input on the oscilloscope is a matter of convention that can be changed easily, the inversion is not considered a problem. It can also be noted that the signal does not rise to more than about 50 percent of its final level in one bit-period. As this is a phenomenon derived from the cables, connectors and the oscilloscope used rather than the software, neither this is considered a problem in this project. After this test, the basic functionality of the MGT-transmitter was considered verified.

Later versions of the serial interface have only been tested using the iSim simulator. Repeated simulations were carried out through the addition of the frame generator, the frame splitter and the activation of 8b/10b encoding. At this time of project closure, all functionality of the serial interface, except from the clock generation, has been successfully tested in simulations. The working principle for the clock generator has been decided but not implemented in HDL. Figure 13 shows the result of the latest simulation where the only deviation from a hardware implementable design is that the HDMICLK_OUT is directly driven by the same source as HDMICLK_IN, instead of by the clock generator. The common HDMI-clock is running with a period of 9 ns, or roughly at 111 Mhz. This HDMI-clock rate was chosen as it gives rise to a need for the insertion of K-characters into the serial data stream. The input data is created by a counter and consists of the binary representations of the numbers 0-255 in sequence. As can be seen in figure 13, the output from the frame splitter is, indeed, the binary representations of sequential integers. By observing the values at a certain point in time, the time each byte spend in the system, i.e. the delay, can be calculated. If the point 950 ns is chosen, the value on the input holds the value 105 and the value of the output is 78. By multiplying the difference between these values with the byte period, which in this case is 9 ns, an approximate delay is calculated. Using these calculations it can be noted that it takes approximately 243 ns for each byte to pass through the system. This is considered a good result as such small delays are not noticeable to humans [7].

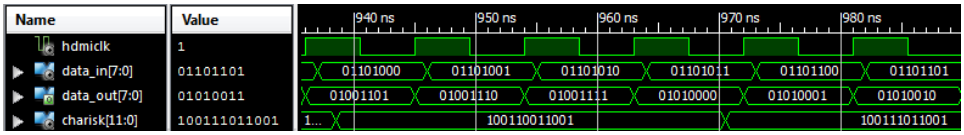


Figure 13: Final simulation

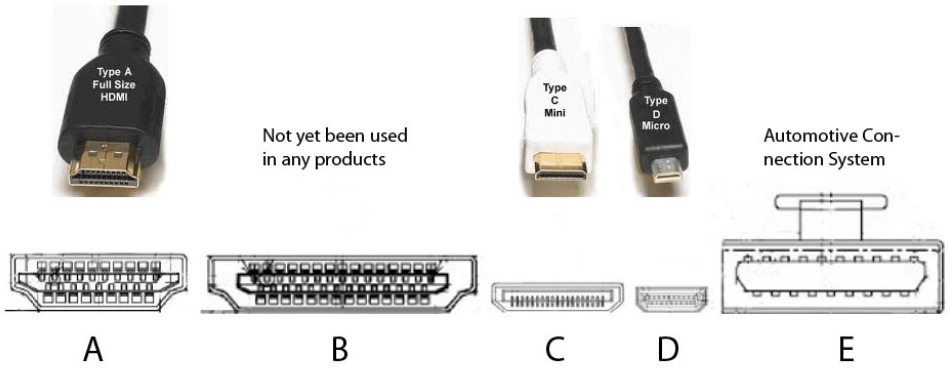


Figure 14: The different types of HDMI connectors, source: [9]

4 HDMI

HDMI is an interface that is designed to transfer video and audio data from a source to a compatible sink, e.g. a monitor or a projector. The video signals are compatible with those of the DVI (Digital Visual Interface) standard [8, p. 136].

Several different video and audio formats can be transferred with HDMI [8].

4.1 HDMI specification

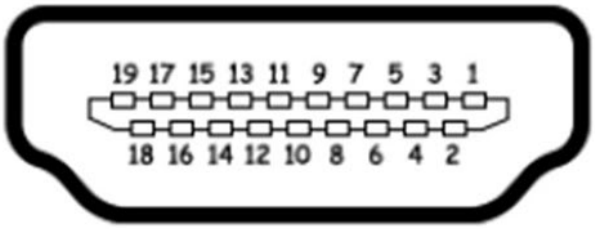
The HDMI standard defines both the physical layer and how pixel, audio and auxiliary data are sent.

HDMI system architecture consists of sources and sinks. All HDMI inputs shall follow the standard for an HDMI sink and all HDMI output shall follow the standard for an HDMI source [8, p. 8].

The HDMI version we have used in this project is version 1.3, see the standard for further reference. This section contains a summary of the major parts of the HDMI standard and other standards related to HDMI.

4.1.1 Hardware

The HDMI standard does currently have five types of connectors, as illustrated in figure 14. Each connector contains atleast 19 pins and cables, as seen in figure 15.



Pin#	Signal	Pin#	Signal
1	TMDS data 2+	11	TMDS clock shield
2	TMDS data 2 shield	12	TMDS clock-
3	TMDS data 2-	13	CEC
4	TMDS data 1+	14	No connected
5	TMDS data 1 shield	15	DDC clock
6	TMDS data 1-	16	DDC data
7	TMDS data 0+	17	Ground
8	TMDS data 0 shield	18	+5V power
9	TMDS data 0-	19	Hot plug detect
10	TMDS clock+		

Figure 15: Pinout of the Type A HDMI connector, source: [10]

4.1.2 Software

The different pins of the HDMI connector have different purposes. All pins except number 14, 18 and 19 is used in three kinds of data channels. The different channels are illustrated in figure 16.

TMDS
Audio, video and auxiliary data is transmitted through the three TMDS (Transition-Minimized Differential Signalling) channels. There is a TMDS clock that is used by the receiver as reference when it recovers data from the three data channels [8, p. 8].

In order to transfer audio and auxiliary data through the TMDS channels, HDMI uses a packet structure. The TMDS encoding converts 8 bit video data into a 10 bit transition minimized sequence. This is done in order to reduce the electromagnetic interference over the cables. 8 bit sequence of audio or auxiliary data is converted into a 10 bit sequence using an error reducing algorithm [8, p. 8-9].

DDC
The DDC (Display Data Channel) is used by the source to request EDID

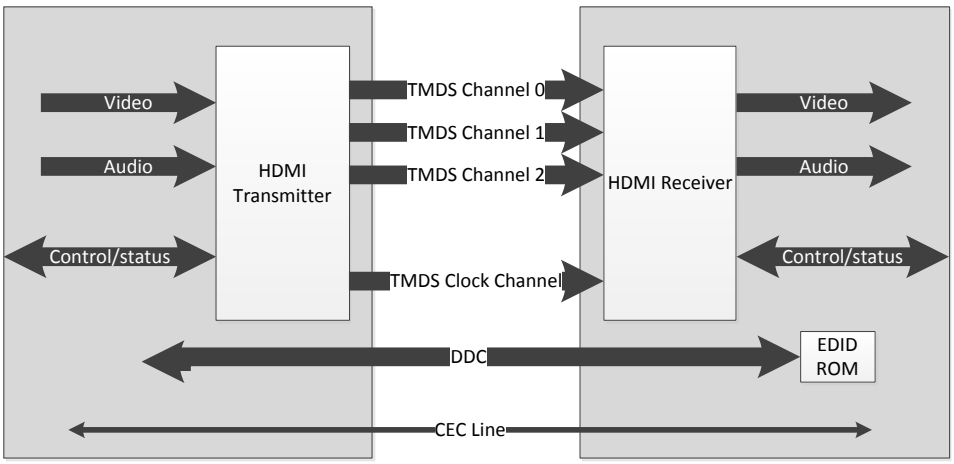


Figure 16: The different data channels in the HDMI system

(Extended display identification data) information from the sink. The EDID information describes the capabilities of the sinks [8, p. 112]. The DDC connection uses two lines, one for a clock and one with data. Data transferred through this channel will be synchronized with the SCL (Serial Clock Line) signal and timings shall comply with the Standard Mode of the I^2C (Inter-Integrated Circuit) specification [11]. The maximum rate for the SCL is 100 kHz [8, p. 122]. A more detailed description of I^2C can be found in appendix A.

The DDC source first sends an address to the sink, if the source receives an answer it proceeds to write or read data. The sink will compare the received address with a stored values, if it doesn't match it will not do anything, however if it match one of the stored values it will answer. The sink accepts two addresses. First the source sends the address 0xA0 for writing what part of the EDID memory should be read and then 0xA1 for reading contents of the memory at the current block [12].

CEC
The CEC (Consumer Electronics Control) channel is optionally used for high-level functions between the source and sink, e.g. automatic setup tasks [8, p. 112]. This channel has not been implemented.

The EDID contains information about which kind of video and audio the sink can use. What resolution the video should be in is also described in the data. The data is sent in 128 byte blocks with the possibility of several extension blocks. HDMI requires atleast one extension block. A sinks EDID data can contain up to 256 blocks of 128 bytes [13].

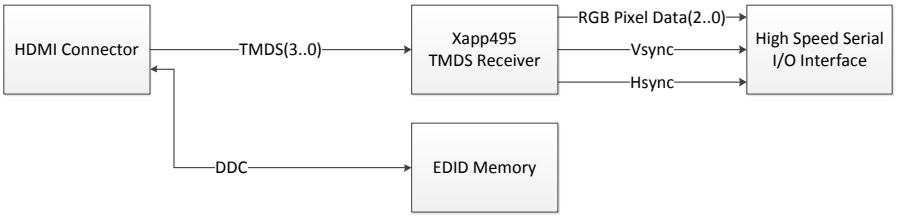


Figure 17: Block diagram of the HDMI part in the sender

The sink will apply some voltage to the HPD (Hot Plug Detect) pin when it is ready to send EDID data. This will only be done once the sink detects a voltage supply from the source on the +5V pin [8, p. 122].

4.2 Using FPGA to implement sender / receiver

4.2.1 Sender

The HDMI part in the sender FPGA has two purposes.

- The first purpose is to negotiate with the HDMI source by sending EDID data through the DDC channel. This is verified in section 4.3.1 and 4.3.2.
- The second purpose is to reformat the video data in the received TMDS signal so that it can be transferred through the high speed interface. This will be verified in section 4.3.3.

The first part has to be done before the second part can begin, as many sources will not send TMDS signals until the negotiation has been done. The TMDS signals are needed to develop a module that will fulfil the second purpose. See figure 17 for an overview of this part.

Negotiation
As the sender can't know which format the monitor connected to the receiver side can support, the EDID data of the sender will force the source to send video in a format that all HDMI devices must support (640x480 at 60Hz). The EDID data of the sender contains only two blocks. The two data blocks contains only the necessary information needed to be identified as a HDMI device. The data of these two blocks are described in appendix B.

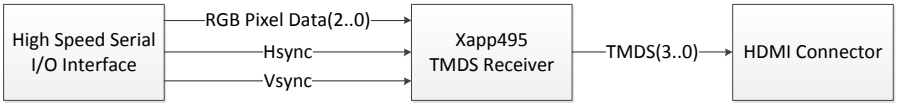


Figure 18: Block diagram of the HDMI part in the receiver

As the +5V pin isn't a port on the FPGA itself, the FPGA can't detect voltage on this pin. It can therefore not be programmed to automatically respond with voltage on the HPD pin. This problem was solved by using two buttons to apply high or low voltage on the HPD pin.

DDC
Dmitry Petrov has published a freeware I^2C module written in VHDL [14]. This was used as a base and heavily modified to suit our purpose.

Video data from TMDS
Xilinx have provided an example code for converting a TMDS signal into RGB (Red, Green, Blue) pixel data. This can be modified and used in the project. If that is done only the RGB ports of the example module will be connected to the serializer. This will decrease the amount of data sent as only video data will be sent. XAPP495 is the example from Xilinx with the example code [15].

4.2.2 Receiver

The HDMI part in the receiver will receive data from the high speed transceiver and reformat it to TMDS data. The same example [15] as mentioned in section 4.2.1 does also show how to convert RGB pixel data into a TMDS signal. As the video is in a format all HDMI devices must support, no DDC and EDID parts are needed for negotiation. As seen in figure 18. This part will be verified in section 4.3.3.

4.2.3 Implement Method and Environment

Hardware environment
The FPGA and the board it is mounted on have been connected to a computer with HDMI and USB (Universal Serial Bus) ports during testing. The HDMI port is used as a HDMI source and the FPGA as a sink. The USB port is used for the FPGA to be programmed and monitored through its JTAG connection.

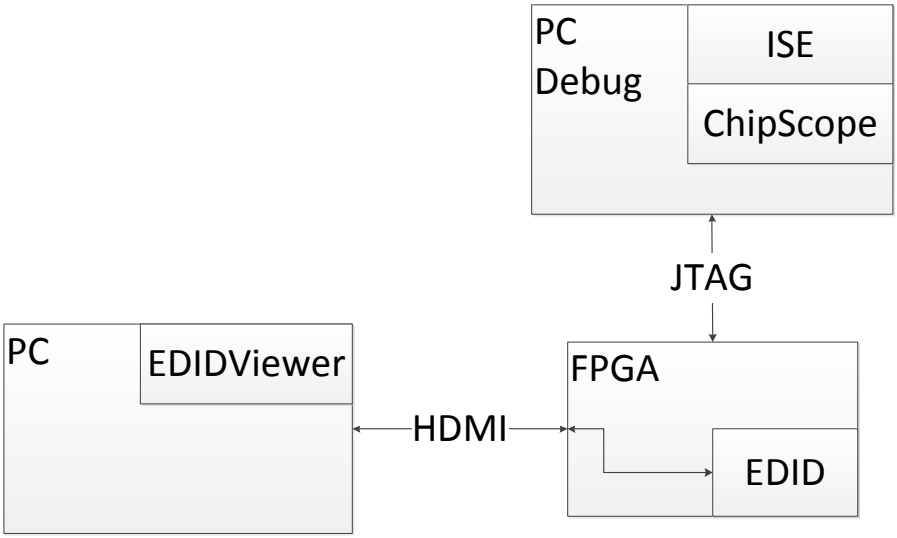


Figure 19: Block diagram of the test: PC to FPGA, EDID. The purpose of this test is to see if the DDC communication is working correctly.

A multimeter was used to determine how the FPGA was connected to the pins.

Software environment
EDIDViewer is a software that can show EDID information from the screens a computer has been connected to. EDIDViewer was used to show the EDID information that was programmed onto the FPGA.

4.3 Verification

The tests outlined in this section will verify that the HDMI parts in the sender and receiver works.

4.3.1 PC to FPGA, EDID

Figure 19 presents a block diagram of this test. The purpose of this test is to see if the DDC channel is working and if the data sent from the FPGA is correct. A PC with EDIDViewer installed is connected to the FPGA with a HDMI cable. Another PC monitors the internal signals of the FPGA by using a JTAG connector and the program ChipScope. If the PC with EDIDViewer is able to read the EDID data from the FPGA this test is successful.

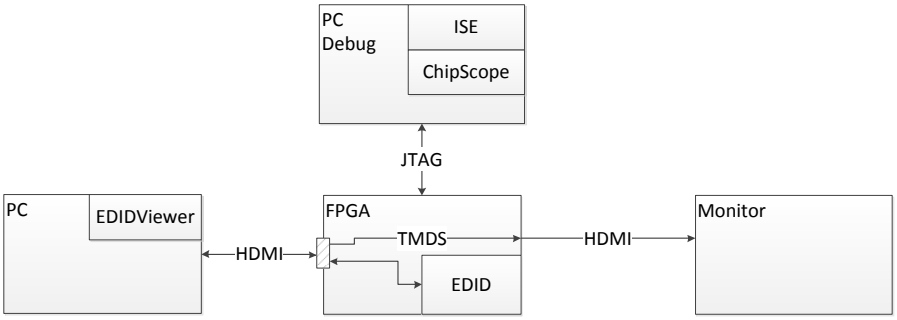


Figure 20: Block diagram of the test: PC to FPGA to screen, HDMI signal. The purpose of this test is to see if the TMDS signals sent by PC can be used by the monitor.

4.3.2 PC to FPGA to screen, HDMI signal

Figure 20 presents a block diagram of this test. The purpose of this test is to see if a TMDS signal is sent from the PC that had previously read the EDID data. The TMDS signal is simply repeated in the FPGA and sent to another HDMI connector. This HDMI connector is connected to a screen. If the screen shows some kind of picture from the PC this test is successful.

4.3.3 PC to FPGA to screen, HDMI signal, with RGB pixel data

Figure 21 presents a block diagram of this test. The purpose of this test is to see if transmitter and receiver from Xilinx XAPP495 is working properly. This test uses the same setup as test 4.3.2 with the difference that the TMDS signal will be converted to RGB pixel data and back to a TMDS signal in the FPGA. If the screen shows some kind of picture from the PC this test is successful.

5 Results

An early version of the serial interface was tested by programming it to the FPGA and then using an oscilloscope and appropriate cables to monitor its output. This version of the serial interface contained just the MGT, with hard-coded transmission data, and supporting clock modules. Figure 12 shows the output from the MGT on the oscilloscope. As the waveform corresponded to the hard-coded transmission data, this basic functionality

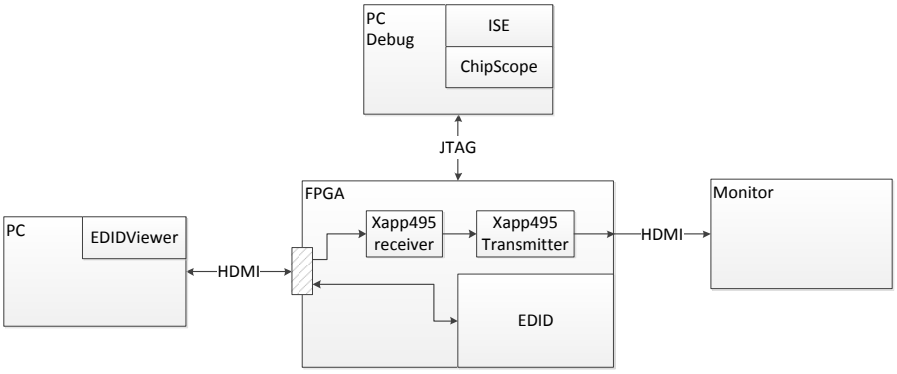


Figure 21: Block diagram of the test: PC to FPGA to screen, HDMI signal, with RGB pixel data. The purpose of this test is to see if the code from Xilinx example XAPP495 is working correctly.

of the MGT within the FPGA was considered verified.

The functionality of the later and more comprehensive versions of the serial interface were only verified through simulations. These simulations were carried out repeatedly through the addition of the desired functions. At this time of project closure, all functionality described in section 3, except from the clock generation which is yet to be finished, have been successfully tested in simulations. In figure 13, the output of the latest simulation is presented. In this simulation, the input data rate was chosen in a way such that all implemented functionality of the system was utilized. That is, the input rate was too slow to fill out the whole 2.5 Gbit/s, giving rise to a need for rate matching. The input data was created by a counter and consisted of the binary representations of the numbers 0-255 in sequence. As can be seen in figure 13, the output from the frame splitter was, indeed, the binary representations of sequential integers. This indicates that the rate matching in both the sender and in the receiver is working properly. By following the reasoning in the end of section 3.4, it can also be noted that the time the serial interface delayed the data was much smaller than what would be noticeable to a human user.

The HDMI part of the project is almost done. The FPGA is now recognized as a monitor by the HDMI source. This shows that the buttons for HPD activation works. Using EDIDViewer we have seen that the EDID data is transferred correctly. This indicates that both the DDC and EDID is now working properly. The tests in section 4.3.2 and 4.3.3 have not yet been done because the hardware we bought didn't work as expected.

6 Discussion

A large part of this project was to learn how HDMI and other standards worksd. This took a lot of time but could maybe have been done more efficiently. The HDMI standard was vague when it described how data was sent through the DDC. When we understood that it was based on two other standards it went much better. It did take time to understand the two other standards as well.

Since the EDID were needed before the source sent any video data, we couldn't work with the TMDS channels at all until the EDID were sent properly. Once the EDID was sent properly a malfunction in the graphic card of the computer we used prevented us to detect any screen using the HDMI port of the graphic card. It took almost a week to realise what caused the problem. When we used another computer as source it worked without reconfiguration.

Our testing equipment for generating HDMI signals were normal HDMI sources. There are testing equipment designed for testing the different parts of the HDMI signal. With access to such equipment we could have worked on the EDID and TMDS parts in parallel. We did not have access to such equipment nor did we ask for it. We have an idea how better testing environment than the one we used could be made of a normal HDMI source and a HDMI monitor. This will be described in section 6.1.1.

Another reason the start went slow were lacking knowledge of how FPGAs work and how to program them to function as wanted. We learnt this during the project. At the start of the project we didn't have access to a full version of XILINX ISE, making troubleshooting of the programmed FPGA almost impossible. When we got the license six weeks into the project it didn't take long until we started finding errors in the code that worked when simulated but not when programmed onto the FPGA.

A problem we encountered late in the project were that one of the HDMI ports which was connected to MGT pins in the FPGA, we couldn't get this port to work with our usual signals. This stopped our tests to just send HDMI data through, since we didn't have a port to output it on.

6.1 Future

The development board have two HDMI connectors but one of those is used for MGT so only one HDMI connector is available for HDMI data. There are 80 pins on the back of the development board to which a HDMI connector

could be soldered. This would enable us to complete the two tests described in section 4.3.2 and 4.3.3.

Once the HDMI parts are done the high speed serial interface is to be connected to the HDMI parts and tested on the development board. The current version of the Serial Interface accepts only an 8-bit wide bus as its input. As it, in a late phase of the project, was decided that only five channels were needed for HDMI-video, the system is far from optimized. The current FRAME_GENERATOR can be used for HDMI-video transmission if the three superfluous bits are set to a constant value in the transmitter and left open in the receiver. This solution is, however, a waste of bandwidth. Better is to modify the FRAME_GENERATOR- and the FRAME_SPLITTER-modules to work with a 5-bit bus. A better utilization of the available bandwidth will allow for higher video-resolutions to be transmitted. If these modifications are successful one could continue with the secondary goal of the project described in section 2.1.

All of the other connectors that are available on the development board, USB, Ethernet etc., requires two way communication in order to work with other units. So if other data signals are to be transferred over the high speed serial I/O two way communication must be implemented.

6.1.1 New testing environment for HDMI

By using HDMI terminal blocks it is possible to separate the cables that make up the DDC from those that make up the TMDS channels. The DDC could be sent to one device and the TMDS channels to another.

During a test of the FPGAs DDC and EDID, the TMDS channels would be sent to a monitor to see if a picture is received. During this an oscilloscope and ChipScope could be used to monitor the signals.

During a test of the FPGAs TMDS receiver and sender the DDC cables could be sent directly to a monitor. The TMDS would be sent to the same monitor when they have passed through the FPGA. During this an oscilloscope and ChipScope could be used to monitor the signals.

This idea is based on separating the cables inside the HDMI cables and sometimes connect them to several places at once. This will maybe violate the HDMI standards description of the functionality of the HDMI cable. It might still work but it must be constructed and tested before it can be used.

References

[1] FPGA, http://www.origin.xilinx.com/fpga/ [May 16, 2013]

[2] FPGA vs. ASIC, http://www.origin.xilinx.com/fpga/asic [May 16, 2013]

[3] "Xilinx® Spartan®-6 LXT FPGA Module", http://www.enclustra.com/en/products/fpga-modules/mars-mx2/ [May 21,2013]

[4] "Base Board for Mars FPGA Modules", http://www.enclustra.com/en/products/base-boards/mars-starter/ [May 21,2013]

[5] A. Athavale. and C. Christensen (2005, April), High-Speed Serial I/O Made Simple(Edition 1.0) [E-book]. Available: http://www.xilinx.com/publications/archives/books/serialio.pdf

[6] Appendix E: - 8B/10B Line Coding, Video Systems in an IT Environment (Second Edition), Focal Press, Boston, 2009, Pages 445-446

[7] International Telecommunication Union. (November 1998). RECOMMENDATION ITU-R BT.1359-1, RELATIVE TIMING OF SOUND AND VISION FOR BROADCASTING [Recommendation]. Available: http://www.itu.int/rec/R-REC-BT.1359-1-199811-I/en

[8] *High-Definition Multimedia Interface*, Specification Version 1.3a, November 10 2006

[9] http://en.wikipedia.org/wiki/File:HDMI_Connector.jpg

[10] http://www.showmecables.com/images/catalog/product/HDMI-PINOUT.jpg

[11] *THE I²C-BUS SPECIFICATION*, Version 2.1, January 2000

[12] *ENHANCED DISPLAY DATA CHANNEL STANDARD*, Version 1.1, March 24, 2004

[13] *VESA ENHANCED EXTENDED DISPLAY IDENTIFICATION DATA STANDARD*, Structure Version 1 Revision 4 Release A, Revision 2, September 25, 2006

[14] D. Petrov, "i2c slave receiver", http://www.mikrocontroller.net/attachment/50416/i2cs_rx.vhd, Sep. 11, 2005 [mar. 13, 2013]

[15] B. Feng, "Implementing a TMDS Video Interface in the Spartan-6 FPGA", http://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf, Dec. 13, 2010 [May 21, 2013]

A I^{2}C protocol

The I^{2}C (Inter-Integrated Circuit) standard is developed by Philips for transmitting data from one IC (Integrated Circuit) to another using few pins at the cost of reduced speed. The interface uses one or more masters that control one or more slaves. All the units connected to the I^{2}C net have predetermined addresses for access. The data is then transmitted over two wires, SCL (serial clock line) and SDA (serial data line). The SCL is driven by the active master and the SDA is driven either by the active master or the active slave.

The transmitting sequence has three stages: address-, register- and data-transmission. First the master transmits a start sequence to signal other units connected to compare the following byte with their stored address. If the address matches the stored address of any unit in the net, the unit sends an acknowledgement by driving the SDA low for a SCL clock pulse to tell the master it received the call. The master then sends a register address, the slave answers with another acknowledgement driving the SDA low. A byte of data is then sent to or read from the slave, followed by an acknowledgement from the receiving unit. The master then proceeds to do a stop sequence. SDA and SCL are left in high impedance until a new start sequence is transmitted.

B EDID data

This is the EDID information that the FPGA contains. It is written in the format:
Byte number. value –comments

B.1 First block

This is the first EDID block. It contains general information about the monitor such as the name of the device and the size of the screen.

0. 00 –header
1. FF
2. FF
3. FF
4. FF
5. FF
6. FF
7. 00
8. 0E –name CTH
9. 88 –name CTH
10. 00 –ID Product Code
11. 00
12. 00 –ID serial Code
13. 00
14. 00
15. 00
16. 2A –Week of manufacture
17. 17 –Year of manufacture
18. 01 –EDID version
19. 03 –EDID revision
20. A2 –Video input definition
21. 00 –Size width
22. 00 –Size height
23. 01 –GAMMA
24. 03 –Feature support
25. 06 –Colour characteristics1
26. EE –Colour characteristics2
27. 91 –Colour characteristics3
28. A3 –Colour characteristics4
29. 54 –Colour characteristics5
30. 4C –Colour characteristics6
31. 99 –Colour characteristics7
32. 26 –Colour characteristics8
33. 0F –Colour characteristics9
34. 50 –Colour characteristics10
35. 20 –Estabished Timings only 640x480
36. 00 –Estabished Timings only 640x480
37. 00 –Estabished Timings only 640x480
38. 31 –Pixelwidth (640)
39. 40 –4:3 ratio, 60Hz
40. 01 –Standard Timing2, do not use
41. 01 –Standard Timing2, do not use
42. 01 –Standard Timing3, do not use
43. 01 –Standard Timing3, do not use
44. 01 –Standard Timing4, do not use
45. 01 –Standard Timing4, do not use
46. 01 –Standard Timing5, do not use
47. 01 –Standard Timing5, do not use
48. 01 –Standard Timing6, do not use
49. 01 –Standard Timing6, do not use
50. 01 –Standard Timing7, do not use
51. 01 –Standard Timing7, do not use
52. 01 –Standard Timing8, do not use
53. 01 –Standard Timing8, do not use
54. D6 –Detailed Timing 1, pixel clock
55. 09 –Detailed Timing 1, pixel clock
56. 80 –Horizontal pixels (640)
57. A0 –Hblank pixel (160)

58. 20 –The two above in 12 bit format, the first 4 bit is in this byte.
59. E0 –Vertical Pixel (480)
60. 2D –Vblank pixel (45)
61. 10 –The two above in 12 bit format, the first 4 bit is in this byte.
62. 58
63. 2C
64. 04
65. 05
66. 00 –Screen size (0mm)
67. 00 –Screen size (0mm)
68. 00 –Screen size (0mm)
69. 30
70. 21
71. 1E -FLAGS
72. 00 –Monitor description block
73. 00 –Monitor description block
74. 00 –Descriptor
75. 0A –Dummy
76. 00 –Descriptor
77. 00
78. 00
79. 00
80. 00
81. 00
82. 00
83. 00
84. 00
85. 00
86. 00
87. 00
88. 00
89. 00 –end of description block, dummy
90. 00 –Monitor description block
91. 00 –Monitor description block
92. 00 –Descriptor
93. 0A –Dummy
94. 00 –Descriptor
95. 00
96. 00
97. 00
98. 00
99. 00
100. 00
101. 00
102. 00
103. 00
104. 00
105. 00
106. 00
107. 00 –end of description block, dummy
108. 00 –Monitor description block
109. 00 –Monitor description block
110. 00 –Descriptor
111. 0A –Dummy
112. 00 –Descriptor
113. 00
114. 00
115. 00
116. 00
117. 00
118. 00
119. 00
120. 00
121. 00
122. 00
123. 00
124. 00
125. 00 –end of description block, dummy
126. 01 -1 extension
127. A9

B.2 Extension block

This extension block and specifically the part vendor specific data block contains the data that describes that the sink is a HDMI receiver.

0. 02 –see table 38 in CEA-861-E
1. 03
2. 41
3. 00

4. 41 –video
5. 81
6. 23 –audio
7. 09
8. 07
9. 07
10. 83 – speaker
11. 01
12. 00
13. 00
14. 65 –vendor specific data block
15. 03
16. 0C
17. 00
18. 00
19. 00
20. 00 –padding
21. 00
22. 00
23. 00
24. 00
25. 00
26. 00
27. 00
28. 00
29. 00
30. 00
31. 00
32. 00
33. 00
34. 00
35. 00
36. 00
37. 00
38. 00
39. 00
40. 00
41. 00
42. 00
43. 00
44. 00
45. 00
46. 00
47. 00
48. 00
49. 00
50. 00
51. 00
52. 00
53. 00
54. 00
55. 00
56. 00
57. 00
58. 00
59. 00
60. 00
61. 00
62. 00
63. 00
64. 00
65. 00
66. 00
67. 00
68. 00
69. 00
70. 00
71. 00
72. 00
73. 00
74. 00
75. 00
76. 00
77. 00
78. 00
79. 00
80. 00
81. 00
82. 00
83. 00
84. 00
85. 00
86. 00
87. 00
88. 00
89. 00
90. 00
91. 00

92. 00
93. 00
94. 00
95. 00
96. 00
97. 00
98. 00
99. 00
100. 00
101. 00
102. 00
103. 00
104. 00
105. 00
106. 00
107. 00
108. 00
109. 00
110. 00
111. 00
112. 00
113. 00
114. 00
115. 00
116. 00
117. 00
118. 00
119. 00
120. 00
121. 00
122. 00
123. 00
124. 00
125. 00
126. 00
127. C6 –checksum