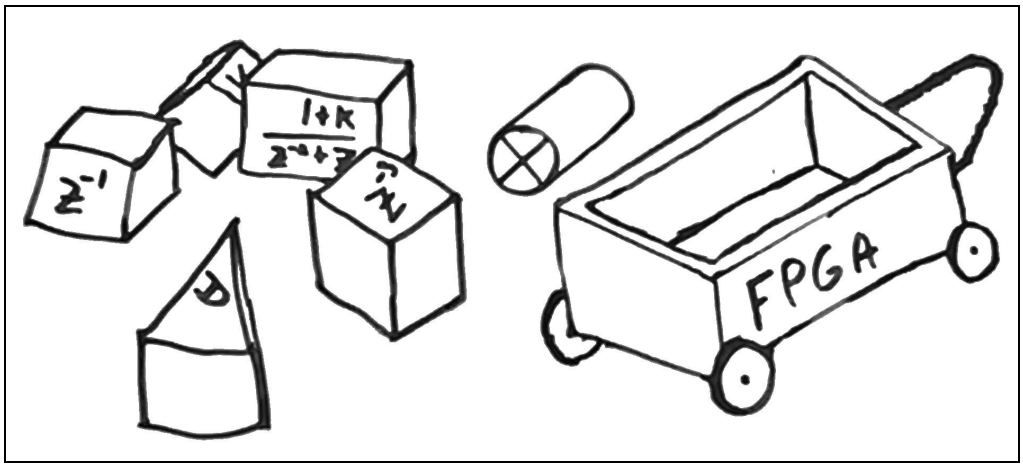
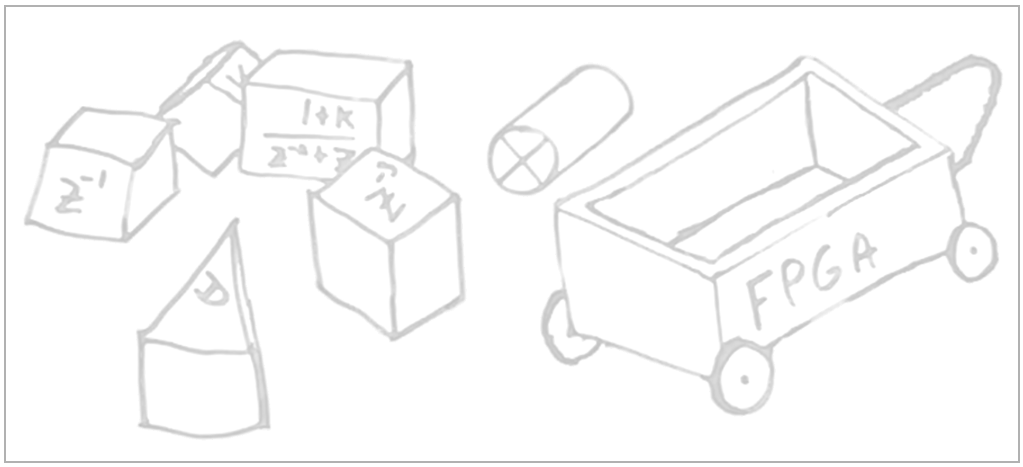
Simplifying development for FPGAs

By means of The MathWorks' HDL coder and
by creating a CameraLink Interface IP



Company:	TNO
Department:	Intelligent Imaging
Company supervisor:	*Ing.* C. van den Berg
Student:	P.P. van der Star
Student number:	11043806
Institution:	The Hague University
Education:	Electrical engineering
Faculty supervisor:	*Ir.* B. Kuiper
Date:	15 December 2015
Version:	1.0



Written by Pieter van der Star during his graduation internship at TNO.
Printed on Tuesday, the 15th of December in the year 2015 CE.
Correspondence to: info@pietervanderstar.nl and kees.vandenberg@tno.nl.

Preface

Well, after I have written this report I get to write something a little less formal. As these are the first sentences of this document you'll probably be reading them first, so I will give a short introduction on how I ended up writing this report. To start at my birth (or nine months prior for that matter) is a little too early I think. So I am skipping the first 21st years of my life. Not that those years do not matter, they do, they made me who I am now, but I think I'll start with the start of my carrier as an electrical engineer.

Once upon a time… No that's no good. The time could be defined better than that. After my previous study I wanted to learn more and thought a study at bachelor level would be a good option. But which study to choose? With my background in theatre-technics a bachelor in the same area would be an option but it did not really feel right. I would re-learn a lot and I thought I could make better use of those four years. What the other options where? I had no idea. It was during the London trip in the final year of my previous study, walking back to the hostel after a Jack the ripper walk, I spoke with a teacher, Jan Brouwer, about this. After I told him I fixed equipment without really knowing how I did it, he suggested electrical engineering. I had never considered thát option. So, skipping forward a bit, half a year later I started my first courses at the The Hague University in Delft.

Within a few days I got my first introduction to digital hardware from Jesse op den Brouw and I remember thinking "This is what I would like to do." One of the last things I did before starting my graduation internship was the minor embedded systems, organised by Harry Broeders. This minor gave me more knowledge of the interaction between custom designed hardware and software as well as providing tools to choose either a hardware or a software solution. (Or of course a combination of both.)

But I am skipping over some stuff. Let's go back to the summer of 2013. It was then that I bought myself an FPGA. I have been tinkering with it ever since. I really enjoy doing this and I wanted to see if I would still like digital design when I was doing that full-time. So when it was time to start looking for a graduation internship I started looking for companies where I could work with FPGAs.

I got that chance at TNO. This document, supplemented with a number of appendices, will tell you all about this. I will confess there were times I thought I would throw the (insert bad words here) FPGA board out of the window, which does not open so I'd have to throw it trough, but overall I enjoyed it.

During my internship I was working in the Intelligent Imaging group. I would like to thank them all for the way they took me in and I want to thank Kees van der Berg especially. He has been supervising my efforts and without his guidance I might have gone along a wrong track somewhere while solving some problems. I also want to thank my faculty supervisor Ben Kuiper for the advice he has given me and the way he has struggled trough the concept versions of this report to give me advice on how to improve the document. But before this turns into half an Oscar speech I'll stop. To you, the reader; I hope you will learn something when you read this document and I thank you for the time you take reading it.

Pieter van der Star

Summary

When designing for FPGAs a lot of knowledge about hardware design and FPGA technology is required. In order to try and reduce the required knowledge, TNO researches two manners of simplifying that design process. One of which is OpenCL, the other is the HDL Coder from The MathWorks. A graduation project was set up to look into the HDL Coder. This software package was to be used to create hardware for a control system. After the control system was developed the hardware would have been tested on a demonstrator system.

The demonstrator system uses data from an image sensor (a camera) as input and from that it controls various actuators. To connect this camera to the system two protocols could be used. These are CoaXPress and CameraLink. Both need a way to connect to an FPGA. In order to achieve this, an electrical camera interface had to be ordered and the logical interface either ordered or developed. There are no companies that can deliver an interface for the CoaXPress that complies with the requirements of TNO. For the CameraLink interface this was no problem. Thus a CameraLink interface was developed.

The interface was created by ordering a card that converts logic levels and connectors. The logical interface was created in-house. The resulting hardware can now be used to connect a camera to the FPGA. The description has been made as generic as possible. This is done so it can be used with a lot of chips. The CameraLink protocol knows a lot of different configurations. Accommodating all the configurations requires a generic description.

For some parts of the logical interface using chip-specific components is the best option. This forced the HDL (hardware description language) code to be less generic. The chip-specific components are selected when needed. The same goes for the parts that are not used in some CameraLink configurations. This option is chosen over the creation of different files because of maintainability. When something should need to be changed, only one file needs editing and than mostly in one place.

The HDL Coder will be used to create the control system. This however is not done in this project as The MathWorks did not think they could provide sufficient support and thus refused to grant a license for the software. This means no system was created and only the development of the camera interface remained. Some research into the capabilities of the HDL Coder was done by means of documentation and questions sent to a user of the HDL Coder. It was found the HDL Coder, at this point in time, is not good enough for simplifying the development. This is mostly due to the fact that timing issues are not easily solved and the documentation of the HDL Coder is not good enough.

The hardware description for the CameraLink however was finished. It was proved to work with a camera transmitting a 2048 by 2048 image over CameraLink in mode base, two ten-bit pixels at a time.

Table of contents

Appendices

1 Symbols and abbreviations

dut
: Design Under Test

FIFO
: First-In-First-Out

FMC
: FPGA Mezzanine Card

FPGA
: Field Programmable Gate Array

GPU
: Graphics Processing Unit

HDL
: Hardware Description Language

HSMC
: High Speed Mezzanine Card

IEC
: International Electrotechnical Commission

IP
: Intellectual Property

LUT
: Look-Up Table

LVDS
: Low Voltage Differential Signalling

MTBF
: Mean Time Between Faults

PCIe
: Peripheral Component Interconnect Express

PLL
: Phase-Locked Loop

PoCL
: Power over CameraLink

SERDES
: SERializer and DESerializer

VCO
: Voltage Controlled Oscillator

VHDL
: VHSIC HDL

VHSIC
: Very High Speed Integrated Circuit

2 Introduction

This report contains the result of a subproject of the project "High Performance Real-time Processing developments". This project is set up to find ways to make development for FPGAs easier. This document describes the results that are part of this project. You can read more about this project and subproject in paragraph 3.2. That whole chapter (3) will give background information about the project. Then the research question is posed and elaborated upon (in chapter 4), followed by an elaboration of the multiple stages of the project (chapter 5). That chapter is also the introduction to the chapters 6 through to 8 as the introduction of those chapters requires a little background information. After chapter 9 the research questions will be answered (in chapter 9) followed by a chapter with recommendations on how to continue the project and the way to implement the results (10). At the end of this document follow a few lists. Like the list of figures, the list of code fragments, the references etc.

Some things written in this document are not completely unambiguous. To make these things clearer the reader should keep a few points in the back of his mind.

- When quantifying information this document uses powers of two with SI prefixes. The Bureau International des Poids et Mesures[1] however says to use the IEC (international electrotechnical commission) units, which are base-2, while they use base-10 numbers [21]. This standard is not followed because the usage of the base-2 with SI prefixes is more commonly used in the computer/information industry. This means 1024 bits go into 1 kilobit (kb). Using the IEC units this would be 1 kibibit (Kib[2]).

- The numbers mentioned in this document are written using the decimal-point notation. So one thousand and a half would be written down as 1,000.5.

- In this document some persons are indicated with male pronouns to increase the readability of the document. The writer did not intend to discriminate anyone. The reader may read the corresponding pronoun as he, she or it wishes.

- The products this document describes are developed using VHDL as the descriptive language. This does not imply other languages such as Verilog do not have that functionality nor does it mean vendors do not have IP (intellectual property) or software support in other languages; those possibilities are just not looked into.

- One of the protocols discussed is the CameraLink protocol. A number of documents refer to the protocol by Camera Link, with a space in the middle. Since the official documentation of the protocol use CameraLink, this document uses that spelling as well.

- Whenever a computer is mentioned in this document a desktop computer or portable computer is meant.

[1] International bureau of weights and measures.
[2] The IEC uses capitals for all prefixes.

3 Background information

Before going into the specifics of this project, some background information on TNO, the company where the project is being held, and the research groups involved in the project is given. This background information is given before introducing the project itself. The project introduction will be followed by an explanation of the system which the project will use as a demonstrator.

3.1 About the company

TNO (Toegepast-Natuurwetenschappelijk Onderzoek[3]) is a Dutch company founded in 1932 by the Dutch government to create knowledge and to make it applicable for the government and businesses. The company has about 3000 employees which are conducting research in the fields of:

- Industry
- Healthy living
- Defence, Safety & Security
- Urbanization
- Energy

TNO has 28 locations of which 22 are in the Netherlands. Although the company is founded, and partially funded by the Dutch government, the company is independent from any government, company or university [22].

The company has two expertise centres. One of which is Technical Sciences, the other is Earth, Life and Social Sciences. Those are then again divided into research groups, each with their own specialty. From these groups three are shown in the simplified organisation chart in figure 1. These three are working together on the project, which is described in paragraph 3.2.

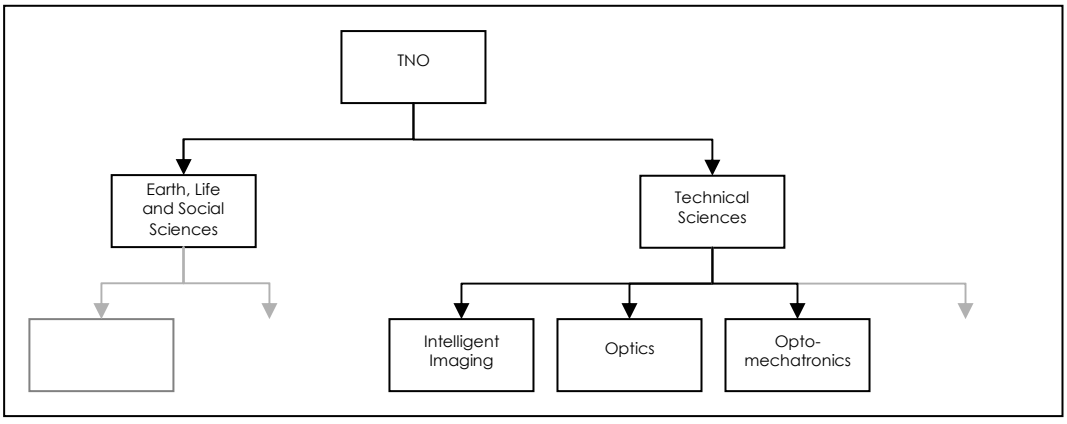


Figure 1. Simplified organisation chart of TNO.

3.1.1 Research groups

From the three research groups involved in the project two are located in Delft. The other, Intelligent Imaging, is located in The Hague. Intelligent Imaging creates systems that analyze images. Such as systems to track people trough images from multiple cameras, image processing for the inspection of train-tracks and a tool to make a 3D-scan of a crime scene. Optomechatronics creates mechanical systems with a high precision for, among other things, space and astronomy and medical equipment. Optics works in the same areas as Optomechatronics, but they work on the optic system such as the lenses and sensors.

[3] Dutch for applied research into the natural sciences.

3.2 About the project

A lot of the technical solutions TNO creates require fast data processing. GPU and FPGA technology is often used[4] in order to achieve this speed. At this point in time usage of the latter requires a lot of knowledge about FPGAs. This knowledge is not always present with the engineers who have to design the systems.

To look into the possibilities of simplifying the design process, the project "High Performance Real-time Processing developments" has been launched. This project will look at two options; (Altera) OpenCL and The MathWorks' HDL Coder (further in this document this is called "HDL Coder").

A part of the project has been assigned to a student who has made that part his thesis project. This subproject will look into the possibility of using the HDL Coder to simplify the development for FPGAs. This option is explored because one of the departments involved uses Simulink in a lot of their solutions. Simulink is a software package with which (control)systems can be designed and simulated using block diagrams [27]. In the rest of this document, whenever "project" is mentioned, this sub project is meant. This project only looks at the HDL coder. OpenCL will be looked into at a later time. By that time this report will already have been finished. This report will look at the possibilities and limitations of HDL Coder. The system on which these methods will be tested is discussed in paragraph 3.3.

As stated in paragraph 3.1 the "High Performance Real-time Processing developments" project is carried out by a collaboration of three different research groups. Those groups are:

- Intelligent Imaging
- Optomechatronics
- Optics

Apart from these three research groups another company[5] is indirectly involved. This company became involved at a later stage of the project and TNO and this company will share experiences with both OpenCL and the HDL coder.

3.3 About the demonstrator system

To test the HDL Coder a control system needs to be designed and implemented. The control system needs a system to control. This system is a (planar) pick-and-place machine (please see figure 2 on page 10). This machine is chosen because the machine was used as a demonstrator for a vision-in-the-loop project in the past and part of the processing is done on an FPGA. To reduce the project costs this demonstrator would have been reused. Due to unforeseen circumstances[6] the methods have not been tested on a system. But, if there would have been time enough, the tests would have been carried out on the system described here.

[4] Among others, by the research group Intelligent Imaging.
[5] This company will not be named.
[6] These will be discussed later on.

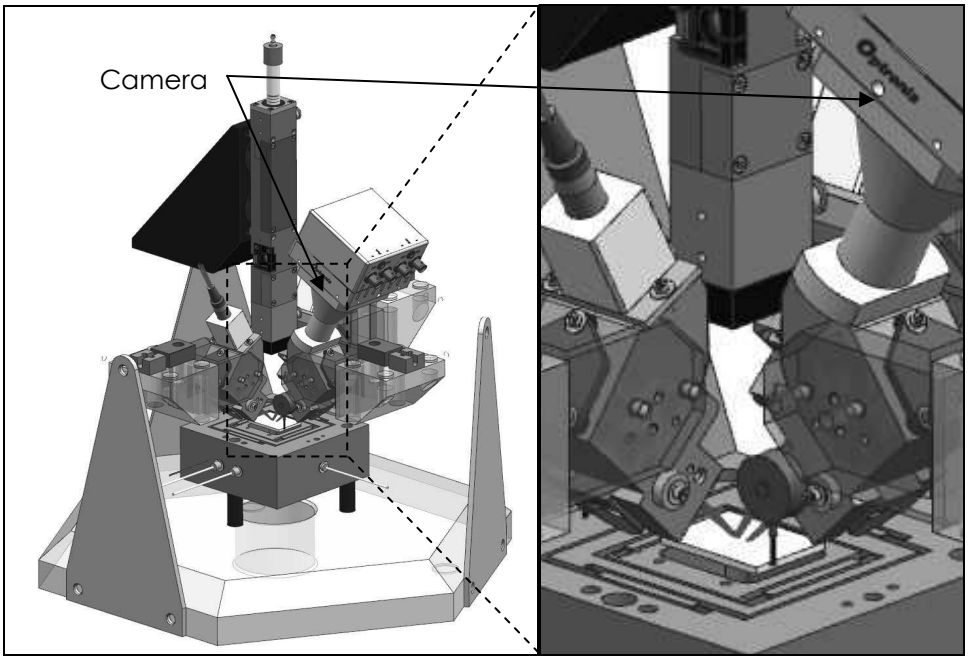


Figure 2. Design sketch of the demonstrator system.

This demonstrator system comprises of four basic elements:

- A camera
- An FPGA
- A computer
- Some actuators

These elements are part of the control loop as can be seen on the left in figure 3. The computer generates some jitter. This is caused by interrupts on the (non-real-time) operating system. This jitter makes delays unpredictable. The minimum and the maximum time could be calculated when working with a real-time operating system[7], but for control systems a larger delay is preferred over an inconsistent delay. If the computer could be removed from this system the jitter would be also removed, or at least greatly reduced. This means the FPGA, which now only does a bit of pre-processing to reduce the time it takes to write the image to the computers memory, will have to take over from the computer completely, resulting in the control loop on the right of figure 3.

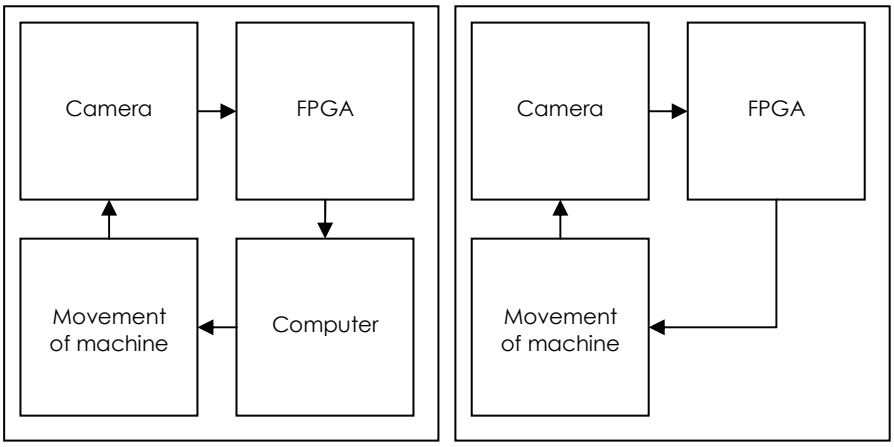


Figure 3. Simplified block diagram of the control loops for the demonstrator system.

[7] In practise the jitter will not be completely removed.

The control loop is formed by the four, or possibly three, elements as mentioned before. The camera has been mounted on the machine in a place where it moves with the machines moving parts. The position of the machine is calculated from images the camera provides. By comparing the current image with the previous one the distance moved can be calculated, and by moving to a position which gives a known image the image acquired in that position can be compared to the expected image. Thus the positional offset can be calculated and corrected for.

4 Research question

The project looks into the possibility of using The MathWorks' HDL Coder to simplify the development for FPGAs. Thus the main question to be answered is the question whether or not The MathWorks' HDL Coder is a good tool for this purpose. In order to find an answer to that question a number of other questions arise and will need to be answered as well.

The HDL Coder is a software tool, but not the only one on the market. As stated before TNO will also be looking into the OpenCL environment, so ease of use of the software will be a factor in the comparison of those two.

Also it could be some designs use off-the-shelf code. To incorporate this code into the design, whilst keeping the design process simple, requires the ability to insert custom-made HDL-code into the design.

As the main goal of this project is to try and limit the required knowledge about FPGAs, the question about the extent of knowledge required arises. What does someone working with the HDL Coder need to know about things as register timing and chip-specific components (e.g. multipliers)?

Another big problem might be the manufacturers' support. If only a handful of boards can be used it severely limits the possible applications and, depending on the devices, could require designs to run on chips that are not quite suited for that design. Meaning either a too slow or a way to big chip. The latter is no problem on the design front, but financially it is something that would rather be avoided.

So in order to determine if The MathWorks' HDL Coder is a good option for simplifying the development for FPGAs the following questions will have to be answered:

- What is the ease of use of the software?
- Is it possible to insert custom-made HDL-code into the design?
- To what extent does the user need to know about digital hardware/FPGA design?
- Which FPGA boards are supported?

These questions will be answered in this document. The next chapter will discuss the steps taken to find the answers to these questions.

5 Project stages & further introduction

The project has multiple stages. These stages could be divided into two categories. One category would be the research into the HDL coder, the other the development of a hardware description for the camera interface. Figure 4 shows the initial planning of these stages. Below figure 4 the steps are explained.

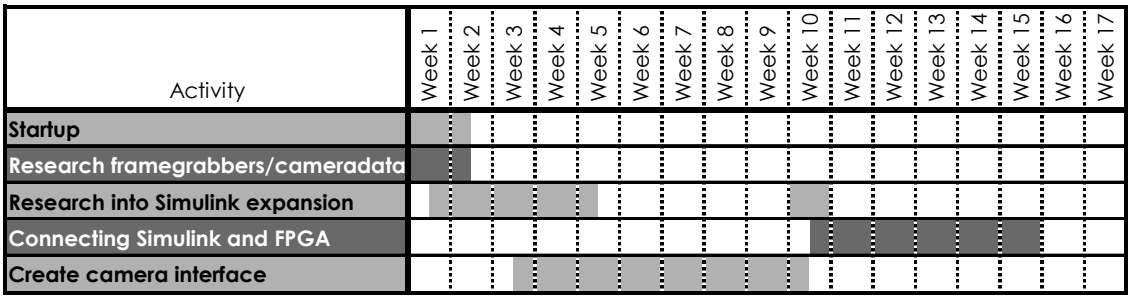


Figure 4. Stages of the project as planned.

As stated in paragraph 3.3 the system has both a camera and an FPGA. These will have to exchange data in one way or another. To do this an interface must be developed. Before doing that however, the capabilities of the HDL coder were superficially looked into. It would be a waste of time to make an interface only to find out at the end the HDL coder does not know what to do with that. Also it may that what the "High Performance Real-time Processing developments" project wants to achieve cannot be done with the HDL coder. So first the capabilities of the HDL coder were looked into. After that an interface for the camera was developed.

The camera interface is created as a bridge between the camera and the control system in the Simulink environment (please see figure 5). The interface is used to control the camera and manipulate the signals from the camera in such a way that the control system only gets pixels as an input.

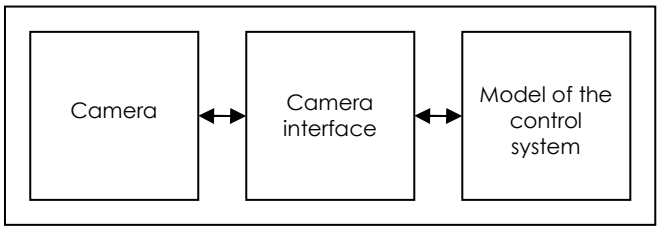


Figure 5. "Position" of the camera interface.

Once the interface was built the connection with the Simulink environment could be made. This stage however was not started. Chapter 6 will discuss why it was that no connection with the Simulink environment was made. Although no connection with Simulink was made, the abilities of the Simulink environment, when designing for FPGAs and the performance[8] of the HDL Coder will be discussed in chapter 6 as well. Chapter 7 will discuss the way the camera will be physically connected to the FPGA and the next chapter (8) will provide more information about the development of the logical interface.

[8] Only second-hand experience is used here.

The order in which the stages are described is not the chronological order. This order is used to create a more coherent structure for this document. The chronological order is given in figure 6. Note the research into the Simulink expansion is split. A part of this is also done at the end of the development of the camera-interface. This was planned as an introduction into connecting the FPGA and Simulink and to implement the designed interface as a Simulink block.

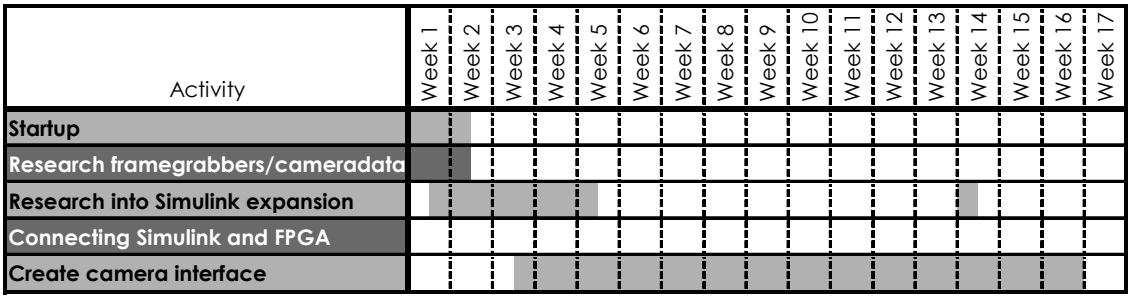


Figure 6. Stages of the project with the time they took.

6 Simulink and FPGAs

The purpose of this project is to look into the possibility of simplifying development for FPGAs. One way to do this is to use the HDL Coder from The MathWorks. First the capabilities of the software were looked into as well as the limitations. The latter was done by searching web forums for problems users of the software encountered. The results of this research can be found in appendix I. Paragraph 6.1 will discus the results of this research.

This research alone is not enough to say anything about the HDL Coder. Practical trials often make things much clearer and to do this a license is needed. In paragraph 6.2 the process of acquiring a license is described. Getting a license was not as straight-forward as first thought; this had some consequences for the project. These consequences are described in paragraph 6.3. The next paragraph (6.4) discusses the capabilities of the HDL coder. What it can and cannot do as well as, to some extent, the user-friendliness. The knowledge needed to work with the HDL Coder is discussed in paragraph 6.5.

6.1 Expanding the use of Simulink's HDL generation

Simulink is mostly used to design control systems. A control system has at least one input and one output. For this project that input is a camera. To connect this camera something must enable the system to read from the camera. Leaving the physical part (ports and protocols) aside there is a couple of options to realise this. A block that handles all the signals can be created within the Simulink environment and the same thing can be done by writing HDL, which will be inserted in the correct place during the HDL generation. Within the Simulink environment multiple options exists to generate the module. This could be done with a system function, but it can also be done using vendor-specific blocks. Because the system will not be simulated with the camera-interface as a part of the simulation[9], the interface will be developed by writing HDL and later adding this to the Simulink environment.

6.2 License

When using the HDL Coder or when reading the documentation[10] a license is needed. TNO already has an HDL Coder licence which is used in another project. That license is in use and is only for one specific computer. Late into the project The MathWorks was contacted to inquire about the licence structure and the possible acquisition of a licence for the remaining duration of the project[11]. The experiences with The MathWorks in acquiring licences were good, so no problems were foreseen.

The MathWorks thought about issuing a trial license. They did not however, because they want to give support and it is not financially viable to give support while it is not being paid for. As the project will partially determine if TNO will use the HDL Coder, The MathWorks want to make a good impression and they are reluctant to let TNO make that decision based on the work of a student who has little experience with Simulink and will work with limited support.

Another reason might be the HDL Coder may not quite do as advertised and they want to manage the expectations. This theory is supported by the fact they initially only wanted to grant the license when the project had specific targets and they could cooperate in reaching those targets. These reasons are all understandable.

[9] The images are the input on the simulation, not the camera signals.
[10] Without a license only parts of the online help pages from The MathWorks can be read. The documentation is available on third-party websites.
[11] At the moment of first contact this was five weeks from the end.

After TNO explained they do not make decisions like this just on the work of a student and that support can also be given by the person within TNO already working with the software they said they would reconsider and be in contact. After a week they had not yet replied. The MathWorks has not been contacted about this again.[12]

6.3 Not using the HDL Coder

Since time did not allow for the project to start using the HDL Coder as planned another option was discussed. This would see the use of the HDL Coder to implement a simple calculation on the image acquired and would not require the control system to be developed. One TNO employee has some experience with the HDL Coder. He was contacted to get insight in the time it might take to implement this idea. He said to have no time to give good support and did not think this solution could be created in the time allotted. Taking all this into consideration the use of the HDL Coder was dropped and the only product to be delivered is now the CameraLink interface.

6.4 (in)Capabilities of the HDL Coder

To see if the HDL Coder is able to simplify the design process for FPGAs the capabilities are looked into. This was done earlier as well (see paragraph 6.1) but to make a better judgment on the simplification more thorough research was done.

The HDL Coder has not been used, but some second hand experience is available in the form of questions on webfora and the documentation provided by The MathWorks itself. A lot of questions on the MATLAB forum are about the usage of mathematical functions e.g. exp(). The manual states these functions must be replaced by a lookup table (LUT). The user will have to generate these LUTs. The manual uses 19 steps to describe how to do this. The user first creates the function he wants. Then creates a wrapper function that calls that function, followed by the creation of a test file in which an array with the values is generated (the LUT). Then a new project is created to where the fixed-point conversion is done. If all this is good then the function must be added to a list with functions that will be replaced by a LUT. When the user becomes more proficient in the conversion it may be a few of these steps can be skipped, but it seems a complicated process.

According to The MathWorks' HDL Coder website, the HDL Coder is target-independent. This means it generates either generic HDL or has ways to use chip-specific features. The workflow can be automated, where the steps of the workflow are as described in figure 7 on page 17. This shows the synthesis and analysis is done with Xilinx' ISE[13] and Altera's Quartus II.

Some years ago Xilinx stopped with the development of ISE and Altera has recently replaced Quartus II with Quartus Prime. This could mean new chips cannot use the description generated by the HDL Coder. Although, if HDL is generated any synthesis tool should be able to process the files.

[12] The MathWorks has not been asked for a reaction to this text, and it is the view of the writer of this document only.

[13] On another part of their website The MathWorks mentions this can be done with Xilinx' Vivado as well [56].

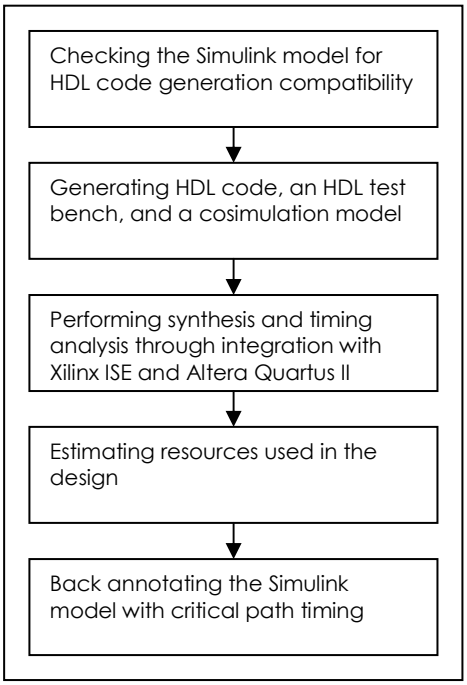


Figure 7. HDL Coder workflow [42].

Apart from the HDL for synthesis the HDL Coder also generates testbenches. These testbenches are generated from the MATLAB or Simulink model. If the entire model is the design under test (dut), then no testbench will be generated [42][51]. Earlier (in paragraph 6.1) is mentioned that the HDL Coder was found to support the insertion of custom-made HDL.

Users of MATLAB use floating-point calculations without ever needing to know they do. In digital hardware floating-point arithmetic is seldom used as it greatly increases the size of the design, and also may reduce the speed. Thus the HDL Coder requires the user to create a model with only fixed-point numbers. Apart from the questions about the mathematical functions most questions on the fora regard to the use of floating point numbers. A lot of the questioners do not know about having to convert the floating point numbers to fixed-point. As one person demonstrates by asking how the conversion works because he has an error after running the checks. The answer states that he needs to convert the floating-point numbers to fixed-point. In the answer an example of how to do this is given with the first line of his code. The questioner then goes on to ask why, after changing that line, the error still persists [48]. So having a little knowledge about digital hardware is required to create code for the HDL Coder.

As stated before TNO already has one license for the HDL Coder[14]. One of the persons who works with the HDL Coder has been asked a few questions. He is good with the Simulink environment and the HDL Coder did not need much getting used to, the software points you in the right directions, but some things are not easily found. Support is definitely needed there. The hardest part is making a black-box. His team was unable to do this without support. With everything up and running it is a good tool. For his design an FPGA communicates with a CPU. There are some timing issues in the communication between the two. They have not yet come to solve these issues, so how easy this is cannot yet be said. He would like to see better documentation, meaning tutorials and more insight into the inner workings of the system. Not just a click-here-than-there description of how to use the software. He thinks the HDL Coder is a powerful tool but the documentation will

[14] Release R2015a.

have to get better. Like the documentation they are used to from other toolboxes from The MathWorks.

The hardware will run on multiple clock sources (more on this in paragraph 8.7). The HDL Coder can be used to create designs with multiple clock sources [44]. If this is possible it should be able to incorporate HDL code that has multiple clocks.

6.5 Needed knowledge

Taking the previous paragraphs of chapter 6 into consideration something can be said about the extent of knowledge a user of the HDL Coder needs. Quantifying knowledge is a difficult thing to do. When using the HDL Coder the user of the software has a lot of things he must either be aware of, or must know about. These things (subjects) are listed in table 1. For each subject is determined whether the user must be aware of, know about or needs detailed knowledge about the subject. Per subject an indication of the extent of knowledge is given. Since the HDL Coder is not used this list is only an indication and may not be complete or the extent of knowledge may have been wrongly estimated. For each subject is described why and when this knowledge is required.

Subject	Aware of	Know about	Detailed knowledge
Fixed point/discrete mathematics		✓	
Parallelism		✓	
Using LUTs for functions (e.g. exp, sin, log)		✓	
Using an approximation formula for functions (e.g. exp, sin, log)	✓		
Hardware multipliers and dividers		✓	
Trade-offs between area and speed		✓	
Solving timing issues			✓
FPGA Synthesis tools		✓	
Delta delays in hardware simulation		✓	
HDL (e.g. VHDL)	✓		

Table 1. Knowledge required when using the HDL Coder.

Fixed point/discrete mathematics
An FPGA is a digital device and thus definition performs discrete calculations. To decrease the size the design takes on the FPGA as well as to increase the speed floating point calculations should be avoided. If decimals are needed a fixed point notation should be used. Floating point can be done, but reduces speed and increases area. Knowing the details is not really necessary as the HDL Coder can help with the conversion. On the other hand having heard about the existence of fixed-point arithmetic is not quite good enough. The user must be able to make the conversion from floating-point to fixed-point and must understand why this is needed.

Parallelism
An FPGA can perform multiple calculations at the same time. Having heard about this should be enough, but when the generated HDL is viewed[15] it is important to know that the statements in the code are all done at the same time. If the code is not looked at, it is something to be aware of.

[15] Viewing the HDL is not required.

Using LUTs for functions (e.g. exp, sin, log)
As discussed before in paragraph 6.4, a number of functions must be implemented with a LUT. Knowledge about how to perform this conversion is required.

Using an approximation formula for functions (e.g. exp, sin, log)
Being aware of the option to not use a LUT, but a formula that approximates the value is a good thing. Using such a formula may result in a slower speed (or longer pipeline), but this may be an option if the area becomes a problem.

Hardware multipliers and dividers
Using hardware multipliers is not really a problem, but hardware dividers can pose some problems. Dividers are usually slow and big. Knowing that a multiplication or division by a power of two is equal to a shift operation can come in handy when the design does not fit or is not fast enough.

Trade-offs between area and speed
With most designs there is a trade-off between the speed of the design[16] and the area it takes. Knowing the consequences of the design choices regarding the speed and area is required as it is easier to adapt the design in the earlier stages.

Solving timing issues
The HDL Coder does nothing to help the design meet the timing requirements. The MathWorks even mentions this in the design flow: "Performing synthesis and timing analysis through integration with Xilinx ISE and Altera Quartus II" [42]. The timing analysis is done in the synthesis tools. Thus the designer needs to be able to adapt the design to meet these requirements. This requires detailed knowledge about timing. A part of the HDL Coder workflow is "Back annotating the Simulink model with critical path timing". This may help in solving the timing issues, but they still will have to be resolved by the user.

FPGA Synthesis tools
As the synthesis of the HDL is done using third-party tools, it is important to know how to use these tools. But not all features are needed as the HDL is "written" at another point. Knowing how to set constraints and pin locations however is required. Performing timing analysis usually requires a better understanding of the tool than just knowing where to click to get the information, but only knowing that should be enough on non-demanding designs.

Delta delays in hardware simulation
When performing hardware (co)simulation in software such as Mentor Graphics' Modelsim or Xilinx' ISim, delta delays are used to simulate the parallelism. The simulator runs on a CPU which performs only sequential statements and thus a workaround must be created to simulate the parallelism. This may cause the simulation to not match the real behaviour. The designer must know about these.

HDL (e.g. VHDL)
Knowing an HDL is not really necessary, but it might be useful when debugging or when solving timing issues. Knowing about this intermediate step is required because the synthesis tools expect these, and a number of synthesis tools cannot use multiple languages at once[17], so the language must be known.

[16] This is not the speed of the design process, but the physical design.
[17] Most can, but depending on a licence the capabilities are limited.

7 Camera-interface for an FPGA

In order to connect a camera to a control system running on an FPGA, the FPGA needs to have a camera-input. This can be realized by connecting an expansion card to the FPGA board, or by using an FPGA board with the necessary connectors already on it. A number of these boards and cards have an IP core available as well. This means the hardware-implementation does not need to be developed in-house. The selection of these cards is based upon research results that can be found in appendix II. The selection process and the final results are elaborated here.

7.1 CoaXPress

The images can be transferred from a camera onto the FPGA using a number of protocols. The preferred protocol is the CoaXPress protocol. This protocol has been chosen because it allows for long cable lengths, high speeds and can power the camera via the data cables. Another reason to choose this protocol was the preferred camera for the system. This camera is preferred because it is already at hand and might be used for similar systems. This camera uses the CoaXPress protocol.

7.2 Criteria

An analysis has been done in order to determine which board to use. This has been done based on a number of criteria. These criteria are primarily determined by the camera and the control system. Secondary to this are the costs and the delivery time. The comparison was done by placing the specifications of each card in a table and checking if each spec met the requirements. The table can be found in appendix II. The criteria are listed below.

- The system requires a PCIe (peripheral component interconnect express) connection in order to perform part of the processing on a computer. Cards that did not meet this specification were dropped from the comparison. The expansion cards were kept as an option as they can be connected to a board with a PCIe connector[18].
- The camera's outputs form an important restriction. The target camera has four outputs. The card must therefore have at least four inputs.
- An important point for the expansion cards is the connector they use to connect to the FPGA board. If this is an HSMC (high speed mezzanine connector) connector it will be almost impossible to connect this to a Xilinx board, while an FMC (FPGA Mezzanine Card) connector fits almost only Xilinx[19]. For Altera boards this is just the other way round. One company (KAYA) produces connector converters which convert both to and from HSMC and FMC. By using one of these, the connector is no longer a restriction.

7.3 Comparison

During the selection of the boards the main concerns were the criteria as given in paragraph 7.2. After the first selection rounds[20] it became apparent that only one company can deliver the devices at the required level, namely KAYA instruments. One other company, Techway can deliver to the required specs as well, but they use a card that is being made by KAYA, or at least that is how it looks from the

[18] E.g. The Xilinx Spartan-6 FPGA Industrial Video Processing Kit which is already at hand.
[19] For the OpenCL part of the "High Performance Real-time Processing developments" project a board was purchased with an Altera Chip and a FMC connector.
[20] In each round one or two criteria are looked at and based upon those the boards pass or fail.

images available. It is difficult to get that information from the datasheet. The complete comparison and actions taken afterwards can be found in appendix II.

After the initial comparison the most promising companies were contacted. This contact suggested they all are unable to deliver the products they say they sell and no board or card has been ordered.

7.4 Availability of IP

Some companies indicated they sell IP with the board. When asked about delivery dates they gave little to no information and when further questions were asked they could not provide a satisfactory answer. Sensor to image indicated they could deliver the IP and the board needed, but the pricing they provided, was complicated and ended up being way outside the planned price range with the costs starting at €40,000.00 with additional costs for support and hardware. So although they have the IP it is too pricy.

7.5 CameraLink

Because the delivery times of the CoaXPress options are relatively long or could not be given nothing was ordered. The project needed to work around this limitation[21]. This situation had been foreseen and before the internship started a CameraLink card was bought[22]. This served as a back-up to prevent the project from stalling. All further actions have been based upon this board. For those who are unfamiliar with the protocol an explanation of the CameraLink protocol can be found in appendix III. Paragraph 8.1 gives a short description of the protocol.

[21] Having no hardware to work with.
[22] No CameraLink IP was bought.

8 Hardware camera-interface

The CameraLink module is designed for, and developed on, a Xilinx Spartan 6 LX150T development board. Connected to this board is an Alpha Data expansion card[23] with the CameraLink electronics. This board connects to a camera which gives the inputs on the design. The outputs of the camera-interface are the pixels and indicators from the camera. Depending on the calculations this can be separate pixels or it can be one image. As the pixels can be accumulated to form the image the module will only output the pixels. Another module may be written to make the image.

A manual for the digital hardware for the interface and its instantiation can be found in appendix IV. This manual gives a quick explanation of the interface and the way the module works. This chapter also gives an introduction, but explains a couple of those items in more detail as well. The idea behind the design is also discussed.

The HDL language used is VHDL as the designer is already sufficiently proficient in that language. The other persons directly involved are also able to read that language.

8.1 Overview of the protocol

This paragraph will give a short overview of the protocol. A more elaborate overview is given in appendix III. The CameraLink protocol sends data over at least one of three links (or lanes) to the receiver. Per link four bits are transmitted in parallel. The data is sent in "packets" of 28 serialized bits. Per link a clock is sent as well. This clock is used to deserialize the packets. The frequency of this clock is equal to the frequency at which the packets are sent. Per packet three bits are used for image synchronization, one bit is a spare bit and the other 24 bits are used to transfer the image. The bits are permutated before transmission. The reasons for these permutations are unclear, although some speculate it has something to do with lowering crosstalk or transmission power [35].

Figure 8 on page 23 shows the relation between the clock and the bits as well as the position of the bits after the permutations have been undone. A shift register is used[24] to read the bits. So the third bit in the shift register is the first bit received on wire pair[25] three and the 23rd bit in the data when the permutations have been undone. That bit is the spare bit. Depending on which of the three links they arrive on the data bits map to different ports. All ports are listed in the bit position and are separated with a '/'. Thus the eleventh reordered bit maps to the seventh bit of either port B, E or H. To which port it maps depend on the link it is received on.

The speed of the packet transmission ranges from 40 to 85 MHz. Not all data-bits are always used. When sending one pixel per packet the maximum bit-depth is 24 bits[26]. There are cameras that send pixels just eight bits deep. This means a few bits remain unused. The protocol also describes how to send multiple pixels over one lane. This means, with a pixel depth of eight bits three pixels can be sent. The standard does not specify where these pixels should come from. The pixels can come from the same vertical line, the same horizontal line or any other way the camera designers can think of[27].

[23] The FMC-CAMERALINK card.
[24] In the deserializer.
[25] The protocol uses LVDS (low voltage differential signalling).
[26] The protocol specifies this as one pixel with 8-bit deep RGB values.
[27] Most manuals list the way the manufacturer reads the pixels.

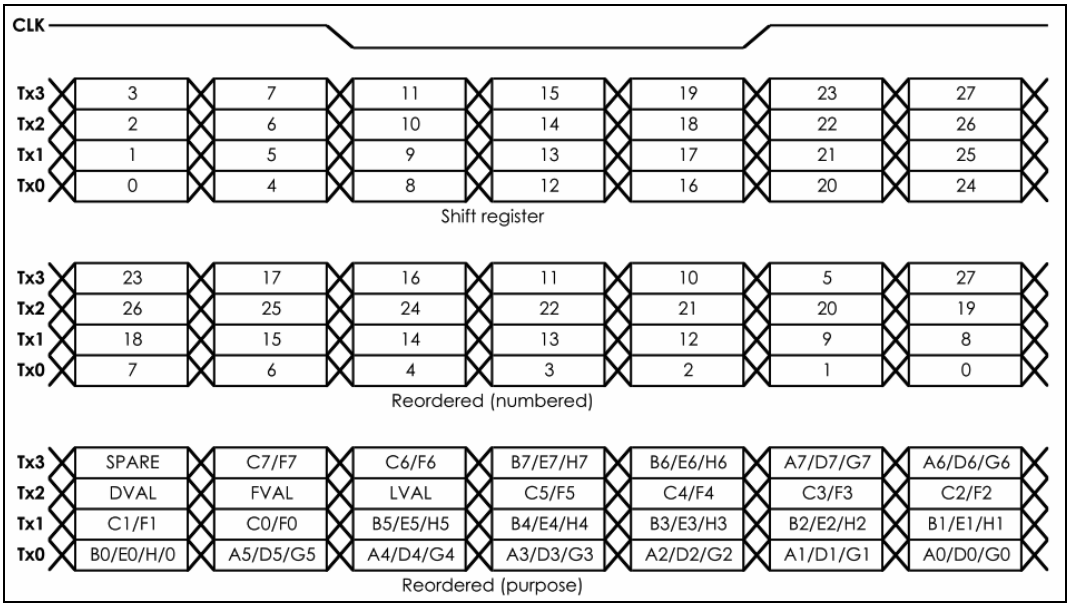


Figure 8. Bit reception and reordering [2].

8.2 Overview of the hierarchy of the CameraLink interface

The module is build from a number of submodules. Each link has its own CameraLink_link submodule. This submodule handles the reception of one link and outputs the pixel and the flags of that link. In order to output the flags in a higher hierarchy some glue logic is needed as the status flags must be merged into one value. This is done by "ANDing" the flags so all the flags must be active in order for the output flag to become active.

Figure 9 shows an abstract block diagram of the CameraLink module. Figure 10 on page 24 shows the hierarchy of the top-level module down to the lower levels. For clarity, the system clock and resets are not displayed. The bus widths indicated with n_x depend on the chosen CameraLink configuration. There are twelve[28] configurations possible. To accommodate all these variations with a single design the description must be able to adapt to the configuration in use. This means the bus sizes are not always the same. The number of implemented CameraLink_link modules is also dependant on the configuration. This paragraph will also give a description of the components of the module.

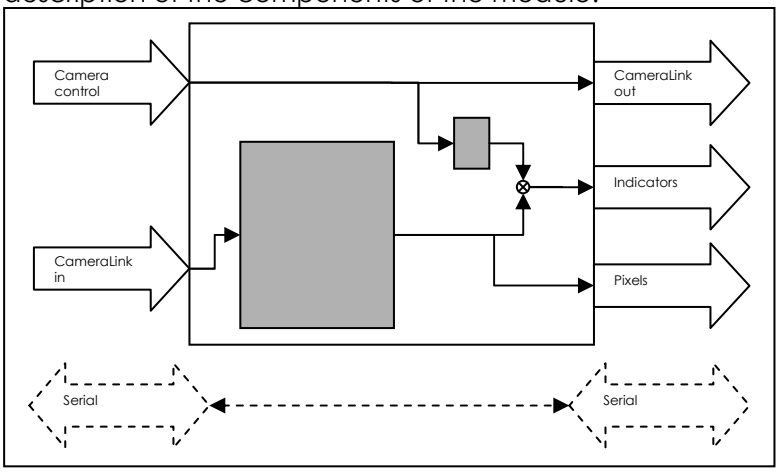


Figure 9. An abstract overview of the CameraLink module.

[28] These are the configurations specified in the standard. Some common used, non-standard configurations are not implemented.

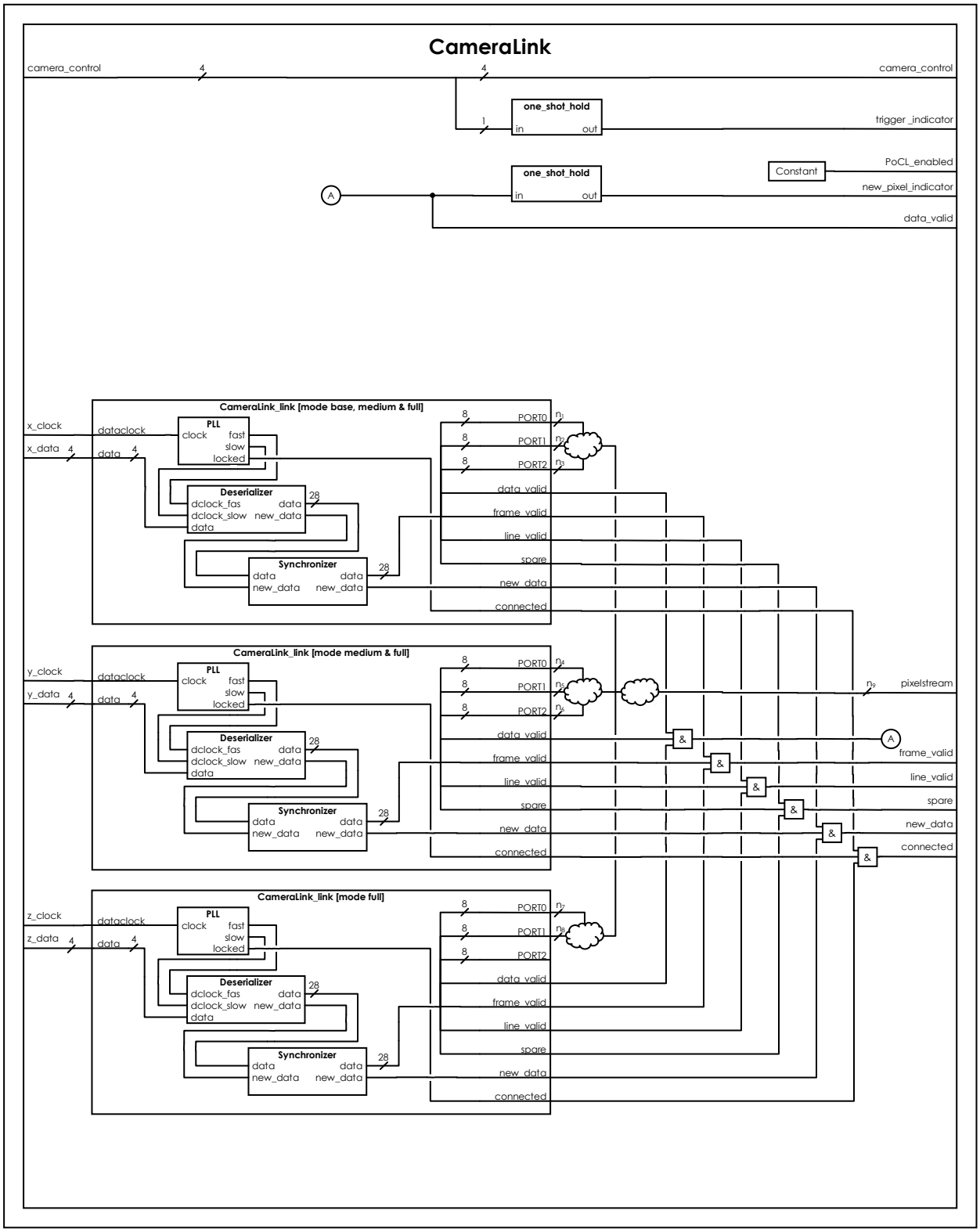


Figure 10. A hierarchical overview of the CameraLink module. (Clocks and resets are left out.)

8.2.1 One-shot hold

Some of the status signals are kept high for a longer period so they can be made visible to the user[29]. This is realized by one-shot hold modules. They trigger on a rising edge of a signal and become inactive after a preset amount of time after the falling edge of that signal has occurred. This can be seen in the timing diagram in figure 11 below.

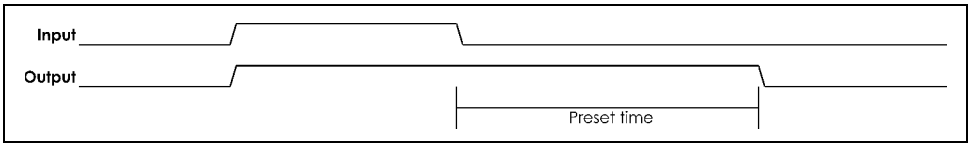


Figure 11. Timing diagram of the one_shot_hold module.

8.2.2 CameraLink_link

This module handles the reception of one link. Using a phased locked loop (PLL), which will be discussed in paragraph 8.2.3, the slow data clock is multiplied to generate a clock seven times faster. This clock is used to read the data from the data lines. (Please see figure 8 on page 23). This readout is done by the deserializer in the module. The data and status signals from the deserializer are then synchronized to the system clock. Paragraph 8.7 goes into more detail on how and why this is done.

8.2.3 PLL

The frequency of the data clock is seven times slower than the data rate. To read the data this clock is multiplied by means of a PLL. The Xilinx PLL IP is used for tests and on compilation an error was given regarding the VCO (voltage controlled oscillator). The VCO can only handle frequencies between 400 MHz and 1080 MHz[30]. So another PLL was created that allowed for slower speed downto 280 MHz (a 40 MHz data clock results in a PLL-ed speed of 280 MHz). This PLL however was unable to drive the deserializer because it had no locked flag[31] and missed a few other signals. Xilinx did not provide good documentation and on the Xilinx forum the solution to the problem was recreating the components [40]. This solution did not work. And another deserializer will have to be used on those lower frequencies.

The used Xilinx PLLs have buffers on their outputs while the components they drive do not need buffers. For that reason these buffers have been removed.

8.2.4 Deserializer

The deserializer used is either a Xilinx IP using the on-board deserializer or a HDL description created during this project. The latter has been created to better understand how the CameraLink protocol works and became useful when dealing with the lower frequencies. As stated before, the slower PLL lacks some of the signals needed by the deserializer from Xilinx. Thus the custom made one had to be used. Later it was found that the Xilinx deserializer worked fine when "lied to" (please see footnote 30).

[29] The visualisation is done via LEDs on the CameraLink card, those LEDs are not a part of the protocol.

[30] In practise however the Xilinx deserializer combined with the first PLL works fine as long as it is set up for 60 MHz.

[31] A signal indicating the PLL is locked to the input clock.

8.2.5 Reset

In order to assure correct behaviour after the reset a reset synchronizer should be used. Paragraph 8.8 will discuss this further. The reset should be connected to the clock domain it is used in. In the current version of the design this is not the case. Developing a good reset synchronizer takes some time and is partially dependant on the compiler and synthesizer. During the development of the modules a quickly build reset synchronizer was created, but it was not thoroughly tested and verified. Such a reset module will have to be built in into the CameraLink module. By taking the input from the CameraLink reset synchronizer that module always gets valid[32] data when its reset is de-asserted because the reset is released later in the CameraLink_link modules.

8.3 CameraLink configuration

The CameraLink protocol has various configurations. In order to keep the size of the CameraLink module down, thereby making more room for the control and image processing systems it is not possible to change the configuration during run-time. In practise only one configuration will be chosen and this will not be changed, so such functionality was deemed unnecessary. Which configuration to choose depends on the application, the camera, etc. The configuration will be set by means of generics. These generics are used to choose the mode, the number of channels and the number of bits per channel. It is possible to set an invalid configuration. The HDL is written to find these incompatibilities and report them back to the developer. The valid code in code fragment 1 describes such an invalid configuration and will trigger the error: "*When using CameraLink in mode base with two channels, the number of bits must be either 8, 10, 12, 14, 16 or 24. Now used: 20.*" and the compilation will stop.

```
CL_0 : CameraLink generic map(
   System_clock_frequency  => 10000000,
   speed_CameraLink        => 80,
   CameraLink_mode         => BASE,
   CameraLink_numchannels  => 2,
   CameraLink_numbits      => 20,
   enable_PoCL             => false,
   camera_framerate        => 25,
   trigger_index           => 0,
   chip_family             => "Xilinx_Spartan_6"
)
```

Code fragment 1. Invalid configuration of the CameraLink module.

These different configurations can cause some ports to be left unconnected. These unused ports give a small problem. They trigger warnings in the compilers telling the developer the inputs of a component are not connected to any logic in that component or that the outputs are not driven by any logic. These warnings can be ignored because these ports are not needed in the chosen configuration.

The CameraLink protocol knows eight ports (A-H), two of the links have three ports one has two. To be able to reuse the CameraLink_link module, the CameraLink module has nine ports (A trough to I). The last one is never used. As it will not be connected to any logic, the compiler will optimize the design and will not

[32] Non-meta-stable, the valid flags from the CameraLink protocol have nothing to do with this.

synthesise the logic for this port. As with the other unused ports this triggers warnings that can be ignored.

8.3.1 Delays

The delays of the module (in clock cycles) depend on the configuration used. The clock cycle delay must be entered when using a black-box in the HDL Coder. When changing the configuration, the delays in the HDL Coder also need to be changed. The difference in delays is caused by different buffer sizes.

8.4 Generalizing the design

The design of the CameraLink interface has been kept as generic as possible to accommodate a wide range of chips and tools as well as to accommodate all twelve[33] CameraLink configurations. This is done by using a widely used VHDL standard and selecting hardware descriptions depending on the chip in use. This paragraph describes why and how this is done as well as some other options to do it and why those methods were not used.

8.4.1 VHDL standard

To be able to use as many tools as possible an older version of the VHDL standard is used. While VHDL-2008 will give a lot more flexibility in the code, the number of compilers that comply with this standard is quite low. Older chips are not always supported in the new tools[34], so the choice for the older VHDL-1993 standard was logical[35].

Some of the flexibility of the VHDL-2008 standard can be achieved by using libraries. The ieee.std_logic_misc is such a library. It is an expansion of the ieee.std_logic_1164, providing, among others, logic operators for std_logic_vectors. For the testbenches such steps are not always possible and for that reason the testbench has a constant where the designer will have to enter the VHDL standard. If the standard[36] is VHDL-2008 more graceful functions are available. To stop the simulation in VHDL-1993 a failure is passed to the simulator, which is basically like crashing a car into a wall in order for it to stop. VHDL-2008 on the other hand has "breaks"[37]. It has a function to stop the simulation after which the simulation can still be continued if needs be. There is a trade-off here between supporting a lot of tools and using a language that enables more flexibility and readability. (Figure 12 illustrates this.)

[33] Only the mode, number of pixels and number of bits per pixel are meant.
[34] The newer tools have VHDL-2008 support.
[35] The development environment only supports VHDL standards up to VHDL 1993.
[36] This is the standard supported by the simulator.
[37] Not to be confused with the break statement some other languages have. This is a function called stop.

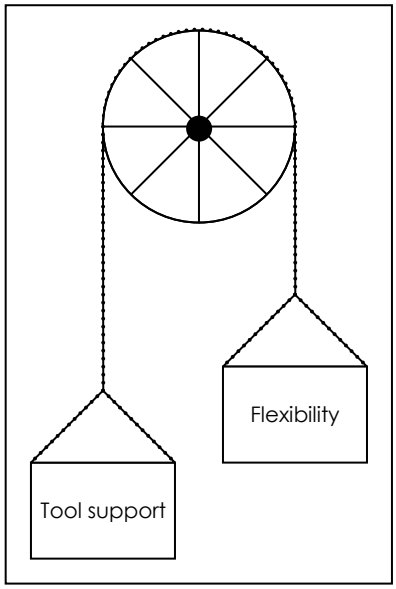


Figure 12. To bring up the tool support the flexibility has to come down and v.v.

8.4.2 Conditional implementation of chip-specific hardware

For a few elements of the design the best option is to use chip-specific hardware. A hardware multiplier for example. FPGA manufacturers supply a VHDL component of this multiplier to the developers. The interface to the multiplier varies per manufacturer, and also per chip. The CameraLink module utilizes a SERDES (serializer/deserializer) and some (IO)buffers. The latter are sometimes inserted by the synthesizer and therefore not always described in the code.

To keep the source code maintainable and reusable it would be a good thing if the code did not have to change too much when using different chips. To prevent the creation of multiple source-code files, and all the version-management troubles that come with it[38], the module has a generic input named "chip_family". By means of this generic the developer can, without having to know the details, select the required instantiation. This generic is used by a number of parts of the module. In this way modules, such as the PLL, can be conditionally implemented (see code fragment 2).

```
if_spartan_6 : if ( chip_family = "Xilinx_Spartan_6" )
generate
    --component instantiation specific for the
    --Spartan 6 chip from Xilinx
end generate;
```

Code fragment 2. Example of chip-specific hardware selection.

The component declaration still needs to be done, and cannot be done conditionally. This means all possible component declarations are listed, but because they will not be instantiated this will not give problems[39]. The libraries in which some of those components are defined however cannot be included. Doing so will give problems as those libraries are not always

[38] E.g. changes that are made to one file that not made in other versions.

[39] Tested with Xilinx ISE14.7, Altera Quartus 15.0, Xilinx ISim 14.7 & Mentor Graphics Modelsim Altera Starter Edition 10.3d.

available. Therefore the CameraLink library has a workaround. The way this has been done is to create aliases in the CameraLink library, which will be discussed further in paragraph 8.5, and using these aliases in all other source files.

Selecting hardware depending on the chip poses problems to the continuity of the behaviour. Not all SERDES' have identical delays which might change the timing of the module. One way to deal with this is to choose a ridiculously long delay so all SERDES' will comply. This is however not good when the whole purpose of using an FPGA is fast processing. Another way is to define a minimum and a maximum. But what if a chip will exceed the maximum given value? There is a trade-off between being generic and being constant. (This is illustrated in figure 13.) Meaning the more generic the behaviour, the more chips can be supported, but this comes at the cost of the timing not being constant throughout all chip families. This is greatly simplified because some SERDES' take the same time to deserialize while others may even be faster. And since the choice for the chip is made during the design process the system engineer can account for those differences. The behaviour of the CameraLink module does not change when data is delayed, as long as all SERDES' replicate the behaviour of the SERDES in the way the CameraLink_link module sees it i.e. IO timing, signal relations and data width.

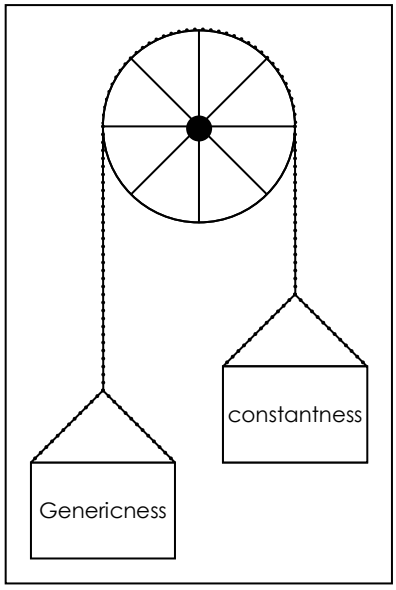


Figure 13. To bring up the "genericness" the "constantness" has to come down and v.v.

8.4.3 Generics or global constants

Based on the constants described earlier in this chapter the hardware is generated. Those constants are passed to the entities as generics. The option of using generics has been chosen over global constants to prevent possible duplicate names and to have traceability on the origin as well as having different values in simulation and in synthesis.

One of the generics is the frequency of the system clock. This is named system_clock_frequency. It could be another module in a larger design uses this name as well. When using global constants it is rather difficult to keep track of which one is in scope at which point. If the values differ it can cause problems the designer does not expect. Also, when using generics, it is easier to keep track of the values passed to a specific entity.

8.4.4 C-Preprocessor

Since parts of the code are selected depending on some of the constants, the code has a lot of conditional statements. VHDL-1993 has an if-generate statement but no elsif-generate or else-generate[40]. The language also does not have a #if as in C-type languages. The if-generate comes close, but is not quite the same as it can only apply to design units, not to declarations and library inclusions. One way to work around this is to use a C-preprocessor in the toolchain. Before the first tool in the toolchain does anything with the VHDL code the code is run through the preprocessor. In this way the #if can be used, or better said borrowed. The preprocessor will take out all parts that do not satisfy the conditions, but it requires a preprocessor and knowledge on how to integrate the tool into the used toolchain. This can be complex and is therefore not used.

8.5 CameraLink library

The CameraLink module and its submodules have a few settings which require a number of special types. These types are gathered in a library. Most of the types are not so special and are not discussed. One of these types however is and is discussed here. Also some of the chip-specific components require a library. Which library this is depends on the chip. How this is handled is also discussed here.

The CameraLink module uses a buffer to enable the use of a low clock speed (more on that later in paragraph 8.7). The selection of the number of buffers is done by means of generics. This generic is a custom type. This type is a bounded integer supplemented with an auto value. VHDL does not allow for such a mixed type. To create a workaround a constant is created (code fragment 3). This constant has a value which cannot be used and each time the type value has to be evaluated. To enable this evaluation a function to do that has also been created(code fragment 4). Other types are used for ease of reading. These types are multidimensional arrays which are mostly used for synchronization on clock domain crossings.

```
subtype CameraLinkNumberOfBuffers is integer range 0 to 2;
constant AUTO : integer := 0;
```

Code fragment 3. Declaration of the custom type CameraLinkNumberOfBuffers.

```
number_of_buffers(value,data clock speed, system clock speed);
```

Code fragment 4. Example of the evaluation function for the CameraLinkNumberOfBuffers type.

Another custom type is the CameraLinkMode. This type is used to select the CameraLink mode. The type is a rather straight-forward type declaration with the values being BASE, MEDIUM and FULL. The values can be compared where BASE<MEDIUM<FULL. This allows for easy selection of the required CameraLink_link modules.

Some chip-specific components have their declarations etc. defined in a library. For the Xilinx Spartan 6[41] this is the UNISIM library. For Altera Cyclone IV[41] this is the altera_mf library. VHDL itself does not allow for conditional inclusion of libraries. Therefore the CameraLink library has aliases for these libraries hard-coded in the package declaration. All the design units use the alias and if another library is

[40] VHDL-2008 does have those as well as a case-generate.
[41] It is possible that other chips use this one as well.

needed the designer will have to change these by hand, but only in one file not in all. How this is done can be seen in code fragments 5 and 6 below.

```
--Comment next line if not using Xilinx Spartan 6 or compatible
hardware
--and include the correct libraries
library UNIMACRO;
library UNISIM;

package CameraLink_lib is
--Comment next lines if not using Xilinx Spartan 6 or compatible
hardware and make new aliases for the ones needed.
alias chip_specific_components is UNISIM.vcomponents;
alias chip_specific_components_sim is UNIMACRO.vcomponents;
```

Code fragment 5. Definition of the aliases for the libraries.

```
library work
use CameraLink_lib.all;
use CameraLink_lib.chip_specific_components.all;
```

Code fragment 6. Usage of the aliases for the libraries.

8.6 Design constraints

Some of the design constrains are given in the format of the used tools[42]. Some of those also depend on the chip used. The designer is expected to set these constraints in the used toolchain. The constraints are given in the list below.

- System clock frequency and duty-cycle
- Data clock frequencies and duty-cycles (is 4:3)
- Synchronizer registers have asynchronous input so prevent inferring shift registers [31].

8.7 Crossing clock domains

The CameraLink_link submodules have hardware running on two clocks. Meaning different components running on different clock sources. Because of the different clocks[43] the data can become metastable and data might be missed. This chapter will discuss the steps taken to prevent this. Also a short explanation of the problems on clock domain crossing is given in appendix V.

8.7.1 Clock domain situation

The data for each link is read using the multiplied data clock. This frequency ranges anywhere from 280 MHz to 595 MHz. The system processes this data on the system clock, which has another frequency and/or phase. The frequency range for the system clock is wide, as this is not standardized. The minimum system clock frequency is given by the design and is the same as the slowest CameraLink clock speed: 40 MHz (please see formula 4 in paragraph 8.7.3). The clock domains are crossed in the CameraLink_link

[42] Altera uses QSF files, while Xilinx uses UCF. Both have different keywords and a different syntax.

[43] They differ in speed and phase.

module. Figure 14 below shows that module with the domain of the data clock marked in grey[44].

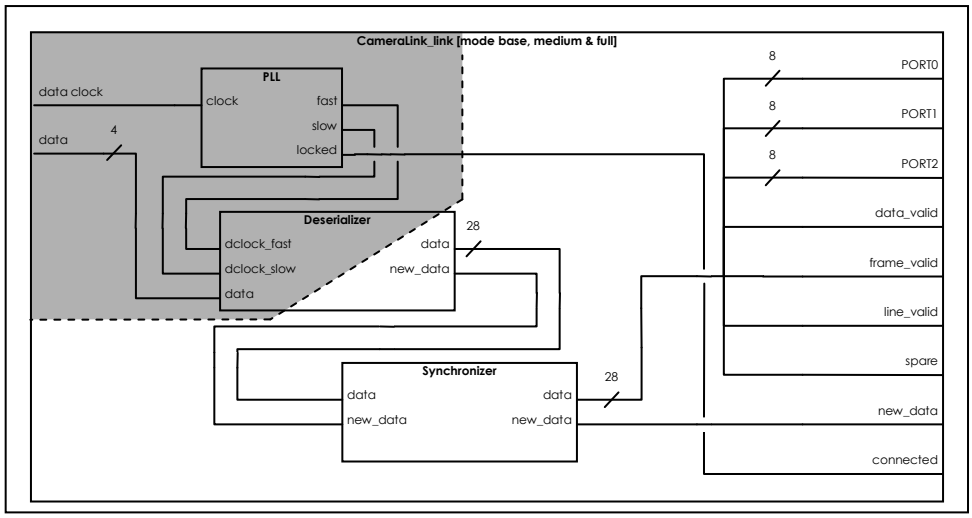


Figure 14. Clock domains, grey runs on the (multiplied) data clock, the white parts run on the system clock.

8.7.2 Data synchronization and deglitching

The data read in the CameraLink_link module is output all at the same time. So the flags, indicators and data are all grouped together. The availability of the new data is signalled by a level change, on which later more. This change is detected in the domain of the system clock and a pulse indicating the availability of new data is created. All the data is synchronised trough a couple of registers to prevent metastability issues. Thus the data is stable, but might not be correct. It could have been recovered to an incorrect value.

To prevent this, a "deglitcher" has been designed. A glitch is assumed to be only caused by reading a metastable value at the input of the synchroniser. From this can be deduced that that data is incorrect only during one clock cycle, assuming the ratio between the rise time and the fall time of the data and the period of the system clock is less then one[45] (please see formula 1). That means to be sure about the correctness of the data the first data has to be dropped and the second-cycle of that data needs to be sent on. Detecting an edge on the new data indicator takes this extra cycle.

$$\frac{\max\{t_{rise}, t_{fall}\}}{T_{system\ clock}} < 1$$

Formula 1. Statement that needs to be true in order for the deglitcher to work.

The data is synchronized through three registers because at least two are needed to prevent metastability. The third is added to reduce the mean time between faults (MTBF). This could be calculated by calculating the MTBF per flip-flop (please see formula 2) and than multiplying those with each other to get the MTBF for the synchronizer, but neither Xilinx nor Altera has these numbers freely available. Calculating the MTBF for the chip on which the design was developed does not say anything about the MTBF on other

[44] Only after the PLL is the clock multiplied.
[45] Ignoring the setup-and-hold times for simplicity, these cannot be ignored in reality. They are too significant to allow that.

chips. So the availability does not really matter. Just to be on the safe side three flip-flops are used.

$$\text{MTBF} = \frac{1}{f_{clk} \cdot f_{data} \cdot T_0} \cdot e^{\frac{t_r}{T}} \text{[sec]}$$

Formula 2. Formula for the MTBF [57].

Shift registers generally have an MTBF that is not good for synchronizing. After compilation it was checked whether the compiler had inferred shift registers for the synchronizer and it was found this was not the case. UDC primitives (D-flip-flops) were used [31][6][46].

8.7.3 Overcoming data loss

To verify the correct steps have been taken to ensure correct "clock-crossing" the flowchart from figure 15 (on page 34) is used. Now that the data is stable and valid, it still might be some data is lost. If the system clock is faster than the data clock, data loss will not occur. When the situation is reversed data loss is an issue. Solutions using a first-in-first-out (FIFO) register, handshaking etc. cannot be used because there is no guarantee of a break in the transmission, or at least a break that is big enough to catch up. So the design requires a system clock with a frequency of at least a frequency equal to the data clock, but it is advised to choose a faster clock to overcome data loss due to jitter in the clocks. The minimum system frequency can be calculated using formula 3. Where the constant factor of two is derived from the data rate dividing property of the "deglitcher" and the number of cycles needed to detect an edge on the new data signal. They both take two cycles.

$$f_{\text{system clock}} \geq f_{\text{data clock}} \cdot 2$$

Formula 3. Formula for the minimum system clock.

This means the system clock frequency for a 40 MHz data clock must be at least 80 MHz. To bring this factor down the data from the deserializer is buffered using the multiplied data clock. This creates the possibility to synchronise two sets of data together (please see figure 16). This results in a lower system clock frequency requirement. Formula 4 shows the formula to calculate the system clock frequency with this improved architecture. This means that if the system clock is fixed at 100 MHz and the data clock is again 40 MHz, one buffer is needed. This can be calculated with the adaptation of the formula, which is given in formula 5 with:

$f_{\text{system clock}} \in \mathbb{Q} \wedge f_{\text{system clock}} > 0$

$f_{\text{data clock}} \in \mathbb{Q} \wedge f_{\text{data clock}} > 0$

$\text{Number of buffers} \in \mathbb{N}_1 \wedge \text{number of buffers} \leq 2$

The number of buffers cannot exceed two since there is no need. The buffers are only needed to overcome the cycles lost in the deglitcher and edge detector, nothing more. Although the system clock speed could be lowered in that case, the data would again be lost because the data is not output in groups.

$$f_{\text{system clock}} \geq \frac{f_{\text{data clock}} \cdot 2}{\text{buffer width}}$$

Formula 4. New formula for the minimum system clock.

[46] The compiler used ignores the constraint on these registers.

$$\text{Number of buffers} = \left\lceil \frac{f_{\text{data clock}} \cdot 2}{f_{\text{system clock}}} \right\rceil$$

Formula 5. Formula to calculate the number of buffers.

After reading all this, the idea still might be a little vague. To clarify this all the signals are displayed in the timing diagram in figure 16.

It could also be thought of as mining carts arriving with the frequency of the data clock. They must go through a tunnel (the synchronizer and deglitcher) before ending up at the place to unload. Instead of sending each on their own, two carts are coupled together before entering the tunnel and uncoupled when exiting. Each of them then making its own way to the unloading site.

Another way of preventing data loss is to use the edge to signal data. This is done with the new data signal. A single pulse could be missed, a level change cannot, provided the value keeps its value for long enough. For the new data signal this is the case. The downside of this solution is a delay on the receiving end as the edge will have to be detected.

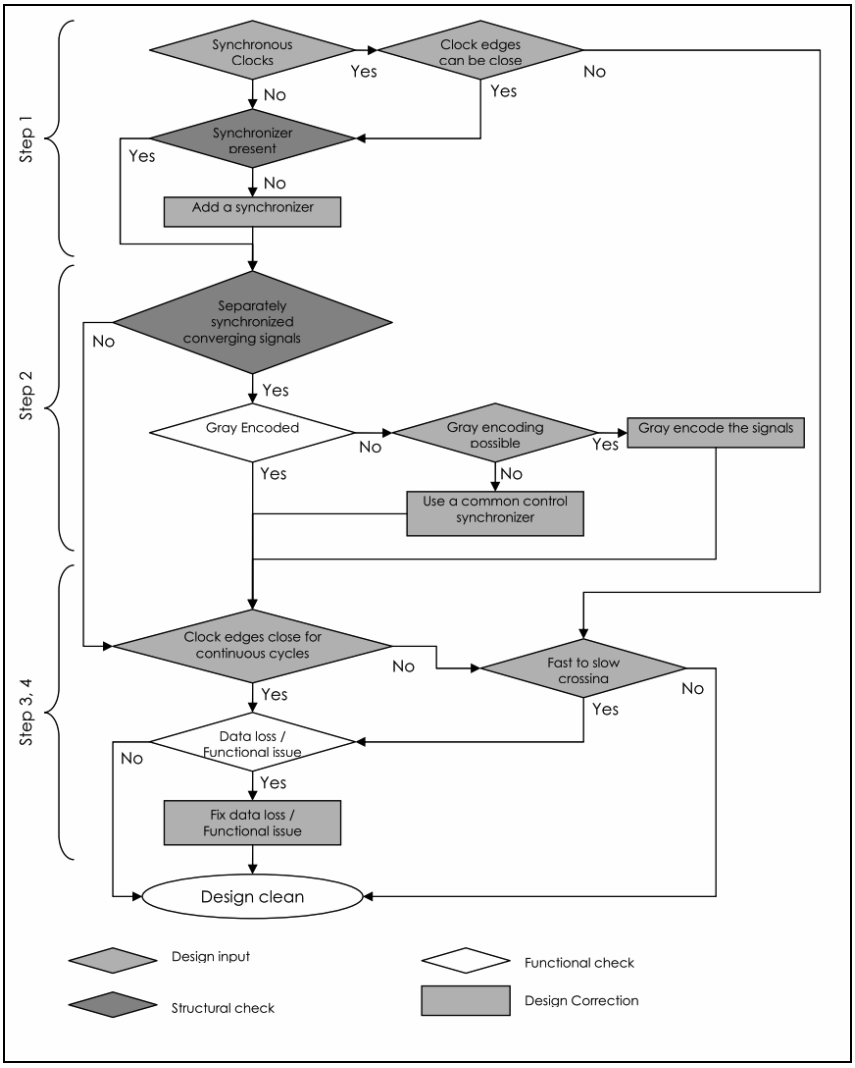


Figure 15. Verification methodology for the clock crossing [28].

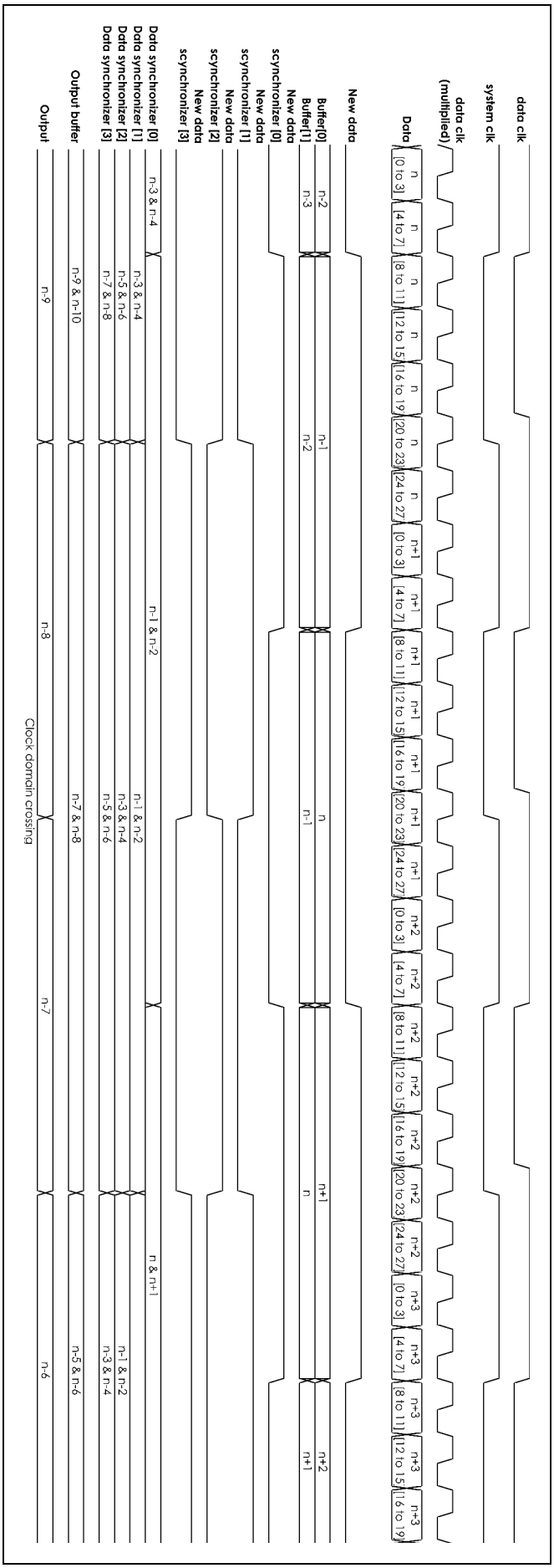


Figure 16. Timing diagram for the synchronizer and deglitcher for the clock domain crossing.

8.8 Reset

The reset on all the components is asynchronous, both in asserting and in clearing. During tests a reset synchronization module was used to assert the reset asynchronously and clear the reset synchronously on the system clock. This module is not implemented in the CameraLink module as the module will be part of a larger design. The designer of the larger design is responsible for correct assertion and clearing of the reset. The reset for the part of the design that runs on the data clocks from the CameraLink module is not synchronized either. This will have to be done in a future version of the code as the asynchronous reset is unsafe [7]. Paragraph 8.2.5 describes how to do this.

8.9 Testing - simulation

In order to test the module a testbench has been written. To be able to test all possible (valid) CameraLink configurations this is a very elaborate testbench. Test parameters can be given using constants. These constants are discussed in paragraph 8.9.1. The testbench has multiple processes. Some of these processes control the CameraLink module, while others provide data and some more check the outputs for correct behaviour. A short description of all these processes and their tasks will be given in paragraph 8.9.4. Automatic testing of the different configurations is not possible as the configuration is done on compilation, not during runtime or simulation. It may be done using scripts for the simulator, but those scripts will be tool-specific.

The testbench is designed to encapsulate the CameraLink module (please see figure 17). The testbench has some parts that simulate the behaviour of a camera, while others provide input signals such as a clock and reset. The module outputs data and status. These are read by the testbench and analyzed for their correct behaviour. The testbench notifies the designer if an incorrect value has been detected.

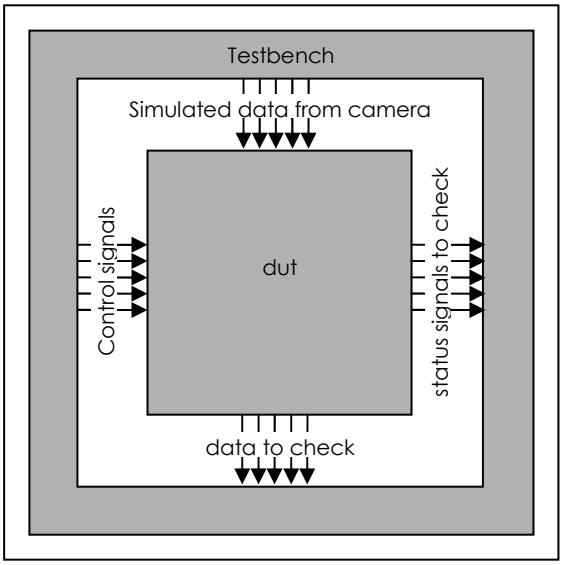


Figure 17. Testbench encapsulating the CameraLink module.

8.9.1 Simulator software

The simulation has been run on Xilinx' ISIM 14.7 . Since it was part of the toolchain with ISE 14.7. Mentor Graphics' Modelsim is more capable in the sense that the user can view the time deltas and the documentation is slightly easier to understand and find. The Modelsim version available is the Altera Starter Edition 10.4b. This means there is no default support for the Xilinx

libraries. For this reason the ISIM simulator has been used for almost all simulations.

One of the modules required a simulation that caused ISIM to stop simulating because the tracing limit was reached. This is a limit build into ISIM to prevent it from using too much memory. In this case it meant the simulation had to be done without logging the signals up to the point of interest, then adding the signals of interest and then continuing the simulation again. Since the window of interest was bigger than the limit allowed Modelsim was used to run this simulation. The dut was slightly changed by removing the Xilinx components as they did not influence the behaviour of the part of the design that needed to be tested. This simulation was run to test the image_storage module. This module will be discussed in paragraph 8.10.3. The testbench for the CameraLink module does not reach the trace limit.

The Xilinx libraries could be added to Modelsim but the documentation on that is vague and requires knowledge about the tools to compile those libraries. Therefore the workaround of removing the Xilinx components was used until a tutorial was found [19]. This tutorial has been followed but the software used to compile the libraries does not work with the Modelsim-Altera starter edition. Which means that version of Modelsim cannot be used to simulate Xilinx components. Which was not unexpected as it is supplied by Altera.

8.9.2 Simulation parameters

To give the testbench the ability to test all possible CameraLink configurations the testbench has several constants to set the simulation parameters. The simulation parameters are listed below:

- The VHDL standard
- Should the simulation fail in order to stop upon completion?
- The CameraLink mode
- The number of channels of the CameraLink
- The number of bits per CameraLink channel
- The frequency of the system clock
- The chip on which the hardware will run
- The speed of the CameraLink connection
- Additional delay on top of the delay for the synchronizer
- Should the testbench use differential signalling
- The index of the trigger signal in the camera control signals
- The time the trigger should be high
- The frame rate of the camera
- Should power over CameraLink (PoCL) be used?
- The number of buffers to use during synchronization
- The skew between data of link x and link y
- The skew between data of link x and link z
- The time between the change in the connected indicator and actually being connected

Some of these parameters have bounded values, others do not. The skew for example can be set to more than a minute, but the module will never be able to get the correct data. The tester has been given the freedom to test what he wants. The manual for the CameraLink module (in appendix IV) explains each of the constants.

8.9.3 FIFO synchronizer for the testbench

The testbench generates data in the domains of the data clocks. This data has to be synchronized to the system clock as that is the clock the CameraLink module runs on. The synchronizer for this part is only for simulation and not synthesizable. The synchronizer comprises of three elements:

- A shift register
- A multiplexer
- A counter

The FIFO-synchronizer has two "operations", reading from the FIFO and writing to the FIFO. Apart from the shift register, a counter is also edited. This counter is used to determine the position to read from. On a write action, given by a rising edge on the rd input, the data on the input is shifted into the shift register and the counter is incremented. On a read action the data on a position, indicated by the counter, is output. Then the counter is decremented and an acknowledgment is given by means of an event on the rdack (read-acknowledge) port. A schematic view of the dataflow is given in figure 18. The module also provides indicators that can be used to block processes on a read request on an empty FIFO or on a write request on a full. These are not used in the testbench of the CameraLink module. The module can overflow when writing data to a full FIFO. The oldest data is then shifted out. This results in the possibility to start shifting in data without the PLL having a lock. This ensures the data can be checked as soon as the PLL is locked and the CameraLink module indicates a connection has been established.

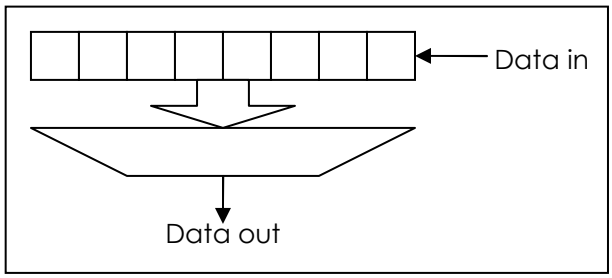


Figure 18. Schematic view of the dataflow in the FIFO synchronizer.

8.9.4 Testbench processes

The testbench has multiple processes. Some of those control the CameraLink module while other processes check the behaviour. A number of these processes need to wait on each other and need to exchange information. Figure 19 (on page 41) shows the tasks these processes perform and the communication between the processes. The sequence diagram has been adapted slightly to allow the communication to be also displayed. The communication is indicated with arrows and an annotation and, in the case of the data generators for the links and the data checker a synchronization bar. This bar indicates a synchronization between all four processes. This chapter will also give a short description of these processes.

If an erroneous value has been detected by any of the processes the process which found the error will report the problem and cause the simulation to fail. Sometimes false errors are given. This is because two processes communicating via the FIFO do not work together properly. They do not block yet, but it may be that implementing the block will solve the problem.

clock generator and constants checker
This process generates a clock with a given period. Before starting with the clock generation this process generates a series of reports[47] that list the simulation parameters. This is done to be able to trace the configuration under test in the simulator log files. This process also tests the PoCL output. It does this at every clock edge. This process does not communicate with the other testbench processes other than that the other processes in the testbench can read the generated clock signal. The process continues until simulation is halted.

system
This process waits on the first rising edge of the system clock and then activates the reset signal and shortly afterward deactivates the reset. Then the process stops.

camera control controller
This process generates trigger pulses with a duty cycle of and frequency determined by the camera_framerate and trigger_high constants.

camera control checker
This process checks if the trigger indicator is correct. The process waits for the first trigger pulse and then starts checking if the indicator stays active. The process continues until the data_checker stops. Then it checks if the trigger indicator keeps high for a long enough, but also not too short, time. Afterwards, if set to do so, this process signals the simulator to stop.

connected checker
This process checks, depending on the CameraLink mode, if one or more links are active. This is done by checking the data clock inputs of the CameraLink module. This process does not communicate with other processes and continues to run until the simulation is stopped. This module does not communicate with other processes.

link X data
Depending on the given skew[48] this process waits for none, one or both of the data generators for link Y and Z. First empty data is transmitted so the PLL can acquire a lock. Then predefined data is transmitted, followed by a short break in the data. After this break empty data is sent in order to give the PLL time to reacquire a lock. Then all possibilities of the valid flags and the spare flag are sent. After this test data is sent but interrupted mid-transmission. Afterward follows a test on sending less links than expected. E.g. sending data over one link while mode full was specified. Then this process stops. This process communicates heavily with the data checker.

[47] With the report or assert keyword to the simulator, this is not a report file.
[48] This skew is given by the constants.

link Y data
Depending on the given skew this process waits for none, one or both of the data generators for link X and Z. First empty data is transmitted so the PLL can acquire a lock. Then predefined data is transmitted, followed by a short break in the data. After this break empty data is sent in order to give the PLL time to reacquire a lock. Then all possibilities of the valid flags and the spare flag are sent. After this test data is sent but interrupted mid-transmission. Afterward follows a test on sending less links than expected. E.g. sending data over one link while mode full was specified. Then this process stops. This process communicates heavily with the data checker.

link Z data
Depending on the given skew this process waits for none, one or both of the data generators for link X and Y. First empty data is transmitted so the PLL can acquire a lock. Then predefined data is transmitted, followed by a short break in the data. After this break empty data is sent in order to give the PLL time to reacquire a lock. Then all possibilities of the valid flags and the spare flag are sent. After this test data is sent but interrupted mid-transmission. Afterward follows a test on sending less links than expected. E.g. sending data over one link while mode full was specified. Then this process stops. This process communicates heavily with the data checker.

data checker
This process checks if the data transfer has been done successfully. This is done by getting the data the data generators send, and then comparing this to the data the CameraLink module has received. Not only the data is checked, the flags are checked as well. This process stops after all the tests in the link processes have run. This process communicates with the link data generators and the camera control checker.
It also reports test progress to the developer.

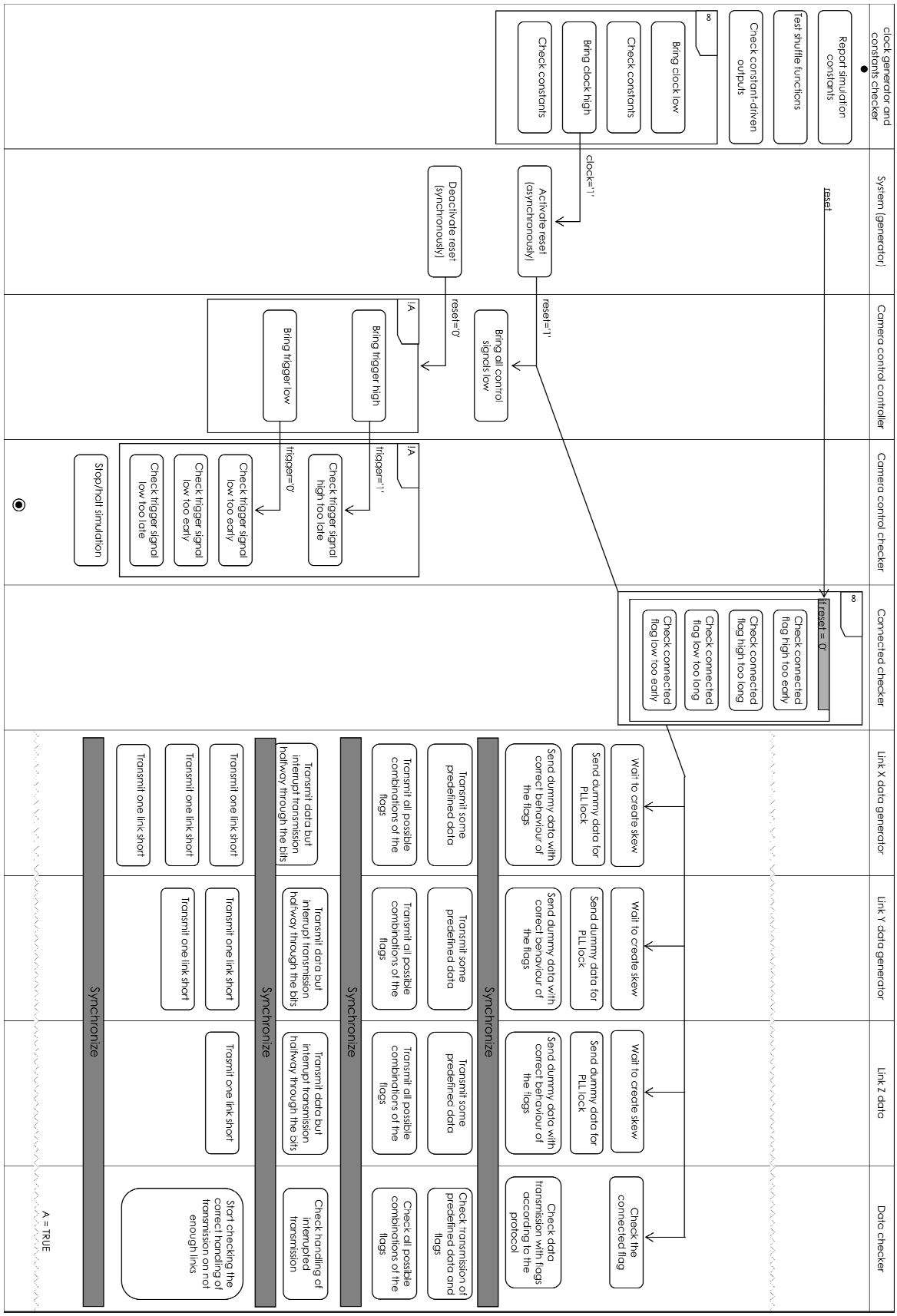


Figure 19. Processes of the testbench and the communication between the processes.

8.9.5 Differences between simulation and reality

The behaviour of some of the IP's from Xilinx in implementation differs from their behaviour in simulation[49].The design was first created and tested with the ISIM simulator and found to be working. Later, when the design was loaded onto the FPGA it was found to be not working and some data had to be delayed for one clock cycle. This could be down to delta delays, which cannot be shown in the simulator used. Another cause may be the description for simulation. If this is not created to simulate the behaviour of the real component there is a difference. This difference in behaviour was found to be at least in the PLL. This problem has been solved by adding a condition on the behaviour of the CameraLink_link module. That module displays different behaviour in simulation, but only when the chip_family is set to "Xilinx_Spartan_6".

At some point the simulation did not work. I.e. The values were not calculated or updated. People on the Xilinx forum suggested initializing the signals [41]. This was only needed for a few signals. Those signals are initialized to 'Z'[50]. This value is chosen because the signals are non-tri-state signals. Thus it is easy to see if any signals remain unused, or at least unwritten.

8.9.6 Simulation libraries

Some checks and conversions have to be executed multiple times. In order to maintain readability and improve ease of writing a simulation library has been created. This library consists of functions that are used to wait on certain values and check the data. The checks on the data are separated from the CameraLink library to prevent the checks not detecting errors because the same erroneous piece of code is used.

As stated in paragraph 8.9.5 the behaviour of some of the Xilinx IP components is different in simulation. To correct this, the destination of the compilation had to be determined. To do this each module was given a generic input with a default value of false. This default value of "false" prevents the necessity to have the value coming from the higher levels. For the same reasons as given for the configuration constants in paragraph 8.3, global constants are not used.

8.9.7 Custom string function

In order to give convenient error messages a function has been made that converts a std_logic_vector to a string[51]. This function converts to the string by converting each element in the vector by means of the image attribute of the std_logic type. Read from right to left, after each fourth element a period is inserted to increase readability. After executing the semi-code from code fragment 7 (on page 43) the value of str is: '"*11.0000.0000.0000.0000*"'. (The inner quotation marks are part of the string.) The function does not work in Modelsim due to an assignment with strings of unequal length.

[49] Tested only in ISIM.
[50] Initializing to 'U' or 'X' did not solve the problem.
[51] VHDL 2008 has these functions implemented in the standard. The simulation however can be done on software that does not support VHDL 2008.

```
variable vect : std_logic_vector(17 downto 0);
variable str   : string(0 to 100);
vect      := "110000000000000000";
str := std_logic_vector_to_string(vect);
```

Code fragment 7. Example of a conversion from a std_logic_vector into a string.

8.10Testing - real world

At a certain point in the development of the module, simulation alone was not enough to determine the correct operation of the design. Therefore the design was loaded onto the FPGA and connected to some test equipment. Details of all the test equipment can be found in appendix VI. The design was tested by manually verifying the output. If one image looks like it is supposed to, than I is likely they all are. When a good image was acquired, multiple readouts were done to assure the image stays good. This chapter will discuss the steps taken to get a good image.

8.10.1 Test image

A test image was first provided by the Baumer HXC 13 camera but was not specified in the documentation of the camera, so to be sure what to expect another camera, an Adimec 4000m/D, was used since the test image that camera provides is, in pixel-detail, described in the manual [12]. To ensure the test image was output as described it was connected to a system known to be working. Thus the test image below (figure 20) was acquired i.e. saved to both an image file and an ASCII-text file with the decimal values in a table. The image itself is a 2048 by 2028 pixel image with a bit-depth of 10 bits. This image is a static image.

Figure 20. Test image from the Adimec 4000m/D camera.[52] Left: full frame, right: top left corner.

8.10.2 Monitoring equipment

This chapter will discuss the testing equipment used. A complete list of the equipment with model numbers etc. can be found in appendix VI. For the first fast checks LEDs were used. But LEDs alone were not enough to see what was happening. An oscilloscope was used to look at the some of the signals. This was sufficient to test the flags and other indicators. But to check the data output, a faster and bigger (more than four channels) way of analysing the data was needed. To do this a logic analyzer was set up. To view all the required bits twenty-eight pins were needed to perform the tests without having to reassign and multiplex signals, although 12 would be sufficient with the multiplexing[53]. The board however did not have any GPIO pins, therefore the SD-card slot was used to provide the extra connections to the logic

[52] Note the black border of the image is not a part of the test image. This image has also been compressed with JPEG compression and is compressed again by the text editor. Please see the Adimec 4000M user manual for the exact test image.

[53] The camera does not use all 28 bits. Please see paragraph 8.1 for more information on this.

analyser (see figure 21). This was possible because there are no extra components between the FPGA and the SD-card slot and the SD-card slot has no obstructions preventing access to the relatively big pins connecting it to the board.

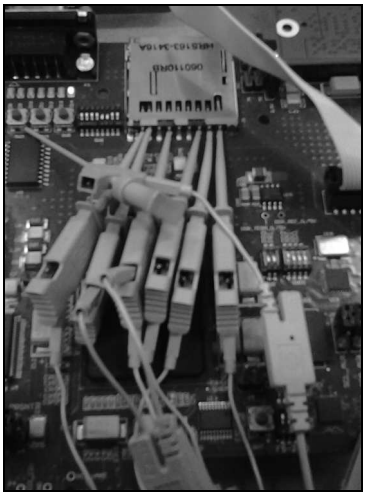


Figure 21. Probes on the SD card slot.

However there were more bits to look at. The board also has an ALI[54] connector. The other signals were connected to that connector. Not all pins of the 50-pin connector are connected to the FPGA so only a few could be used, some of which happened to have an offset which made some of the pins unusable for testing purposes. After being connected the logic analyzer gave strange values that could have been due to meta-stability or differences in clock speeds causing sampling to occur on incorrect data. Figure 22 shows this problem. There are spikes on Xdata 2 (indicated with an arrow) while it is expected to be "flat".

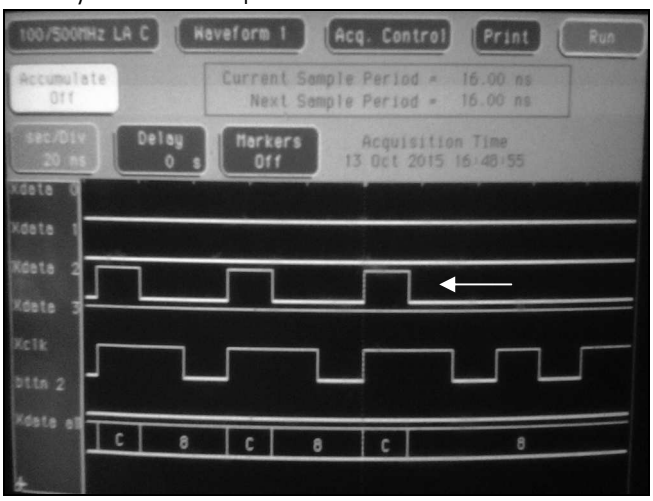


Figure 22. Screenshot from the logic analyzer with Xdata 2 showing spikes while it is expected to be all zero.

To overcome the possible difference in clock speeds another logic analyzer has been used. This logic analyzer is an HDL IP core generated by software provided by Xilinx. The IP core runs on the same clock as the hardware. That is, it runs on a clock that is twice as fast, but the two clocks are in phase with one another. Using this logic analyzer proved to be more reliable. But still

[54] Avnet LCD Interface [3].

some strange spikes occurred. The number of spikes were reduced, but not completely removed after inserting terminating resistors across the LVDS pairs. These terminators had been added early on in the design process, but were removed in an effort to get some data. Removing those resistors gave more level changes and thus made it look like it worked better. Also it was unclear whether or not those resistors were already on the FMC card. The documentation does not say anything about this, but not having them was deemed more logical[55]. After adding the resistors the spikes that were still there were put down to a clock skew between the system clock and the analyzer clock. Therefore other means of viewing the output were needed.

As mentioned before strange spikes occurred during the tests. In an effort to try and find the cause of this one of the inputs from the ChipScope was connected to logic '1'. The image in figure 23 below shows the value of that input under the name Constant high. The signal appears to have dips at regular intervals. With those dips all other signals also change their value. When this was found the time allotted for the development of the HDL was already over but a working module was required. Using the custom designed RS-232 testing component (please see paragraph 8.10.3) proved to be the most reliable way of testing with only a few fallen bits per image-line. And the module seemed to be working with a few small errors.

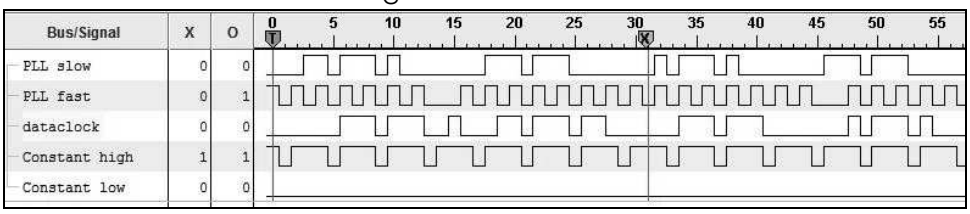


Figure 23. ChipScope screenshot with strange dips in the constant high signal.

8.10.3 Image storage and output

In order to see the image that has been received the entire image is sent to a computer and further processed by the computer to create an image file. To do this an RS-232 transmitter was created together with a RAM block. This module is named image_storage. The image is stored in the (on-chip) RAM and later read from there to be transmitted over the RS-232 line. The test image is, as stated in paragraph 8.10.1, a 2048 by 2048 pixel image. To store one image, for these tests multiple frames are not needed, 5 MB of storage is needed (for the calculations please see figure 24).

$$2048^2 = 4{,}194{,}304 \text{ pixels in one image}$$
$$4194304 \cdot 10 = 41{,}943{,}040 \text{ bits for one image}$$
$$\frac{41{,}943{,}040}{8} = 5{,}242{,}880 \text{ bytes for one image}$$
$$\frac{5{,}242{,}880}{1024^2} = 5 \text{ megabytes (MB) for one image}$$

Figure 24. Calculations for the image size.

This can come into conflict with the amount of RAM used by the on-board logic analyzer. To ensure both could be placed on the chip the memory of the logic analyzer has been reduced. The image was not completely stored

[55] If they had been on the card, the card would have to generate "new" signals to the board since the board cannot measure after the resistors and get a valid value.

to allow a smaller memory size. At a later moment the logic analyzer was removed to free up enough memory space to store the image. This was still not enough space. The maximum[56] size is 320 kB. This means a data width of 20 bits (two ten-bits pixels) and a data depth of 131,072. With this size 262,144 pixels can be stored. This results in the image having to be split up in 16 parts. The image is split vertically in groups of 128 lines. A counter is used to count the pixels and to stop if the memory is full. Then the image is sent. Another counter counts the groups of 128 lines that have been sent. If not all lines have yet been sent a new group of lines is read. The pixels are transmitted with a baudrate of 115200. Figure 25 shows an extremely simplified version of the image_storage module. The module has more counters than shown in the diagram also not shown is the state machine. This state machine controls when to read from the camera and when to write to the computer.

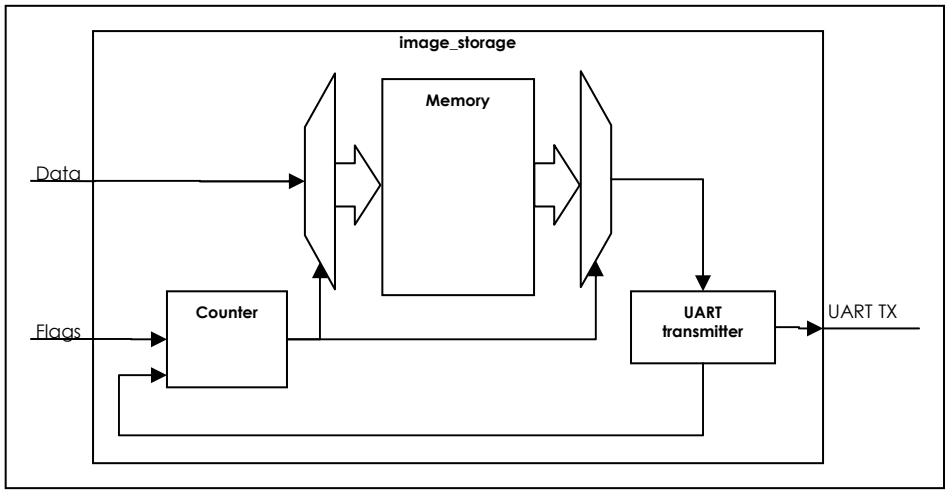


Figure 25. Simplified block diagram of the image storage with RS-232 transmitter.

8.10.4 Image size

The image size for a full-size frame of the Adimec 4000m was calculated to be 5 MB. This does not fit in the available memory, as shown in paragraph 8.10.2. This may not be a problem as de demonstrator currently uses a 256^2 image (256 by 256 pixels) and the most real-time applications work with a 512^2 image in order to run at a high frame-rate. Using the same steps as in figure 24 the size of the 512^2 image has been calculated to be 320 kB. In the same manner the memory size for the 256^2 image has been calculated at 80 kB. Both fit in the internal memory (which is 320 kB). The maximum image size with this memory size is 512 by 512 pixels. For calculations, please see figure 26 on page 48. Some processing can be done on a pixel-by-pixel basis so there may not be a need to store the image, enabling faster processing and processing of bigger images. If bigger images do need to be stored, external memory is needed[57].

[56] This may not be truly the maximum size, but to keep the design and calculations simple a size is chosen such that a multiplication by a power of 2 results in the image.
[57] When working with the Spartan 6 at least.

$$320 \cdot 1024 = 327{,}680 \text{ bytes memory size}$$
$$327{,}680 \cdot 8 = 2{,}621{,}440 \text{ bits memory size}$$
$$\frac{2{,}621{,}440}{10} = 262{,}144 \text{ pixels}$$
$$\sqrt{262{,}144} = 512 \text{ so the image is 512 by 512 pixels}$$

Figure 26. Calculations for the image size of a 512 by 512 image.

8.10.5 Test results

As stated in paragraph 8.10.2 the module had a few small errors. The CameraLink module has been tested using the image_storage module with the RS-232 transmitter. The image that was acquired in that way was not correct. When comparing that image (please see figure 27 below) with the image the camera transmits as seen in figure 20 on page 44, you can clearly see there are a number of differences. The image in figure 27 has the same size but contains the image four times and the lines are not in sync. The lines are shifted, but not with a constant number of pixels and the lines are not always shifted in equal groups. The transmission of the test image was 3,414 pixels short of the number of pixels needed to create the image. The same number of (black)pixels was added at the end of the image fill the image to the correct size.

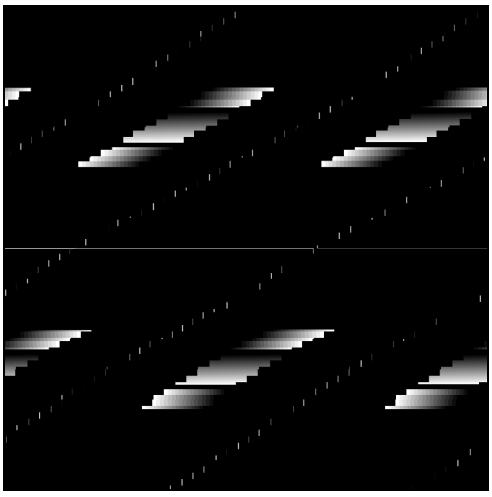


Figure 27. First test image, clearly not quite correct.

The reason for these errors could be a wrong transmission between the FPGA and the computer or the CameraLink module does not behave correctly. Another reason for the errors may be found in the way the image is stored and retrieved from the memory. The compilation warnings were checked to see if there were any missed in previous checks. Also the timing requirements were checked. This showed a failure in the receiver. The path delay from the SERDES was about three and a half times too late.

A solution for this might be changing the timing constraints. These were set for a CameraLink frequency of 85 MHz. This timing constraint could not be met by the synthesizer. The test image is provided with a speed of 40 MHz. This big difference in speed could help meet the timing constraints. This helped but it was not enough to solve the problem. The CameraLink_link module was checked and it was found it only signalled the arrival of half the bits, with that fixed new tests were run. This resulted in the image in figure 28. This was already seen when transmitting the image. Some lines were sent with the

correct number of pixels, but others were a number of pixels short. The pixels lost per transmission part are given in table 2. The table shows that some groups of lines are transmitted correctly, but others were not.

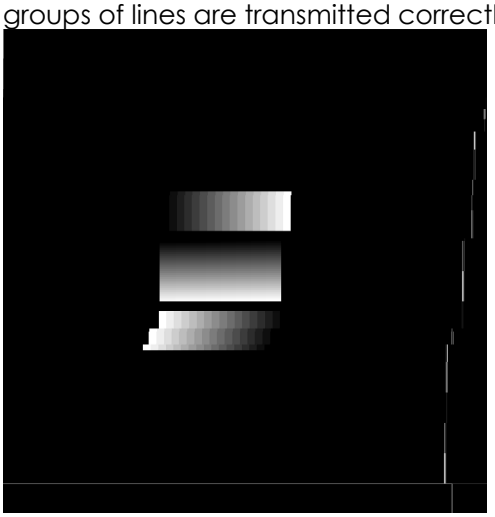


Figure 28. Second test image, still not correct.

Image part	Lines	Pixels		Difference	
		Expected	Read	Absolute	Relative
1	128	262144	262144	0	0
2	256	524288	524288	0	0
3	384	786432	786418	-14	-14
4	512	1048576	1048523	-53	-39
5	640	1310720	1310666	-54	-1
6	768	1572864	1572808	-56	-2
7	896	1835008	1834916	-92	-36
8	1024	2097152	2097060	-92	0
9	1152	2359296	2359204	-92	0
10	1280	2621440	2621308	-132	-40
11	1408	2883584	2883428	-156	-24
12	1536	3145728	3145572	-156	0
13	1664	3407872	3407716	-156	0
14	1792	3670016	3669860	-156	0
15	1920	3932160	3932004	-156	0
16	2048	4194304	4194148	-156	0

Table 2. List with number of pixels per transmission.

In an effort to reduce the lost pixels the number of stopbits was increased from one to five. This effectively gives a bigger break between the pixels reducing the risk of data loss. This in combination with a lower baudrate (9600) should at least cause a loss of fewer pixels. While it did indeed reduce the lost pixels, it did not solve the problem. No image was produced because during transmission the bits were counted and it was found some pixels were lost. The analogue D-sub-9 connector is used for the pixel transmission. The data lines running from the FPGA to the computer lay next to the power supply for the FPGA, a laptop and the power supply for that laptop. This could mean some bits may fall. Also the receiving computer is running windows, which is known for losing data from the serial port. To overcome these problems the USB-RS232 converter on the development board is used instead. This, in combination with giving the software reading the serial port the highest

priority, resulted in no lost pixels. The image now looks more like the one the camera is sending (please see figure 29). The way the image is stored is not quite right. The top of the image is stored twice. Once at the correct position, once in the lower parts of the image. The number of pixels is correct, but the pixel values are not.

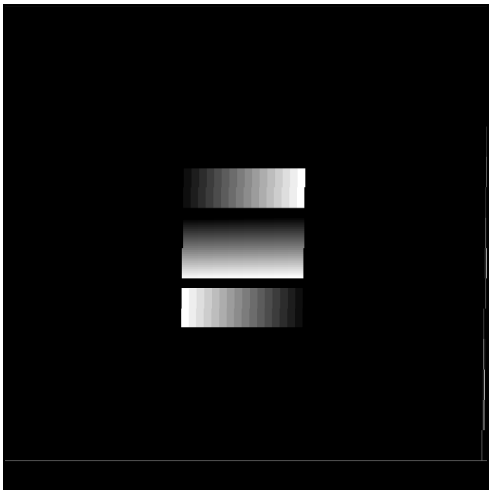


Figure 29. Test image, almost like expected.

The most visible error is the one where the last block of 64 lines shows the first 64 lines. This is clearly an error in the image_storage module. While this was corrected the transmitting speed was changed to the fastest option on the pc (256000 baud). This changed the transmission time of an image from almost 2½ hours to a much more practical 4¼[58] minutes. The settings on the camera were also changed to trigger free-running. Thus the time it takes to see the transmission stop and then push a button are removed as well. From just the transmission data it showed the first pixels having the wrong value (Please see figure 30). These pixels are not written to the memory. Some pixels at the side edges of the image are incorrect as well. Still an image was produced to see how far wrong, or correct, the image is. When taken a closer look at the image it appears as if every second pair of pixels is repeated (please see figure 32).

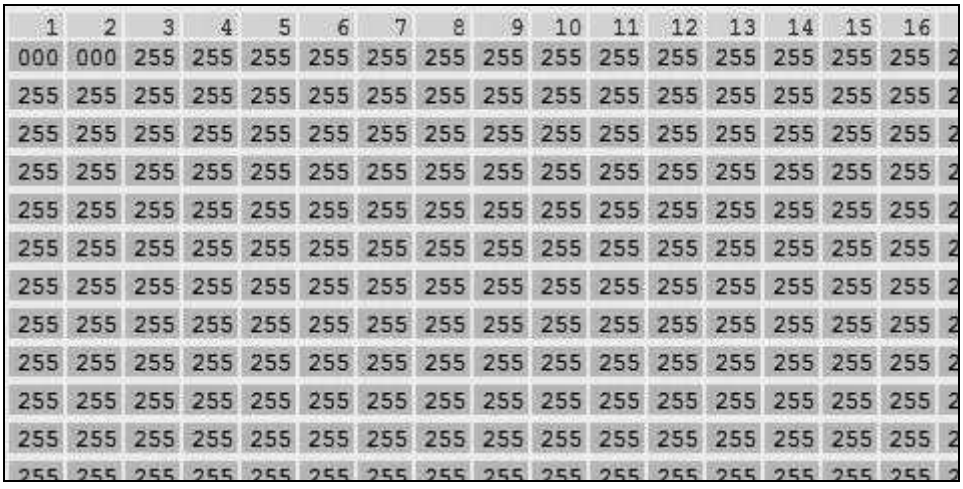


Figure 30. Screenshot of the received pixel values.

[58] Actually varies slightly. The first one after reset takes 04:16 the ones after that take about 04:24.

To get a better view on this a real image was used instead of the test image. Pixel duplication is better seen in a real image as the values per pixel differ much more than in the test image. The pixel duplication was found to be true, and this was corrected for. The resulting image can be seen in figure 31. This image shows some strange interleaving effect. This was caused by wrong counter limits.

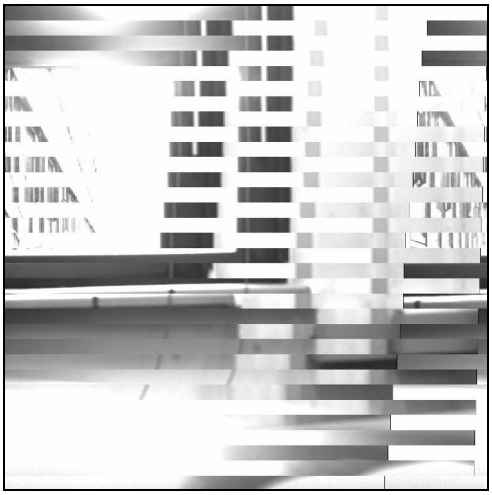


Figure 31. Real image, not completely correct.

With the counter limits adjusted the image looked much better, but not quite as it should have been. Figure 32 shows the problems on that image. The number of pixels per block are one pixel short of what it should be. This means the image wraps around. The other problem is duplicate pixels. This is showed on the right side of the image. For every line, each group of two pixels is duplicated.

The problems are now only in the test modules. Some signals are not delayed long enough, or become active too late. All these problems will be systematically taken care of.

Figure 32. Real image, with an error causing a wrap and pixels being repeated.

The first thing to be corrected were the counter limits. Now the image looked good, with the exception of the first two pixels being duplicated. Also some extra pixels were sent. Figure 33 on page 52 shows this image. The duplicate first pixels cannot be seen at this scale and the extra pixels are cropped so it looks like a good image. The problem of the first pixel was solved by delaying

the moment a counter was incremented. The extra pixels were solved by, again, changing the counter limits.

Figure 33. Correct image, slightly oversize and incorrect first pixel.

The image was now received as it should, with the exception of each last pixel from the image parts. (16 times a wrong pixel at the end of a line). These pixels had the value the previous end of the image part should have had. Since this is not visible no image of this is given. This was caused by the writer. it stopped too early, thus it did not write that last pixel.

After that was done a correct image was received (please see figure 34). And this was done multiple times. Sometimes the received image was missing some pixels, but this was caused by the computer also performing other tasks.

By viewing all the test images next to one another the progress described here can be easily seen. Appendix VII shows this.

Figure 34. Correct image.

9 Conclusion

This project was set up to look into the possible use of The MathWorks' HDL Coder as a way to simplify the development for FPGAs. To answer the question if it is a good option research questions have been posed. These questions, listed in chapter 4 would help to answer this question. In paragraph 6.3 can be read that the practical trials with the HDL Coder were not conducted. This makes answering these questions difficult, nonetheless they will be answered. Mostly from documentation. Some of this documentation is supplied by The MathWorks. This must be considered biased as they want to sell the software. Thus a little less weight is given to the documents from The MathWorks. On the other hand the questions on the MATLAB forum are not representative of the HDL Coder user base either. Apart from these sources a user of the HDL Coder has answered some questions regarding the HDL Coder. These sub-questions will be answered first, then the answer to the main question is answered. At the end of this chapter the HDL for the CameraLink interface is discussed.

9.1 Answers to the research questions

To answer the main question sub-questions have been posed. These will be answered here.

What is the ease of use of the software?
The ease of use is hard to tell when the software itself has not been tested. Thus an answer to this question relies on second-hand experience. This is, for the most part, positive.

Is it possible to insert custom-made HDL-code into the design?
Adding custom-made HDL into the design should be easy according to the documentation available. Paragraph 6.1 touches the subject and that chapter refers to appendix I where this subject is also discussed. This question can be answered with a simple yes. It is possible, but it is not an easy thing to do.

To what extent does the user need to know about digital hardware/FPGA design?
This is discussed in paragraph 6.5. As stated in that paragraph quantifying knowledge is difficult. So to what extent is not really answerable but a list of the required knowledge and skills is listed in table 1. The user has only one subject that he needs to know a lot about. This is the subject of timing. This is also the most critical part of designing digital hardware. So a hardware designer may find the tool easier to use than someone with a Simulink background[59].

Which FPGA boards are supported?
This is discussed in paragraph 6.5 as well. All boards are supported as it generates HDL. Just as with manually written HDL, it works on all platforms as long as it is kept generic as discussed in paragraph 8.4.

9.2 Answer to the main question

So is the HDL Coder a good option to simplify development for FPGAs? No it is not. As long as specific knowledge about the FPGA timing is required it does not simplify the design. It might be used to generate an HDL framework which then can be optimized. This depends on the quality of the HDL generated by the HDL Coder.

[59] These are not mutual exclusive. Someone with a background in designing digital circuits may also know how to use Simulink and v.v.

Does this mean it is not a good tool? No it does not. Although still some specific knowledge is needed the whole translation from a model in one "language" (Simulink) into another (HDL) is no longer a process that needs to be done by hand. The HDL Coder is a good tool, but not quite there yet. If The MathWorks is able to create better documentation and if the HDL Coder can help solving the timing issues this will make design for FPGAs much easier.

9.3 Regarding the CameraLink description

The hardware description for the CameraLink does work. The module has been made as generic as possible to accommodate all kinds of chips and all the CameraLink configurations. Not all these configurations have been so it is not finished. The module has been proven to work in mode base, sending two ten-bits pixels at a frequency of 40 MHz.

10 Recommendations

The recommendations are split into multiple paragraphs. The first paragraph will list and discuss the recommendations for the CameraLink HDL and the second discusses the recommendations for the continuing line of the project. The third paragraph will list some lessons learned and can be applied outside the project.

The recommendations are prioritized using the MoSCoW method. All the consonants (in MoSCoW) stand for the modal verbs:

- Must
- Should
- Could
- Would

For readability those modal verbs are abbreviated to their first letter. Recommendations indicated with must (M) have the highest priority and must be implemented or executed to finish the product. A bit less urgent, but still a good idea to implement are the recommendations indicated with should(S). Should means it is a good idea to realize this recommendation but the design will function without it or the project does not depend on this action. Could(C) indicates the recommendations that could be implemented as long as it does not affect the other requirements or aversely affect the functionality. The lowest priority is would (W). Would indicates ideas that may be implemented in future versions, or tasks that might be done if all other recommendations are implemented or done and time allows.

10.1 Recommendations for the HDL

Before the HDL can be used in production some things need to be either removed, adapted or added. This paragraph will discuss these changes.

M Test all configurations.
At this moment only the two channel, ten bit configuration is tested. This was done at a frequency of 40 MHz. To ensure correct behaviour at all configurations these will have to be tested.

M Repair the testbench.
Some parts of the testbench start running out of sync after a few transmissions. This triggers errors while the behaviour is still correct. In order to simulate all configurations this will have to be fixed.

M Build a reset synchronizer into the CameraLink_link module. This way the hardware in the data clock domain can be reset safely. The reset source should be the reset signal from the CameraLink module.

M Use a reset synchronizer in the design where the CameraLink module will be a part of. The reset is not synchronized in the CameraLink module, so this must be done in one of the higher hierarchy levels.

S Remove logic analyzer IP from the design.
During the development of the hardware a logic analyzer IP was used to look at the signals inside the FPGA. This IP is used depending on a constant defined in the CameraLink module. After testing the analyzer is no longer needed and can be removed from the design. When removing the analyzer all the corresponding signals, constants and if-statements should be removed as well.

S Cleanup the design.
Remove all unused constants, variables and pieces of commented code, as long as they are not needed. Watch out for the conditional implementation of the libraries in the source code of the CameraLink library. Some pieces of code there are commented but should not be removed.

S Rewrite std_logic_vector_to_string
The function std_logic_vector_to_string does not work in Modelsim. While ISim has no problems with the assignment of strings of unequal lengths, Modelsim does. Adapt or rewrite this function to make it work in Modelsim too.

S Create a list of delay times.
When using a black box in the HDL Coder delay times need to be entered. It would be convenient to be able to look up the delays in a table instead of having to be either measured or deduced from HDL. So for each chip the delay times for all significant signals should be given. The delays of the signals in table 3 must be known.

From	To
First data clock	PLL locked
Data input	Data output
camera_control	cc_data
camera_control	trigger_indicator
Data input	new_data
Data input	New_pixel_indicator

Table 3. Signals to determine the delay of.

10.2 Recommendations for the continuity

The project is not quite finished. In order to ensure the continuity of the project the following recommendations are made.

M Acquire a license for the HDL Coder.
In order to give a better answer as to the user friendliness and to integrate the CameraLink interface into a system the HDL Coder is needed.

M Add the CameraLink module to the Simulink environment
Create the interface for the camera in the
M Check the HDL output of the HDL Coder.
In order to say something about the HDL Coder capabilities the generated HDL should be looked at to see the readability and coding style.

M Compare with the OpenCL environment.
As the "High Performance Real-time Processing developments" project looks into OpenCL as well as the HDL Coder the two will have to be compared. As soon as something can be said about both they should be compared.

10.3 Lessons learned

Apart from the recommendations that are specific to this project other recommendations can be made. These are listed here.

M Start with acquiring a license.
When a license is needed, acquiring one must be one of the first things done in the project. If a time-limited license is used, let the activation take place when the license is needed, but then the project can continue when the license is needed. So when developing for Xilinx it would be helpful to be able to use Modelsim. But before doing his the capabilities of the Vivado simulator[60] should be looked into.

C Acquire a license for Modelsim.
The ISim simulator cannot display the delta delays and this makes the development a little more difficult and thus a little more time consuming.

[60] The successor of ISim.

11 Glossary

(Design)Area
: The size of (a part of) the design; how big a space the design takes on the chip.

Bit-depth
: The number of bits used to hold a value. A bit-depth of 10 bits means the value can range from 0 to 1023.

CoaXPress
: A standard for video-transmission and camera control with power being supplied over the data cables [26].

Component
: A "virtual chip" in a hardware description when used in an entity.

Deserialization
: The inverse operation of serialization. I.e. get serial input and output it parallel.

Entity
: A "virtual chip" in a hardware description.

Generic
: A VHDL keyword that is used to generate hardware under given conditions. Similar to #define in C-languages.

Instantiation
: "Virtual placement and soldering" of components in a hardware description.

PCIe
: A high-speed computer bus to expand the capabilities of the computer

Port
: "Virtual pins" of entities in the hardware description.

Phase-Locked Loop
: A circuit used to change the phase and/or frequency of a periodic signal.

Pipeline
: A row of signal manipulations or calculations.

Pixel depth
: The number of bits used to hold the value of the pixel. A pixel depth of 10 bits means the pixel can have a (greyscale) value ranging from 0 to 1023.

Planar movement
: Movement parallel to a plane with only two degrees of freedom, not counting rotations, to reach all positions on that plane.

PLL
: Please see Phase-Locked Loop

Spec
: Specification

Toolchain
: A series of software applications that work in series to complete a certain task. In this case the compilation and synthesis and/or simulation of the hardware is meant.

Tri-state
: (A signal) that can be both input and output. (Output '0', Output '1' and input (three options)

12 References

This list of references contains all sources used to create the CameraLink interface and all documentation regarding the interface, including this report. So some references are listed here, but not referenced in this document. For the webfora the original poster is mentioned as author. This is in most cases not the only "author". The order of the references is arbitrary and does not say anything about the chronological order nor does it say anything about the importance or extent of use.

12.1 Textual documents

[1] Author: Photon focus
Title: Application note AN021 V1.0
Created in: July 2004
Viewed on: 28 September 2015

[2] Author: Texas Instruments
Title: Texas Instruments DS90CR287/DS90CR288A +3.3V Rising Edge Data Strobe LVDS 28-Bit Channel Link - 85MHz)
Created in: October 1999
Revised in: March 2013

[3] Author: Avnet
Title: Avnet LCD Interface Specification (ALI) Revision 1.00
Created on: Unknown
Revised on: 9 May 2010[61]

[4] Author: CameraLink
Title: Specifications of the CameraLink Interface Standard for Digital Cameras and Frame Grabbers
Created in: October 2000

[5] Author: Adimec
Title: Adimec-4000M Operating and Technical Manual
Created on: 6 May 2003

[6] Author: Xilinx
Title: Spartan-6 Libraries Guide for Schematic Designs (v 11.4)
Created on: 2 December 2009

[7] Authors: C. E. Cummings, D. Mills
Title: Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use? Revision 1.1
Created in: 2002

[8] Author: Alpha Data
Title: FMC-Cameralink User Manual v2.3
Created: Unknown

[9] Authors: Active Silicon Ltd ; Chris Beynon, Adimec Advanced Image Systems BV; Jochem Hermann
Title: Coaxpress the next generation digital interface Rev. 1.0
Created on: 27 November 2009

[61] It could also be 3 September as the date notation was numerical in the source document.

[10] Author: National Instruments
Title: PCI Express - An Overview of the PCI Express Standard
Created on: 5 November 2015

[11] Author: Baumer
Title: Baumer HXC 13 v14 users guide
Created in: September 2012

[12] Author: Adimec
Title: Manual ADIMEC 4000 M/D
Created on: 6 May 2003

[13] Author: Photon focus
Title: User Manual MV1-D1024E CameraLink Series CMOS Area Scan Cameras September 2014 V1.0
Created in: September 2014

[14] Author: Xilinx
Title: Spartan-6 FPGA SelectIO Resources User Guide v.1.7
Created on: 21 October 2015

[15] Authors: Xilinx; Nick Sawyer
Title: Source-Synchronous Serialization and Deserialization (up to 1050 Mb/s) v1.2
Created on: 19 November 2013

[16] Author: Basler AG
Title: Easier than ever – What to Consider in Modern (Po)CL Camera Systems (Setup, Advantages, and Costs)
Created in: September 2012

[17] Authors: Texas instruments; National Semiconductor
Title: Channel Link II Design Guide
Created in: 2011

[18] Author: Basler Vision Technologies
Title: Camera Link Technology Brief
Created on: 28 March 2001

[19] Author: Gandhi Puvvada
Title: Steps to run compxlib to compile Xilinx libraries in Modelsim SE10.1 for EE101/EE201L/EE560 students as well as USC ITS
Created on: 26 January 2012

[20] Author: The MathWorks Inc.
Title: HDL Coder User's Guide
Created in: March 2012
Revised in: March 2015

12.2Websites

[21] Author: Bureau International des Poids et Mesures
Title: SI Brochure: The International System of Units (SI)[8th edition, 2006; updated in 2014] Chapter 3
Created on: unknown
Available at: http://www.bipm.org/en/publications/si-brochure
Viewed on: 13 November 2015

[22] Author: TNO
Title: About TNO
Created on: unknown
Available at: https://www.tno.nl/en/about-tno/
Viewed between: 25 August and 13 October 2015

[23] Author: TNO
Title: Intelligent Imaging
Created on: unknown
Available at: https://www.tno.nl/nl/samenwerken/expertise/technical-sciences/intelligent-imaging/
Viewed between: 25 August and 17 November 2015

[24] Author: TNO
Title: Optics
Created on: unknown
Available at: https://www.tno.nl/en/collaboration/expertise/technical-sciences/optics/
Visited on: 23 November 2015

[25] Author: TNO
Title: Optomechatronics
Created on: unknown
Available at: https://www.tno.nl/nl/samenwerken/expertise/technical-sciences/optomechatronics/
Visited on: 23 November 2015

[26] Author: CoaXPress
Title: What is CoaXPress?
Created on: unknown
Available at: http://www.coaxpress.com/coaxpress.php
Viewed between: 24 August and 28 August 2015

[27] Author: The MathWorks Inc.
Title: MathWorks, Simulink - Simulation and Model-Based Design
Created on: unknown
Available at: http://nl.mathworks.com/products/simulink/
Viewed between: 26 August and 27 August 2015

[28] Authors: EE Times; Saurabh Verma, Ashima S. Dabare, Atrenta:
Title: Design How-To, Understanding clock domain crossing issues
Created on: 17 July 2012
Available at: http://www.eetimes.com
Viewed on: 26 October 2015
Note: Image has been redrawn.

[29] Authors: EDN Network; Tejas Dave , Amit Jain & Divyanshu Jain:
Title: Synchronizer techniques for multi-clock domain SoCs & FPGAs
Created on: 30 September 2014
Available at: http://www.edn.com
Viewed on: 26 October 2015

[30] Author: Mikrocontroller.net forum; Andreas N.
Title: Problem mit Xilinx SERDES bei EMV Störung
Created on: 22 May 2013
Available at: http://www.mikrocontroller.net
Viewed on: 8 September 2015

[31] Author: NoASIC; Guy Eschemann
Title: How (not) to design a 2DFF Synchronizer
Created on: 10 December 2012
Available at: http://noasic.com/blog/how-not-to-design-a-2dff-synchronizer/
Viewed between: 16 November and 17 November 2015

[32] Author: The MathWorks Inc.
Title: Simulink Simulation and Model-Based Design
Created on: unknown
Available at: http://nl.mathworks.com/products/simulink/
Viewed on: 25 August 2015

[33] Author: CoaXPress
Title: CoaXPress
Created on: unknown
Available at: http://www.coaxpress.com/
Viewed between: 24 August and 28 August 2015

[34] Author: Quality magazine; John Egri
Title: Consider CoaXPress
Created on: 1 February 2013
Available at: http://www.qualitymag.com/articles/90956-consider-coaxpress
Viewed on: 31 August 2015

[35] Author: VolkerSchatz; Volker Schatz
Title: The Camera Link camera interface
Created: unknown
Available at: http://www.volkerschatz.com/hardware/clink.html
Viewed between: 2 September and 8 October

[36] Author: National Instruments
Title: What Are The Differences Between Base, Medium, and Full Camera Link Configurations?
Created on: 27 February 2013
Updated on: 24 April 2015
Available at: http://digital.ni.com/public.nsf/allkb/2EE80B8C381D61D286257B1F005674A5
Viewed between: 3 September and 1 October 2015

[37] Author: Xilinx Forum; patric.lewis
Title: Spartan 6 Camera Link Receiver
Created on: 3 August 2012
Available at: https://forums.xilinx.com/t5/General-Technical-Discussion/Spartan-6-Camera-Link-Receiver/td-p/252640
Viewed on: 7 September 2015

[38] Author: StackExchange.com - Electrical Engineering; sj755
Title: How to Add the Xilinx Library to Modelsim
Created on: 9 March 2013
Available at: electronics.stackexchange.com/questions/60319/
Viewed between: 20 October and 19 November 2015

[39] Author: Xilinx
Title: ISE - Simulation libraries
Created on: unknown
Available at: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_simulation_libraries.htm
Viewed between: 20 October and 19 November 2015

[40] Author: Xilinx Forum; Pawel
Title: BUFPLL problems (driving SERDES)
Created on: 14 June 2014
Available at: https://forums.xilinx.com/t5/Spartan-Family-FPGAs/BUFPLL-problems-driving-SERDES/td-p/489860?db=5
Viewed on: 23 November 2015

[41] Author: Xilinx Forum; PietervanderStar
Title: ISIM loop unrollment gives multiple drivers?
Created on: 27 October 2015
Available at: https://forums.xilinx.com/t5/Simulation-and-Verification/ISIM-loop-unrollment-gives-multiple-drivers/m-p/662007
Viewed on: 27 October 2015

[42] Author: The MathWorks Inc.
Title: HDL Coder
Created on: unknown
Available at: http://nl.mathworks.com/products/hdl-coder/
Viewed between: 27 August and 2 December 2015

[43] Author: The MathWorks Inc.
Title: Vision HDL Toolbox
Created on: unknown
Available at: http://www.mathworks.com/products/vision-hdl/
Viewed on: 27 August 2015

[44] Author: The MathWorks Inc.
Title: Using Multiple Clocks in HDL Coder
Created on : unknown
Available at: http://nl.mathworks.com/help/hdlcoder/examples/using-multiple-clocks-in-hdl-coder.html
Viewed on: 14 December 2015

[45] Author: MATLAB Central; Liclin
Title: A question about model-based design for video/image processing
Created on: 14 August 2015
Available at: http://nl.mathworks.com/matlabcentral/answers/233837
Viewed on: 27 August 2015

[46] Author: MATLAB Central; Tom
Title: HDL Coder "Requested CP number exceeds total number of register-to-register CPs"
Created on: 16 April 2015
Available at: http://www.mathworks.com/matlabcentral/answers/203913
Viewed on: 27 august 2015

[47] Author: MATLAB Central; Tamer
Title: Found unsupported division expression for HDL ode generation; signed input data type is not supported for division with Floor RoundMode
Created on: 6 July 2015
Available at: http://www.mathworks.com/matlabcentral/answers/228493
Viewed on: 27 August 2015

[48] Author: MATLAB Central; Tamer
Title: Error: Cast between fixpt and floating point type is not supported
Created on: 19 June 2015
Available at: http://www.mathworks.com/matlabcentral/answers/224524
Viewed on: 27 August 2015

[49] Author: MATLAB Central; Shubham Kapoor
Title: how to use exp funnction in hdl coder?
Created on: 22 April 2015
Available at: http://www.mathworks.com/matlabcentral/answers/212861
Viewed on: 27 August 2015

[50] Author: MATLAB Central; Tom
Title: Integrating HDL Coder, System Generator and VHDL/Verilog projects
Created on: 12 May 2015
Available at: http://www.mathworks.com/matlabcentral/answers/216394
Viewed on: 27 August 2015

[51] Author: MATLAB Central: fl
Title: Does HDL Coder support testbench generation for FFT block in Simulink?
Created on: 21 February 2013
Available at: http://www.mathworks.com/matlabcentral/newsreader/view_thread/326951
Viewed on: 2 December 2015

[52] Authors: Electronic Engineering Journal; Stephan van Beek and Sudir Sharma, MathWorks
Title: Best Practises for FPGA Prototyping of MATLAB and Simulink Algorithms
Created on: 25 August 2011
Available at: http://www.eejournal.com/archives/articles/20110825-mathworks/
Viewed on: 27 August 2015

[53] Author: BDTI, InsideDSP
Title: MathWorks' HDL Coder and Verifier: High-Level Synthesis Expands to MATLAB Users
Created on: 4 September 2012
Available at: http://www.bdti.com/InsideDSP/2012/09/05/MathWorks
Viewed on: 27 August 2015

[54] Authors: LinkedIn; Rexa Ameli, Nutaq
Title: What is your way of writing DSP HDL codes?
Created on: 4 May 2012
Available at[62]: https://www.linkedin.com/grp/post/1817484-112741367
Viewed on: 27 august 2015

[55] Author: The MathWorks Inc.
Title: Create a Custom Block
Created on: unknown
Available at: http://nl.mathworks.com/help/simulink/ug/tutorial-creating-a-custom-block.html
Viewed on: 27 August 2015

[56] Author: The MathWorks Inc.
Title: FPGA Design and Codesign
Created on: unknown
Available at: http://nl.mathworks.com/solutions/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html
Viewed on: 2 December 2015

12.3 Other documents

[57] Author: J.E.J. op den Brouw
Title: Digitale Systeem Engineering 1 2013/2014
Document type: Slides used in lessons
Created on: unknown

[58] Author: IEEE
Title: Standard VHDL Synthesis Packages (IEEE Std 1076.3, NUMERIC_STD)
Version: 2.4
Document Type: VHDL Library
Created on: 12 April 1995

[62] This is only available for group members, but when using Google's cache it can be read by non-members as well.

13 List of figures

14 List of code fragments

15 List of tables

16 List of formulas

Appendix I. Expanding the use of Simulink's HDL generation

Number of pages: 4
Description: This document describes the things to keep in mind when designing HDL that is to be used with The MathWorks' HDL coder and the things to think of when designing in Simulink for HDL generation. This document is written with the design of a system with a camera as input in mind.

Expanding the use of Simulink's HDL generation

In order to use a camera as an input on an FPGA implementation made with Simulink, the camera-interface will have to be a Simulink block as well. According to The MathWorks it is possible to define custom Simulink blocks. These blocks are S-functions [1]. These are system functions and are written in MATLAB, C, C++ or FORTRAN [2]. If a function is described as HDL a black-box will need to be used [4]. This black box is described in MATLAB. There the ports are created and the entity and architecture names are connected[1], the delay in clock pulses is mentioned etc. [5]. The behaviour can be described in both VHDL and Verilog. The help-pages from The MathWorks are behind a license check and cannot be viewed [6]. There is a Wiki page that explains how this works for Xilinx [7]. It is possible to do this for Altera as well, but there are no good examples available [8]. Via third-party websites manuals of the HDL coder can be downloaded [9]. The black-box is not used for simulation, only for HDL generation. This means the developer has to create a simulation model [10]. When using the Altera DSP Builder the HDL Import block can be used. This way HDL or a Quartus II project can be imported[2]. The imported description is used to generate a simulation model [11]. It is also possible to use the EDA Simulator Link to connect the Simulink simulation to for example Modelsim [10][12]. It depends on the application whether or not a simulation model is required. If the entire system needs to be simulated some form of a simulation model is needed. But how good must this model be. If the inputs on the model are the CoaXPress signals a full model is required. If images are the input only a correct delay supplemented with some control signals would be sufficient. Another option is to create the interface in Simulink itself. This can be done specific to the manufacturer with the DSP builder from Altera or the System Generator from Xilinx. It is not clear however to what extend these are usable for things other than DSP's. Both Simulink libraries require the purchase of a license [13][14][15][16].

The problems that might occur are also looked into, but these are mostly limited to not reading the instructions or errors in the written code [17][18].

The information The MathWorks provides is only enough to know if it is possible to expand the use of the Simulink environment. The question about feasibility remains yet unanswered. The company BitFlow makes framegrabbers and has created a MATLAB and Simulink adaptor for their framegrabbers. This adaptor enables their products to function as input for both MATLAB and Simulink. This adaptor is a piece of software that creates a connection between the drivers of the framegrabber and the Image Acquisition Toolbox framework (please see figure 1) [19]. This would mean that the interface that is to be developed must use the drivers of the FPGA board. This in turn means the drivers become a major criterion in the selection process for boards. Not only must the current operating systems be supported, but the support must be continued for future versions as well. To what extend that is needed is determined by the planned economic and technical lifespan of the board. In other words for how long does the board work and for how long can this board remain profitable and does it not obstruct the development process of the system.

Using an adaptor means part of the expansion of Simulink must be done with separate software and cannot be done using MATLAB or Simulink. However the adaptor is only needed

[1] This is only the case for VHDL, when using Verilog the modulename must be entered [3].

[2] After this research was done Altera released the next generation of Quartus. It is unknown if this capability still exists.

if calculations will be done in either MATLAB or Simulink with input from the card or if the results are written to that card.

If an adaptor is needed the Image Acquisition Toolbox Adaptor Kit is needed. This is a C++ framework. The adaptor is a DLL (Windows) or a shared library (Linux). The functions of the framework will have to be written by the programmer, possibly extended with custom classes to pass on the data internally. All required functions are explained in the Image Acquisition Toolbox Adaptor Kit - User's Guide. That user guide gives an example algorithm for some of the functions [19].

The HDL Coder manual contains a list with the supported floating-point operations. These operations are partially dependant of the FPGA manufacturer. For some of the unsupported functions HDL Coder can generate a lookup-table to replace the function. The lookup-table functionality of Simulink cannot be used. This applies to Simulink counters as well [3]. Given these limitations and other hints in the manual it looks like a lot of knowledge about FPGAs and hardware design is still required in order to generate good HDL. An important upside of the HDL Coder is the conversion from floating to fixed point numbers. This can be done within Simulink which makes the process of determining the size of the fixed point number easier.

From all this can be concluded that expanding the use of the Simulink environment is possible. The feasibility is something this first exploration cannot answer. More in-depth research, and preferably first-hand usage may provide an answer to this question.

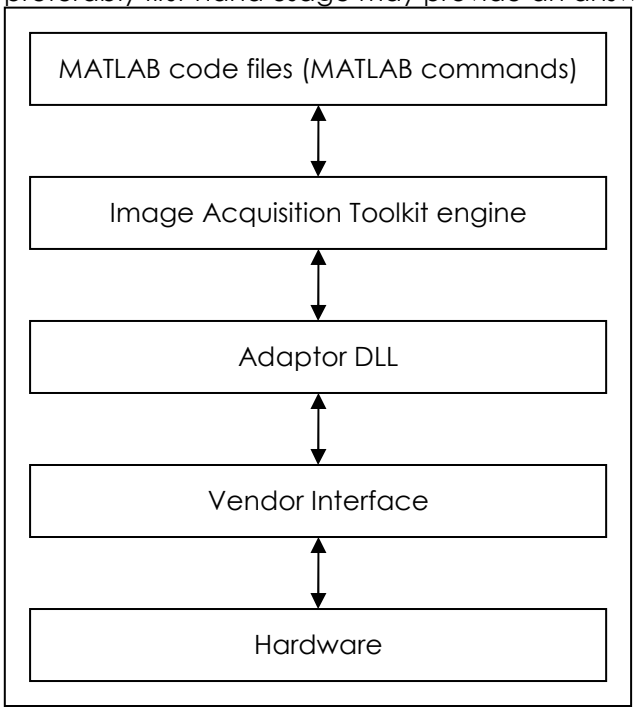


Figure 1. Connection between the adaptor and the Toolkit [20].

[1] Author: The MathWorks Inc.
Title: "Create a Custom Block"
Created on: unknown
Available at: http://nl.mathworks.com/help/simulink/ug/tutorial-creating-a-custom-block.html
Viewed on: 27 August 2015

[2] Author: The MathWorks Inc.
Title: "What Is an S-Function?"
Created on: unknown
available at: http://nl.mathworks.com/help/simulink/sfg/what-is-an-s-function.html
Viewed on: 27 August 2015

[3] Author: The MathWorks Inc. (2012)
Title: HDL Coder User's Guide, version 3.6 release 2015a
Created in: March 2012
Revised in: March 2015

[4] Author: Matlab Central; Tom
Title: "Integrating HDL Coder, System Generator and VHDL/Verilog projects"
Created on: 12 May 2015
Available at: http://nl.mathworks.com/matlabcentral/answers/216394-integrating-hdl-coder-system-generator-and-vhdl-verilog-projects
Viewed on: 27 August 2015

[5] Author: The MathWorks Inc.
Title: "Black box for including custom HDL code"
Created on: unknown
Available at: http://nl.mathworks.com/help/hdlcoder/ref/hdl.blackbox-class.html
Viewed between: 27 August 2015 en 28 August 2015

[6] Author: The MathWorks Inc.
Title: "MathWorks Account Login"
Created on: unknown
Available at: http://nl.mathworks.com/help/hdlcoder/ug/include-custom-hdl-code.html
Viewed on: 28 August 2015

[7] Author: CASPER; Jack Hickish
Title: " Tutorial HDL Black Box "
Created on: 4 September 2013
Available at: https://casper.berkeley.edu/wiki/Tutorial_HDL_Black_Box
Viewed on: 28 August 2015

[8] Author: Altera Forum BMX
Title: " MOving from Vhdl code to simulink black box"
Created in: April 2010
Available at: http://www.alteraforum.com/forum/showthread.php?t=22790
Viewed on: 28 August 2015

[9] Author: Matlabsite
Title: "HDL coder documentation download"
Created on: unknown
Available at: http://www.matlabsite.com/4516/hdl-coder-documentation-download.html
Viewed on: 2 September 2015

[10] Author: MATLAB answers; Sam
Title: " Simulink Library from my own Block created using my hand-written HDL code "
Created on: 24 January 2012
Available at: http://www.mathworks.com/matlabcentral/answers/26950-simulink-library-from-my-own-block-created-using-my-hand-written-hdl-code
Viewed between: 27 August 2015 and 28 August 2015

[11] Author: Altera
Title: Black-Boxing in DSP Builder, version 1.0
Created in: October 2005

[12] Author: The MathWorks Inc.
Title: "HDL Verifier"
Created on: unknown
Available at: http://nl.mathworks.com/products/hdl-verifier/
Viewed on: 28 August 2015

[13] Author: YouTube; DSPMinion
Title: "FPGA: Altera Advanced DSP Builder 8.0 Demo Section 1 of 3 "
Created in: May 2008
Available at: https://www.youtube.com/watch?v=dSxqM7S2upA
Viewed on: 28 August 2015

[14] Author: The MathWorks Inc.
Title: " Using Altera DSP Builder Advanced Blockset with HDL Coder"
Created on: unknown
Available at: http://nl.mathworks.com/help/hdlcoder/examples/using-altera-dsp-builder-advanced-blockset-with-hdl-coder.html
Viewed on: 28 August 2015

[15] Author: The MathWorks Inc.
Title: "Create Xilinx FPGAs and Zynq SoCs "
Created on: unknown
Available at: http://nl.mathworks.com/solutions/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html
Viewed on: 28 August 2015

[16] Author: Altera Forum; Toks
Title: "Information needed: Altera DSP library"
Created in: May 2012
Available at: http://www.alteraforum.com/forum/showthread.php?t=34524
Viewed on: 28 August 2015

[17] Author: Xilinx User Community Forum;schmitt.maximilian1@web.de
Title: "Matlab Simulink black box sub module"
Created on: 17 December 2013
Available at: http://forums.xilinx.com/t5/New-Users-Forum/Matlab-Simulink-black-box-sub-module/td-p/393317
Viewed on: 27 August 2015

[18] Author: StackOverflow; skaffman
Title: "s-function in simulink MATLAB"
Created on: 21 January 2012
Available at: http://stackoverflow.com/questions/8953759/s-function-in-simulink-matlab
Viewed on: 27 August 2015

[19] Author: Opli
Title: "New Adapter Connects BitFlow Frame Grabbers with Mathworks Image Acquisition Toolbox Software"
Created on: 16 August 2012
Available at: http://www.opli.net/magazine/imaging/2012/bit_flow_matlab.aspx
Viewed on: 31 August 2015

[20] Author: The MathWorks Inc. (March 2015)
Title: Image Acquisition Toolbox Adaptor Kit - User's Guide, version 1 release 2015a
Created in: September 2005
Revised in: September 2015

Appendix II. Selecting a CoaXPress card

Number of Pages:	5
Description:	This document describes the process of selecting a CoaXPress card and CoaXPress IP.
Note:	This document was written in the last full week of August 2015. The delivery times and product specs may now be different.

Selecting a CoaXPress card

In order to connect the camera's currently in use on the demonstrator system, two different interfaces are needed. One camera, the Baumer HXC 13, uses CameraLink. The other camera, the Optronis CP80-3-M/C-540 uses CoaXPress. A CameraLink card has already been ordered but for the CoaXPress there is still some research to be done. The internet has been searched for interface cards that have or support CoaXPress. At the end of the search nine cards were found. For a list of these cards, please see table 1 on page 4. The items on this list have been compared against one another depending on the specifications below:

- Interface support
- Speed of the images
- Power-over-CoaXPress
- Number of inputs
- Image sizes
- Memory size
- User-programmable FPGA on print
- Availability of manufacturers HDL code (IP)
- Output protocol

After this comparison the Matrox Radient eV-CXP and the Euresys COAXLINK Quad G3 could be eliminated. Both those cards have no FPGA and their output is PCIe, which means that they cannot be easily connected to an FPGA. Both cards were added to the comparison to see how the "FPGA-boards" compare to the "non-FPGA-boards". The first one of the two does have a small plus: it is available in both a CameraLink and a CoaXPress version, meaning code written for that specific card does not need to be rewritten when switching between the two protocols.

Because an on-card FPGA is desired the availability of HDL source code of the IP's and their language have been looked at. In the datasheets and product overviews this is only mentioned for the Techway Highs speed video treatment CXP. To get this information for the other cards a bit more research was needed. For the Gidel Proce V has no HDL code examples could be found. Aside from the product brief nowhere is it mentioned this board can be used with OpenCL. Also there are no HDL examples available nor is there a detailed description of its hardware[1] available. The only advantage this card has is that it has room for both CoaXPress and CameraLink expansion cards.

For both KAYA instruments Open FPGA Platform cards, no hardware is available. However there is correspondence with a KAYA representative about this.

For the Gidel CoaXPress PCIe Frame Grabbers is also no hardware available though they sell a Proc Dev Kit with which they also sell an IP licence for one year.

Apart from the cards with a PCIe output there are also two cards with an FMC connector. These cards are the FMC CXP 4I1O from Sensor to image and the FPGA Mezzanine Card for CoaXPress from KAYA instruments.

The card from Sensor to image has IP available, but for a limited number of FPGAs and they use a softcore. They sell the card via E-bay which causes some doubts about the reliability. The other card, FPGA Mezzanine Card for CoaXPress (KAYA), has a hardware manual available as well as an IP. The manual cannot be downloaded freely but will be supplied with the card.

[1] Here the non-configurable parts of the card are meant.

The last card is the High Speed Mezzanine Card (HSMC) for CoaXPress(KAYA). This card has, as well as the FPGA Mezzanine Card for CoaXPress(KAYA), a hardware manual and an IP available.

Because of the less then optimal experience with Gidel in the past all their products are dropped. Taking this all into consideration, only the Techway High speed video treatment CXP card and the four KAYA cards remain. A good thing to note is the inconsistency in the documentation of the High Speed Mezzanine Card(KAYA). It has four inputs according to the product brief, but ordering of this configuration not possible. In practice this causes four of the inputs to remain unused. The advantage of the FPGA Mezzanine Card for CoaXPress is that the card is future proof because the connectors on the card already comply with the vision on the connectors and cables as defined in the CoaXPress Roadmap.

To further reduce the selection the specifications of the camera are compared against the remaining cards. The camera has a speed of up to 25 Gbps with four channels in use. With one channel this is 6.25 Gbps. This means the cards must have at least four inputs, each with a speed of 6.25 Gbps, to be able to make the most of the camera. All remaining cards comply and the camera poses no further limitations.

Looking at pictures of all of the cards, those pictures that are available at least, it seems the FPGA Mezzanine Card for CoaXPress(KAYA) is used in the solution from Techway (please see figure 1). It is hard to say for sure with just the information from the datasheets because the FPGA itself also determines some of the characteristics. If this is the KAYA product, this means KAYA is, at the moment, the only manufacturer supplying products at the required quality level.

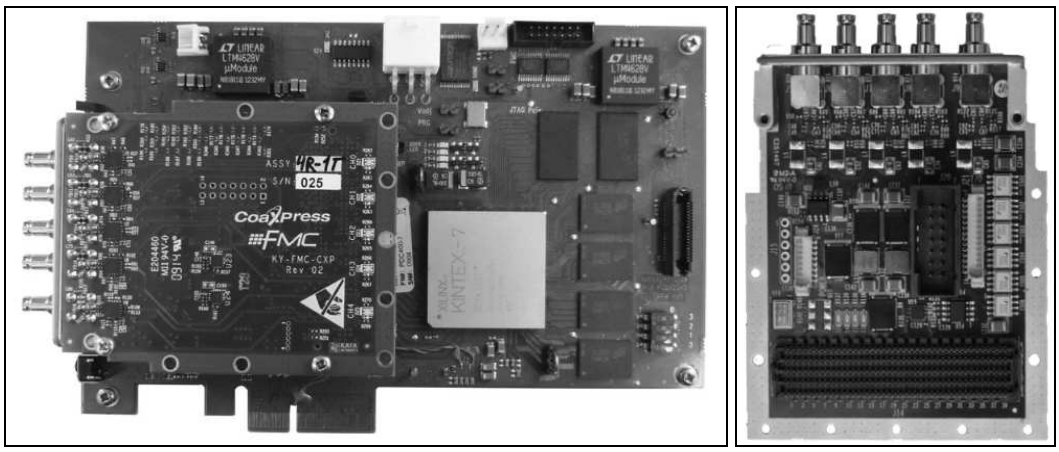


Figure 1. The board from Techway (left) with the KAYA FMC card (right)?

The camera's specifications have not led to a decisive answer. This means the pricing and delivery times start to play a role. It has been decided to contact Techway for pricing and delivery times[2] of the cards. For KAYA this contact has already been made and the delivery time of the Komodo CoaXPress Open FPGA Platform card is a week, and six weeks for the FPGA Mezzanine Card for CoaXPress.

For the FMC en HSMC cards KAYA sells adapters from HSMC to FMC and vice versa. This enables connecting the cards to both Xilinx and Altera boards.

[2] In September of 2015.

Addendum 28 August 2015:
The representative of KAYA stated KAYA needs more time to work on the documentation. This is not good for the confidence in the company.

Techway has not yet reacted to inquiries in the pricing and delivery times. At the next moment of contact the support and availability of IP needs to be asked after.

Addendum 31 August 2015:
During the research into the expansion options of Simulink it was found the drivers of the card are an important factor in the connection between the board and Simulink[3]. This connection is put between the drivers and Simulink. For some framegrabbers and cameras these connections are already made. It might be a good idea to ask if this is also the case for the selected cards.

Contact with Techway hinted they did not have their solution marked-ready. There is still some e-mail contact with Techway about the delivery time.

Addendum 1 September 2015:
Because the delivery times of both KAYA and Techway are very vague sensor to image has become an option again. They sell an expansion card, but there is no documentation available. They also sell IP. This IP has been prepared for speeds up to 6.25 Gbps, but in the accompanying text they mention speeds of 6.125 Gbps. The correct number must be asked. Apart from the card a third-party FPGA board needs to be bought. Usage of the, already available, Spartan-6 board might be possible[4], but only one lane will be supported.

Addendum 18 November 2015:
Kaya won't give any details about their IP. They have provided a QSF file and they say that ought to be enough to tell if it is does meet expectations. This is not sufficient. The information needed is not written in the file and it looks like KAYA won't be an option much longer.

Sensor-to-image has provided pricing and the price they ask is way out of the budget, so that option is dropped.

Techway does not reply to queries and does not ring back despite promises to do so. This does not suggest they are serious about the card and thus this option is dropped.

[3] Only when using Simulink on a PC with part of the calculations and/or IO on the external board.
[4] The accompanying text specifies the SP605 development board. The available resources need to be looked at.

Manufacturer	KAYA instruments	KAYA instruments	Techway	KAYA instruments	KAYA instruments	Gidel	Gidel	Matrox	Euresys	Sensor to image
Type	Komodo CoaXPress Open FPGA Platform	Gecko CoaXPress Open FPGA Platform	High speed video treatment CXP	FPGA Mezzanine Card for CoaXPress	High Speed Mezzanine Card (HSMC) for CoaXPress	CoaXPress PCIe Frame Grabbers	ProceV (CoaXPress and CameraLink via expansion card)	Radient eV-CXP	COAXLink Quad G3	FMC CXP 4I1O
# Inputs	8	4	4	5	4	12		4	4	4
POCE	13 W/link	13 W/link	Yes	Yes	Yes			Yes		
Image format						Bayer, RGB, RGBA, YUV, Mono, Planer, etc.		Bayer(2x2 average) interpolar, colorspace conversion	Bayer, RGB	
Speed	6,25 Gbps/link	6,25 Gbps/link	6,25 Gbps/link	6,25 Gbps/link	6,25 Gbps/link	70 or 75[5] Gbps		6,25 Gbps/link	25 Gbps	
/link calculated						5,83 or 6,25 Gbps			6,25 Gbps	
Memory	144 Gb max	4 GB	128 MB	-	-	16 GB		1 GB	1 GB	
OS drivers			Windows, Linux 32&64							
FPGA intern	Yes	Yes	Yes	No	No	Yes		No	No	No
Manufacturer FPGA	Altera	Altera	Xilinx			Altera				
FPGA	Arria V GZ	Arria V (GXMA?)	Kintex-7 (KX325 or KX410)		0	Stratix (III,IV,V)				
HDL code available			Yes, VHDL				No, via OpenCL			
Output protocol	PCIe	PCIe or stand-alone	PCIe	FMC	HSMC	PCIe				FMC

Table 1. Criteria for comparison.

[5] Both values are mentioned in datasheet, 75Gbps is most likely.

KAYA instruments Komodo CoaXPress Open FPGA Platform
http://www.kayainstruments.com/komodo-coaxpress-open-fpga-platform/

KAYA instruments Gecko CoaXPress Open FPGA Platform
http://www.kayainstruments.com/gecko-coaxpress-open-fpga-platform/

KAYA instruments FPGA Mezzanine Card for CoaXPress
http://www.kayainstruments.com/fmc-coaxpress/

KAYA instruments High Speed Mezzanine Card (HSMC) for CoaXPress
http://www.kayainstruments.com/hsmc-coaxpress/

PLDA and KAYA Instruments announce a high performance CoaXPress system
http://www.kayainstruments.com/plda-and-kaya-instruments-announce-a-tested-high-performance-coaxpress-system/

Gidel CoaXPress PCIe Frame Grabbers
http://www.gidel.com/image-processing/CoaXPress-Frame-Grabber.asp

Techway High speed video treatment CXP
http://www.techway.eu/files/techway/produits/tw-PFC_CXP_datasheet_UK.pdf?PHPSESSID=b262c831956735430af2ee57a02164e4

Gidel ProceV (CoaXPress and CameraLink via expansion card)
http://www.gidel.com/image-processing/CoaXPress-Frame-Grabber.asp

Matrox Radient eV-CXP
http://www.matrox.com/imaging/en/products/frame_grabbers/radient_ev/radient_ev_cxp/Euresys

Coaxlink Quad G3
http://www.euresys.com/Products/CoaXPress/CoaxlinkSeries.asp

Sensor to image FMC CXP 4I1O
http://www.s2i.org/index.php/s2i-online-shop/fmc-development/fmc-coaxpress-4i1o

CoaXPress Roadmap
http://www.coaxpress.com/coaxpress-download.php?file=./coaxpress-documents/CoaXPress-WP-4-CoaXPressRoadmapV10.pdf

CoaXPress Multi-way Connector Proposal
http://www.coaxpress.com/coaxpress-download.php?file=./coaxpress-documents/CoaXPress-Multiway-Connector-Proposal-May2012.pdf

Appendix III. Description of the CameraLink Protocol

Number of pages: 4
Description: This document describes the way the CameraLink protocol works.

Description of the CameraLink protocol

The CameraLink protocol is used to transfer information from image sensors to frame grabbers. The protocol is based on the Channel Link protocol. The protocol transmits 28-bit wide data over one link. The data is serialized with a ratio of 7:1 with the four resulting data streams transmitted in parallel. Apart from these four data lines a clock is also transmitted with the same reduction ratio of 7:1. The CameraLink protocol specifies one, two or three links, four lines for camera control and two lines for a serial protocol which is RS-232 compatible [1].

1 Data transmission

The image is transmitted using a protocol that can be visualised as layers[1] as can be seen in figure 1, where the pixels of an image are the topmost layer. The pixels are then divided over ports and flags are added to signal the state of the data. These are then assigned to links which are then transmitted over the wires. By reversing all the operations the pixels can then be recovered. The actions taken to change into the next layer are given in figure 2.

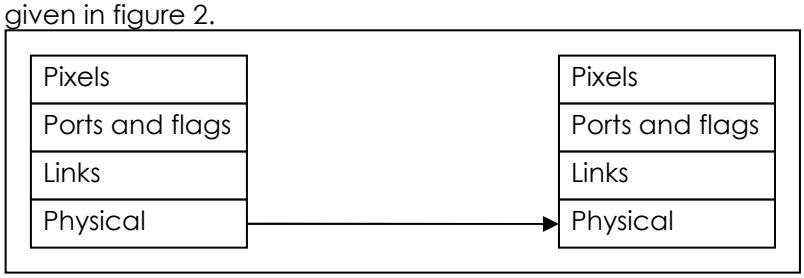


Figure 1. Layer model of the CameraLink protocol.

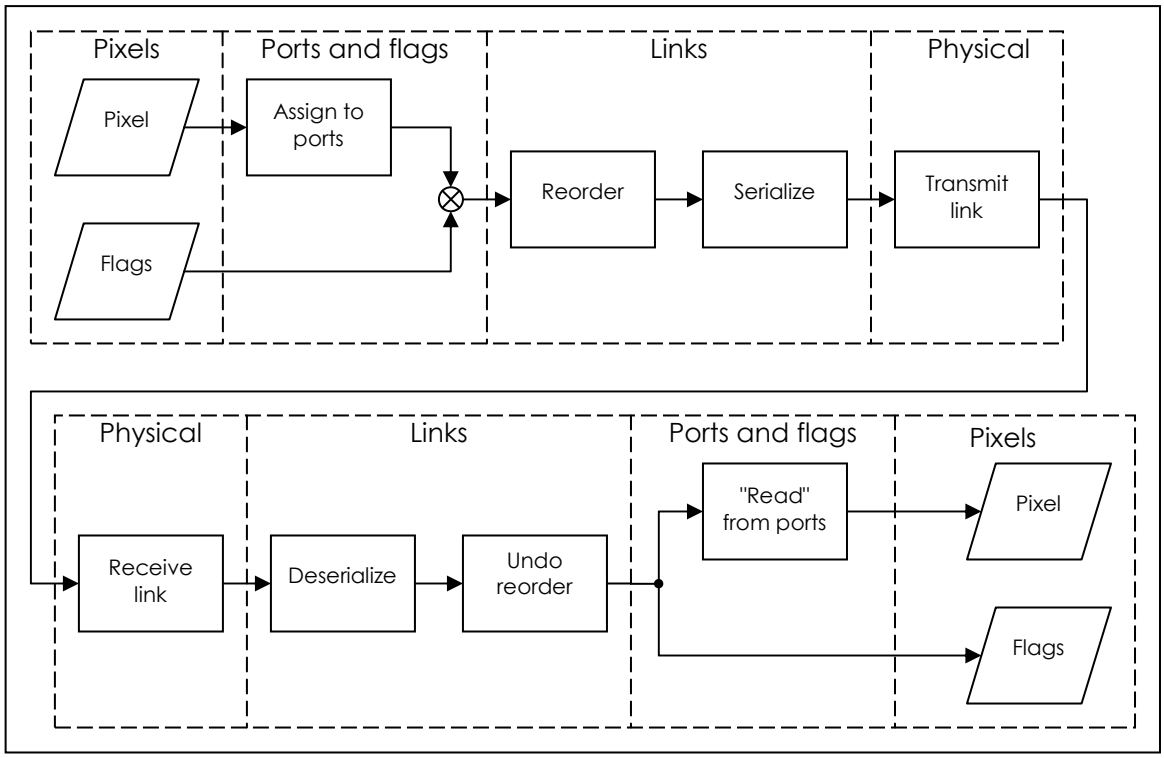


Figure 2. Actions to transfer to the next layer, mapped to the layer model of the CameraLink protocol.

[1] There is no documentation that uses a layer model, but it helps in explaining what happens. One document ([1]) uses "physical layer" as an alternative name for the wires.

2 Layers explained

As stated in footnote 1 (on page 1), the layers are not used in any documentation, but it helps greatly in explaining what happens on the protocol. Next follows a short explanation per layer, with the exception of the physical layer.

1.1 Pixels

This layer is rather straightforward, it has the pixels of the image.

1.2 Ports and flags

The ports, or taps as they are called in other protocols, have the information of one or more pixels mapped to one-byte wide ports. The mapping depends on the configuration. The flags are added to indicate the position and status of the pixel. The standard does not specify the correct behaviour. Some cameras do not use all flags. Others use them, depending on the camera settings, only in some configurations [5][6][7].

The flags are:

- Dval; The data valid flag. This flag is high if the data is valid.
- Fval; The frame valid flag. This flag is high if the line is valid.
- Lval; The line valid flag. This flag is high when the pixels are valid.
- Spare; The spare flag. This flag currently has no purpose, but it might be given one in future versions.

When sending four frames of a 4×4 picture the flags will behave as shown in figure 3 below. The data valid flag is not always used. Some cameras don't use it at all, with others the flag is always high to indicate the camera is active. How the pixels are mapped to the ports depends on the configuration. The manual[2] gives a good description on this point thus they are not listed here.

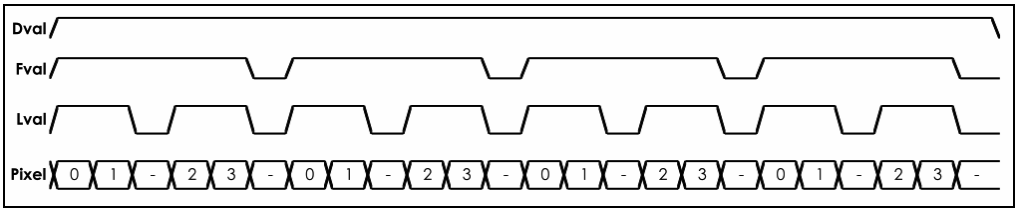


Figure 3. Behaviour of the flags.

1.3 Links

The ports are mapped to 28-bit wide links, each with a minimum of one and a maximum of three links. How the ports are mapped to the link is listed in table 1. The reason for the permutating of the bits is unclear. Although some speculate it has something to do with lowering crosstalk or transmission power [2]. When the bits are transmitted they are permutated even further. Figure 4 shows this permutation and how they map to the bits of the ports and flags.

Input Signal Name	28-bit Solution Pin Name
Strobe	TxClk Out/TxClk In
LVAL	TX/RX 24
FVAL	TX/RX 25
DVAL	TX/RX 26
Spare	TX/RX 23
Port A0, Port D0, Port G0	TX/RX 0
Port A1, Port D1, Port G1	TX/RX 1

[2] Specifications of the CameraLink Interface Standard for Digital Cameras and Frame Grabbers [1].

Input Signal Name	28-bit Solution Pin Name
Port A2, Port D2, Port G2	TX/RX 2
Port A3, Port D3, Port G3	TX/RX 3
Port A4, Port D4, Port G4	TX/RX 4
Port A5, Port D5, Port G5	TX/RX 6
Port A6, Port D6, Port G6	TX/RX 27
Port A7, Port D7, Port G7	TX/RX 5
Port B0, Port E0, Port H0	TX/RX 7
Port B1, Port E1, Port H1	TX/RX 8
Port B2, Port E2, Port H2	TX/RX 9
Port B3, Port E3, Port H3	TX/RX 12
Port B4, Port E4, Port H4	TX/RX 13
Port B5, Port E5, Port H5	TX/RX 14
Port B6, Port E6, Port H6	TX/RX 10
Port B7, Port E7, Port H7	TX/RX 11
Port C0, Port F0	TX/RX 15
Port C1, Port F1	TX/RX 18
Port C2, Port F2	TX/RX 19
Port C3, Port F3	TX/RX 20
Port C4, Port F4	TX/RX 21
Port C5, Port F5	TX/RX 22
Port C6, Port F6	TX/RX 16
Port C7, Port F7	TX/RX 17

Table 1. CameraLink bit assignment [1].

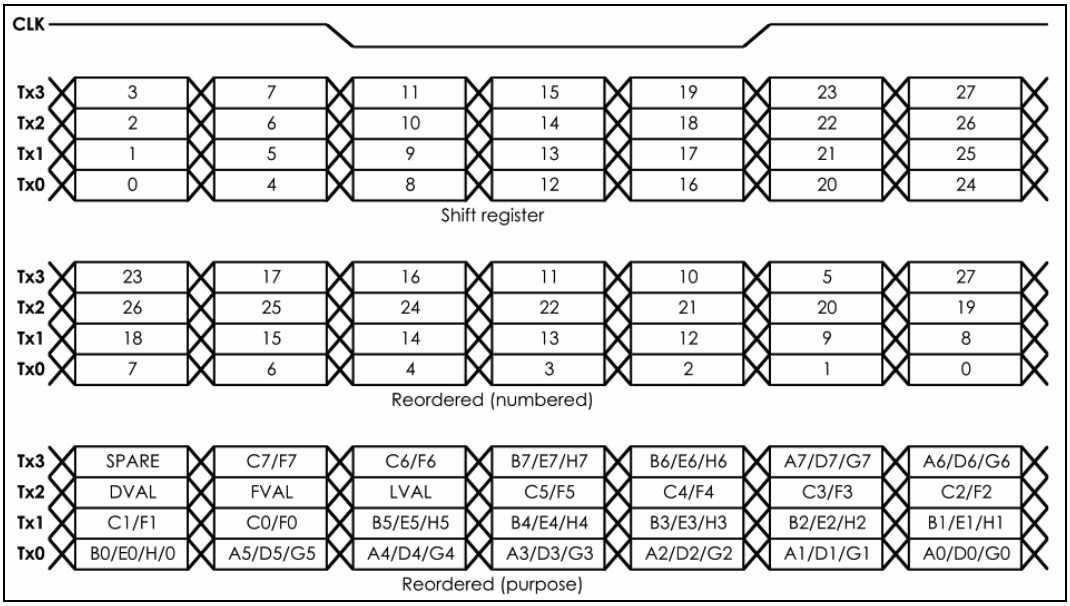


Figure 4. Bit reception and reordering [3].

3 CameraLink modes

The protocol can transmit the data using one of three modes:

- Base; which uses one cable and can transmit one link.
- Medium; which uses two cables and transmits two links.
- Full; which uses two cables and transmits three links.

Some manufacturers use an extended mode as well using some unused bits and repurposing some flags. Thus a few extra bits can be transmitted [1][2][4].

4 Sources

[1] Author: CameraLink
Title: Specifications of the CameraLink Interface Standard for Digital Cameras and Frame Grabbers
Created in: October 2000

[2] Author: Volker Schatz
Title: The Camera Link camera interface
Created on: Unknown
Available at: http://www.volkerschatz.com/hardware/clink.html
Viewed between: 2 September and 8 October

[3] Author: Texas Instruments
Title: Texas Instruments DS90CR287/DS90CR288A +3.3V Rising Edge Data Strobe LVDS 28-Bit Channel Link - 85MHz)
Created in: October 1999
Revised in: March 2013

[4] Author: National Instruments
Title: What Are The Differences Between Base, Medium, and Full Camera Link Configurations?
Created on: 27 February 2013
Revised on: 24 April 2015
Available at: http://digital.ni.com/public.nsf/allkb/2EE80B8C381D61D286257B1F005674A5
Viewed between: 3 September and 1 October 2015

[5] Author: Baumer
Title: Manual Baumer HXC 13 v14 users guide
Created on: 18 February 2014

[6] Author: Adimec
Title: Manual ADIMEC 4000 M/D
Created on: 6 May 2003

[7] Author: Photon focus
Title: User Manual MV1-D1024E CameraLink Series CMOS Area Scan Cameras September 2014 V1.0
Created in: September 2014

Appendix IV. Description of the HDL implementation for CameraLink

Number of pages: 13
Description: This document is a manual for the CameraLink module and tells how to instantiate the module and gives a description of the parameters.

Description of the HDL implementation for CameraLink.

1 Global description

A CameraLink camera sends pixels in pieces. These parts will have to be reconstructed. The described hardware receives the parts and rearranges the bits into the correct order. The communication links to the camera are directly connected; there is no logic in between other than the conversion from a single-ended to a differential signal (LVDS). These signals are also used to control the trigger indicator. This indicator becomes high if a trigger signal is present. The serial (RS232) signals are not routed trough the module. A simplified block diagram is given in figure 1. An explanation of the hierarchy can be found in chapter 8. Before continuing reading it is recommended to read the Description of the CameraLink protocol [1].

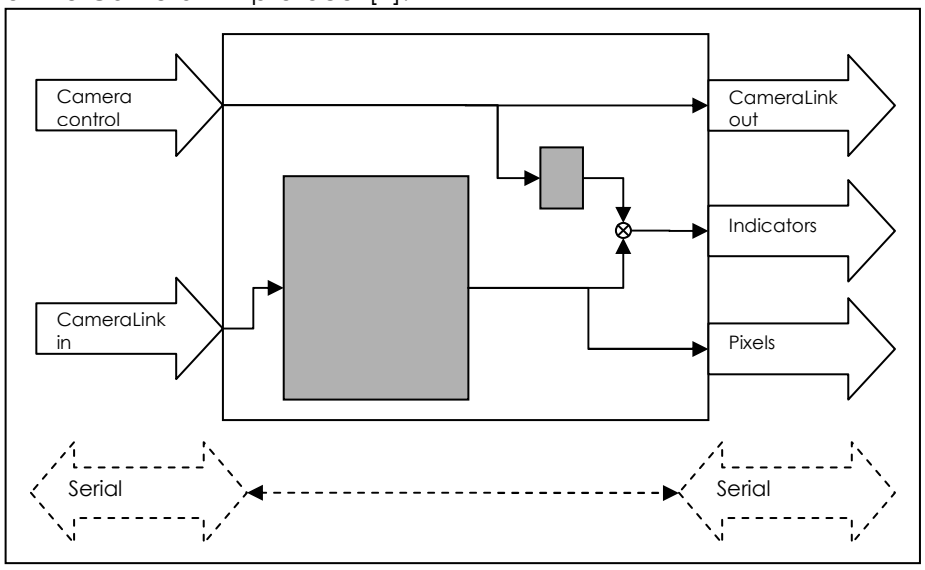


Figure 1. Simplified block diagram.

2 CameraLink card

The CameraLink module has been written with tests run on an Alpha Data FMC-CAMERALINK card which is connected to a Xilinx Spartan 6 development board. The code should port to any other CameraLink card or board, but that board may differ in functionality and/or IO standard. Before implementation one should at least check:

- The voltage levels
- The signalling type (Single-ended or differential)
- The signal polarity
- The frequency of the system clock (at least the speed of the CameraLink data clock, preferably faster)

3 Interface

To be able to connect the CameraLink module with the other hardware the module has some inputs and outputs, as well as some constants. A description of the ports and generics of this interface is given in table 1 below. The allowed values for the signals are, unless otherwise specified, the values according to the VHDL standard and active-high.

	Name	**Type**	**Meaning**
Constant	system_clock_frequency	natural	The speed of the system clock in Hz
Constant	CameraLink_frequency	natural	The speed of the CameraLink data clock in Hz
Constant	CameraLink_mode	CameraLinkMode (BASE,MEDIUM,FULL)	The mode of the CameraLink
Constant	CameraLink_numchannels	natural (1 to 8)	The number of channels(pixels) per link
Constant	CameraLink_numbits	natural (8 to 36)	The number of bits per channel
Constant	enable_PoCL	boolean	Activate power over CameraLink[1]
Constant	camera_framerate	natural	The frame rate of the camera in frames per second
Constant	differential_signalling	boolean	Are the inputs to the module LVDS signals?
Constant	trigger_index	natural (0 to 3)	The position of the trigger signal in the camera control bits
Constant	trigger_high	time	The time during which the trigger is high
Constant	num_buffers	CameraLinkNumberOfBuffers (Auto,1,2)	The number of buffers used for each link synchronizer
Constant	chip_family	string (Xilinx_Spartan_6)	The name of the chip family
Constant	is_simulation	boolean [default: false]	Is the module compiled for simulation?
Input	clock	std_logic	The input for the system clock
Input	reset	std_logic	The input for the asynchronous reset
Input	x_clock_p	std_logic	The data clock of link X (positive line)
Input	x_clock_n	std_logic	The data clock of link X (negative line)
Input	x_data_p	std_logic_vector (3 downto 0)	The data of link X (positive line)
Input	x_data_n	std_logic_vector (3 downto 0)	The data of link X (negative line)

[1] Only if the card supports PoCL.

	Name	Type	Meaning
Input	y_clock_p	std_logic	The data clock of link Y (positive line)
Input	y_clock_n	std_logic	The data clock of link Y (negative line)
Input	y_data_p	std_logic_vector (3 downto 0)	The data of link Y (positive line)
Input	y_data_n	std_logic_vector (3 downto 0)	The data of link Y (negative line)
Input	z_clock_p	std_logic	The data clock of link Z (positive line)
Input	z_clock_n	std_logic	The data clock of link Z (negative line)
Input	z_data_p	std_logic_vector (3 downto 0)	The data of link Z (positive line)
Input	z_data_n	std_logic_vector (3 downto 0)	The data of link Z (negative line)
Input	camera control	std_logic_vector (3 downto 0)	Camera control signals
Output	cc_data_p	std_logic_vector (3 downto 0)	Camera control (positive line)
Output	cc_data_n	std_logic_vector (3 downto 0)	Camera control (negative line)
Output	data_valid	std_logic	Data flag from the camera
Output	frame_valid	std_logic	Frame valid flag from the camera
Output	line_valid	std_logic	Line valid flag from the camera
Output	spare	std_logic	Spare flag from the camera
Output	pixelstream	std_logic_vector (Max: 63 downto 0 Min: 7 downto 0)	Output of the pixel values (Size depends on the number of channels and bits)
Output	new_data	std_logic	Indicator new data available, strobe
Output	new_pixel_indicator	std_logic	Indicator new pixel available, longer pulse
Output	connected	std_logic	Indicator camera connected
Output	Enable_PoCL	std_logic	Indicator power active
Output	trigger_indicator	std_logic	Indicator camera control trigger command

Table 1. Table with the CameraLink interface.

It is possible to assign the constants in such a manner that the resulting configuration is not recognised in the standard. If this is the case the hardware cannot be generated. An error message will be shown during both HDL simulation and synthesis and the process will stop.

3.1 The constants explained

The interface has multiple constants. Using these constants the hardware can be configured to handle the mode and the number of bits per channel. This paragraph will give a short explanation for each constant.

system_clock_frequency
This constant is used to set up the timer range for the trigger indicator.

CameraLink_frequency
This constant is used to set up the timer range for the new pixel indicator.

CameraLink_mode
This constant is used to set the mode of the CameraLink.

CameraLink_numchannels
This constant is used to determine the number of bits per pixel and map the ports to the pixel.

CameraLink_numbits
This constant is used to determine the number of bits per pixel and map the ports to the pixel.

enable_PoCL
This constant activates power over CameraLink, this only works if the card supports this. Elsewise this does nothing, but it is incorporated in the design and will be synthesized.

camera_framerate
This constant is used to set up the timer range for the trigger indicator.

differential_signalling
This constant is used to determine if buffers are needed to convert the differential signals into single-ended signals.

trigger_index
This constant sets the position of the trigger indicator in the camera control bits. This ins only used for reading the signal, not to drive it.

num_buffers
This constant determines the number of buffers that are to be used to enable a lower system clock frequency.

chip_family
This constant is used to determine which HDL description is needed for the PLL and the SERDES. These descriptions are either board or vendor specific. Changing only this generic may not be enough. Some selection is also made in the CameraLink library.
The chip families below are supported:

- Xilinx Spartan 6

is_simulation
This constant is used to change the behaviour of some components so that the simulation matches the reality.

3.2 The inputs explained

The interface has multiple inputs. Per input a short description is given.

clock
This is the input for the system clock.

reset
This is the input for the reset. The reset is handled asynchronously. The upper layer(s) of the hierarchy are responsible for the correct implementation of a reset circuit.

x_clock_p & x_clock_n
The clock input of link X, coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

x_data_p & x_data_n
The four data inputs of link X, coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

y_clock_p & y_clock_n
The clock of link Y, coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

y_data_p & y_data_n
The four data inputs of link Y, coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

z_clock_p & z_clock_n
The clock input of link Z, coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

z_data_p & z_data_n
The four data inputs of link Z, link coming from the CameraLink connector. It expects an LVDS signal, unless the differential_signalling constant is set to false.

camera_control
The four bits that control the camera. The module converts these signals to LVDS signals.

3.3 The outputs explained

The interface has multiple outputs. Per output a short description is given.

cc_data_p & cc_data_n
The four output bits for the camera control signals, four LVDS pairs. This is does not depend on the differential_signalling constant.

data_valid
The data valid flag from the last transmitted data. If multiple channels are used this is the result of the multiplication (AND) of the data valid flags of all channels in use.

frame_valid
The frame valid flag from the last transmitted data. If multiple channels are used this is the result of the multiplication (AND) of the frame valid flags of all channels in use.

line_valid
The line valid flag from the last transmitted data. If multiple channels are used this is the result of the multiplication (AND) of the line valid flags of all channels in use.

spare
The spare flag from the last transmitted data. If multiple channels are used this is the result of the multiplication (AND) of the spare flags of all channels in use.

pixelstream
Depending on the configuration this outputs the value of one or more pixels, with the first pixel on the first bits. The data changes on the rising edge of the new_data output.

new_data
This output is strobed for one pulse of the system clock, and indicates the arrival of new data.

new_pixel_indicator
This output indicates the availability of a new pixel. This signal keeps active until the next frame should arrive. This time is calculated using the system_clock_frequency constant, the trigger_high constant and the camera_framerate constant.

connected
This output indicates a camera is recognised by the module. This is done by looking at the availability of the input clock(s).

Enable_PoCL
This output is a constant-driven output, with enable_PoCL being that constant. It can be used to drive the power over CameraLink some boards provide.

trigger_indicator
This output becomes active high if a trigger pulse is given and keeps high until the next trigger pulse should arrive. This time is calculated using the camera_framerate and trigger_high constants.

4 Instantiation

When the CameraLink module is used in designs the designer will have to instantiate the module with the interface as given described in chapter 3. Also the CameraLink_lib library will have to be edited. This library is used to include libraries that very per manufacturer. These libraries are selected by commenting and uncommenting the inclusion and aliases depending on the chip family.

5 Synchronization

The reception of the data is done using the data clock. This clock is multiplied using a PLL, which makes the clock have the same frequency as the data changes. The data is read on this generated clock and then, once a complete "package" has been received, it is clocked trough three registers to synchronise it to the system clock.

5.1 Skew between links

Due to a variation in cable lengths between the two connectors the data will become skewed. In simulation a skew has been simulated between the links x and y as well as between the links x and z. All varying between minus eight and positive eight data clock cycles. In simulation the module was able to recover the data when a skew of 3[2] bit-times is given. Outside this limit the module cannot deliver the correct data. The delayed data will be put into the next pixel.

6 Testbench

The testbench has been designed to use a number of constants to easily test different scenarios. The testing of these scenarios can be handled automatically. This can be done because the testbench encloses the module. The testbench generates data and control signals and feeds these to the module. The outputs of the module are read by the testbench and checked on their correct value. Using asserts and reports the testbench will tell the developer what happens when a fault has been found and the entire simulation will stop with a failure on that error. The parameters of the testbench are given in table 2.

Note: The parameters are put in two locations, a number of lines apart, in the testbench file. This is done because a number of the simulation constants are calculated based upon previously set values. These results can be used to enter values in the other series of constants.

	Name	**Type**	**Meaning**
Constant	VHDL_standard	integer	The VHDL standard used for simulation. If unknown, use 1993.
Constant	fail_simulation_to_stop	boolean	Should the simulation use a fail to stop? This is only used if the VHDL_standard <2008.
Constant	CameraLink_mode	CameraLinkMode (BASE,MEDUIM,FULL)	Which mode needs to be simulated?
Constant	CameraLink_numchannels	natural (1 to 8)	How many channels are used?

[2] This simulation was done on an earlier version of the CameraLink module, and this value may now be incorrect.

	Name	Type	Meaning
Constant	CameraLink_numbits	natural (8 to 36)	How many bits are used per channel?
Constant	system_clock_frequency	natural	What is the frequency of the system clock in Hz?
Constant	chip_family	string (Xilinx_Spartan_6)	Which chip family needs to be simulated?
Constant	CameraLink_frequency	natural	What is the speed of the CameraLink data clock in Hz?
Constant	additional_output_delay	integer	Depending on the used components the delay in the module may vary. This is used to correct for that.
Constant	differential_signalling	boolean	Are the inputs to the module LVDS signals?
Constant	trigger_index	integer (0 to 3)	What is the position of the trigger signal in the four camera control signals?
Constant	trigger_ high	time	For how long is the trigger high?
Constant	framerate_camera	natural	What is the frame rate of the camera in fps (Hz)?
Constant	enable_PoCL	boolean	Should the power over CameraLink be used?
Constant	num_buffers	CameraLinkNumber OfBuffers (Auto,1,2)	The number of buffers used for each link synchronizer
Constant	skew_x_to_y	time	How big a skew needs to be used between link X and link Y?
Constant	skew_x_to_z	time	How big a skew needs to be used between link X and link Z?
Constant	connected_rising_ actual_to_expected	time	Maximum time for 'A' in figure 2 on page 10.
Constant	connected_falling_ expected_to_actual	time	Maximum time for 'B' in figure 2 on page 10.
Constant	connected_rising_ expected_to_actual	time	Maximum time for 'C' in figure 2 on page 10.
Constant	connected_falling_ actual_to_expected	time	Maximum time for 'D' in figure 2 on page 10.

Table 2. Table with the constants for simulation.

6.1 The constants for simulation explained.

A number of these constants are also used in the interface of the CameraLink module and are therefore not discussed here. Please see paragraph 3.1 for the explanation of these constants.

VHDL_standard
The number of the VHDL standard supported by the simulator. With VHDL-2008 a number of simulation decisions and waiting times can be implemented in a nicer way when compared to earlier versions. This constant is used to determine if these options can be used.

fail_simulation_to_stop
In order to stop the simulation once it has finished a report with severity-level failure can be "sent to the simulator". This will cause the simulator to stop. A disadvantage of this is that the simulation will not be able to continue. VHDL-2008 has a better option for this. Thus this constant will be ignored when the VHDL_standard is set to a value >= 2008.

additional_output_delay
Depending on the components used the delay varies. The testbench does not know this delay and assumes the delay is equal to the number of synchroniser steps, which is fixed to 3, multiplied by the number of buffers used. This is not always accurate and this constant is used to compensate for the miscalculation.

skew_x_to_y[3]
In order to simulate a skew between the links it is possible to enter a skew between the links X and Y. Negative values are allowed. This will cause link Y to start transmitting before link X.

skew_x_to_z[3]
In order to simulate a skew between the links it is possible to enter a skew between the links X and Z. Negative values are allowed. This will cause link Z to start transmitting before link X.

connected_rising_actual_to_expected[3]
Because the connected flag depends on the lock signal of the PLL the flag does not always rise the moment the clock becomes active. This constant sets the negative slack allowed for the flag. For more clarity please see 'A' in figure 2.

connected_falling_expected_to_actual[3]
Because the connected flag depends on the lock signal of the PLL the flag does not always fall the moment the clock becomes inactive. This constant sets the positive slack allowed for the flag. For more clarity please see 'B' in figure 2.

connected_rising_expected_to_actual[3]
Because the connected flag depends on the lock signal of the PLL the flag does not always rise the moment the clock becomes active. This constant sets the positive slack allowed for the flag. For more clarity please see 'C' in figure 2.

connected_falling_actual_to_expected[3]
Because the connected flag depends on the lock signal of the PLL the flag does not always fall the moment the clock becomes inactive. This constant sets the negative slack allowed for the flag. For more clarity please see 'D' in figure 2.

[3] These can use calculated constants to be assigned a value.

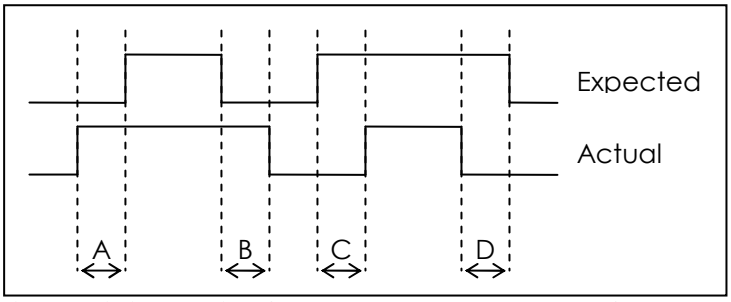


Figure 2. Timing locations for the checks on the connected flag.

Apart from the constants mentioned before some constants are calculated. They are used in the simulation, but can also be used to calculate values entered by the user. (An example of this is shown in code fragment 1.) These constants are:

clockPulse
The period of the system clock.

camera_frame_duration
The period of one frame.

trigger_hold_time
The trigger indicator keeps high during this time

CameraLink_frequency
The frequency of the CameraLink data

CameraLink_bittime
The time for which the four bits stay on the data lines.

```
constant skew_x_to_y : time := CameraLink_bittime*5;
```

Code fragment 1. Example of using a calculated constant in the assignment of a value to another constant.

7 Delays

Processing the data takes time. These delays are given in clock cycles because those are the most significant delays and the other propagation delays depend on the design and on the FPGA. The delay times depend on the components used and the speed of the CameraLink data transfer. Thus they are not given here. The most significant delay is caused by the synchronizer and the deglitcher. Depending on the configuration this takes four to five cycles of the system clock. The synchronizer delay of 4 cycles also applies to the indicator signals with the exception of the connected signal. This signal depends on the PLL locked flag. Depending on the PLL used this flag may rise a little after a data clock signal is present, and it will fall a little after the data clock has been lost.

8 Hierarchy

The CameraLink module consists of a number of submodules. In this chapter each of the submodules will be discussed. A schematic overview of the connections in the module, which is configured for mode full, is given in figure 4 (on page 12). The given connections are for CameraLink mode full. The number of CameraLink_link modules depends on the mode being used. For clarity the reset signal and the system clock signal have been left out.

8.1 CameraLink_link

This module receives the data from one CameraLink link, handles the bit permutations and extracts the flags. The CameraLink module contains three CameraLink_link modules if used in mode full, two modules if used in mode medium and one link if used in mode base.

8.2 one_shot_hold

This module outputs high when the input value becomes active and holds the high value for a given time after the input becomes inactive (please see figure 3). This hold time is given when this module is instantiated. The CameraLink module contains two one_shot_hold modules. The hold time is determined by constants/generics.

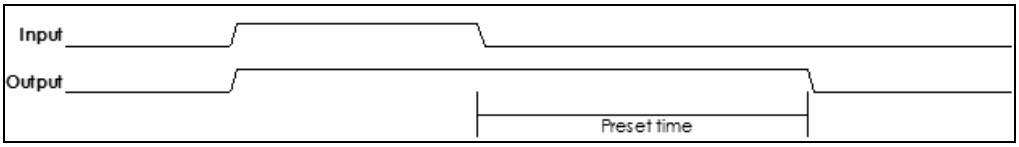


Figure 3. Timing diagram of the one_shot_hold module.

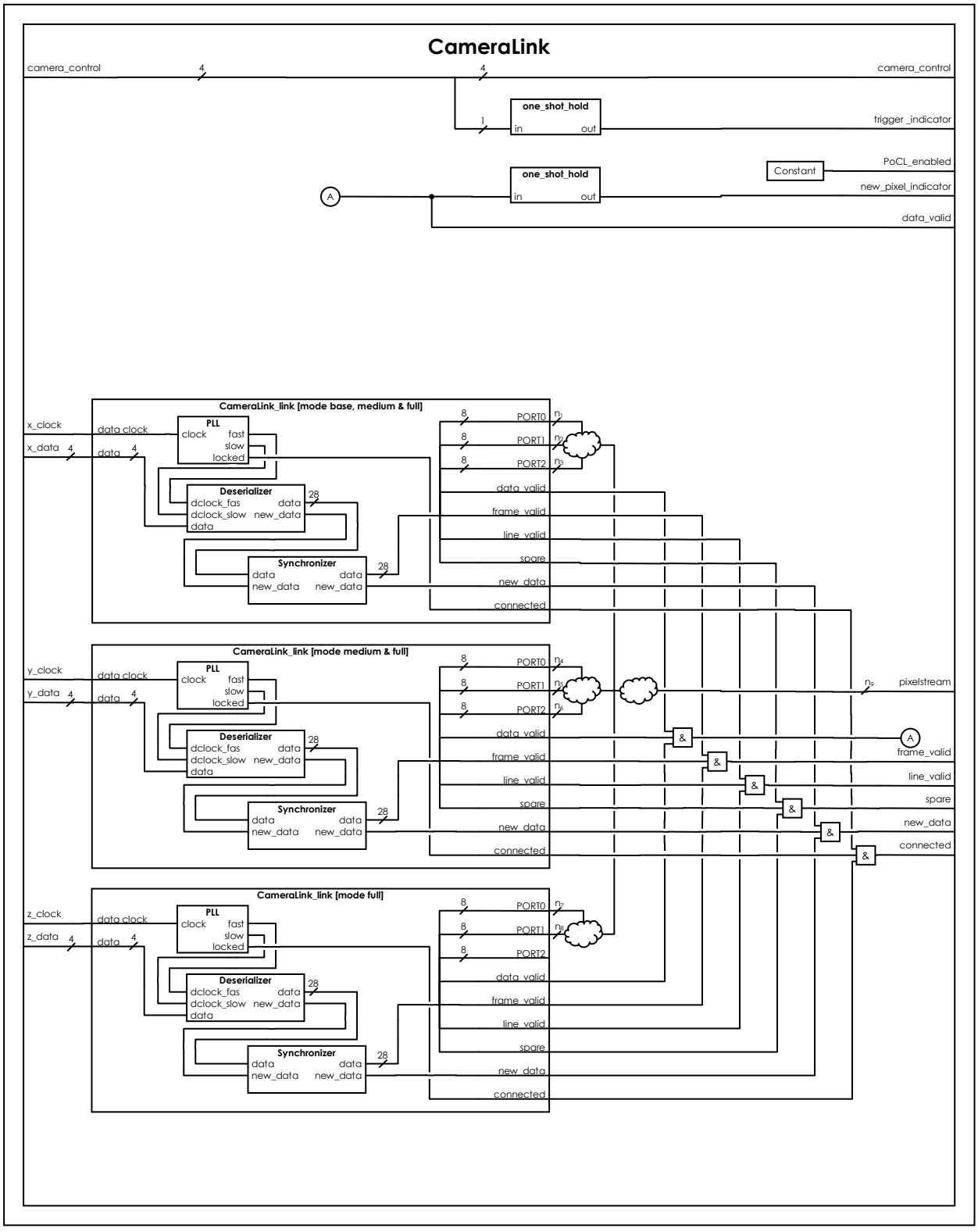


Figure 4. Schematic overview of the connections between the submodules in the CameraLink module mode full.

9 Errors

Because the hardware is conditionally generated it is possible some of the inputs of the CameraLink module are not connected to any logic. Some compilers[4] give warnings for each unused port. These warnings can be ignored.

10 Adding chip family support

In order to get the module to work on a not yet supported board a PLL and a deserializer will have to be designed. The PLL interface has the following inputs and outputs:

- One clock input.
- One output clock with a frequency of seven times that of the input clock.
- One clock output with the frequency of the input clock, in phase with the multiplied clock
- One reset input
- One locked output

The deserializer needs to have a component that takes the LVDS signals and maps them to an internal signal. The other "stuff" for the deserializer is not chip specific. If a chip does not have a deserializer the generic deserializer can be used.

These chip-specific components will have to be placed inside an if-generate statement where the chip_family constant is tested. The file where these additions will need to be placed is CameraLink_link.vhd. Also the buffers in the CameraLink module itself need to be added. Adding comments to the chip_family constants and generate statements is a good thing to do. It is also recommended to update this document.

Some of the components are declared in a library. The CameraLink library handles the inclusion of the correct libraries. If a new board is added the CameraLink library must be adapted to enable the selection of the correct library for the selected chip. This is done by adding the libraries in the list.

Special attention must be taken when the IP infers buffers. Some of these buffers are already instantiated on a higher level. Thus these buffers are not needed and might even cause a compilation failure. In that case they will have to be removed from the lower level.

11 External IP

Some components of the CameraLink module are IP blocks belonging to the chip vendor. Their license allows usage of these cores in the design [2][3]. Some of this IP is edited slightly either to increase efficiency[5] or to enable easier usage higher up in the hierarchy.

The PLL module from Xilinx[6] has been edited slightly to allow a variable value for the input clock period. This value is now determined on instantiation instead of being a fixed number. This might allow for smaller logic surrounding the PLL. It is recommended to do the same when adding a PLL for other chip families.

Some buffers have been removed from the deserializer. These buffers are implemented at a higher hierarchy level and therefore not needed here. If those buffers were not removed they would have been put in series with other buffers, causing errors on

[4] E.g. Xilinx ISE.
[5] A constant is changed into a generic for adaptability.
[6] For the Xilinx Spartan 6.

compilation. The buffers are needed in a higher hierarchy level to prevent unused LVDS signals from causing errors.

12 References

[1] Description of the CameraLink protocol
Pieter van der Star
15 December 2015

[2] LogiCORE IP Clocking Wizard Product Guide
Xilinx inc.
25 July 2012

[3] Xilinx, inc. End user licence agreement
Xilinx corp.
Downloaded from the Xilinx.com website on 21 October 2015

Appendix V. Clock domain analogy

Number of pages: 1
Description: This document gives an explanation of the problems surrounding the crossing from one clock domain into another.

Clock domain analogy

A clock domain may sound like it is something like a time zone, but it is a bit different[1]. It is more like a day on Earth not quite being the same as a day on Mars[2]. When crossing clock domains two problems arise. One being possible metastability, the other being data loss. Data loss is easily explained. Imagine a tennis court. You stand at one side of the court, at the other side a ball machine fires tennis balls at you at a rate such that you can catch them all, one by one, and put them in a bin. Next the machine starts firing the balls at a gradually higher rate. At first you can keep up, until the moment you cannot and you drop some balls. If those balls were to have information on them you would start to lose that information. That is data loss. One part of the system gives information so fast that the other part cannot keep up and loses the data or it may not even know the data was sent.

Explaining metastability requires a bit of philosophy. The ball machine now fires balls at a constant rate. You do not have to catch them, but you say at which half of the court they fall, yours or the ball machine's. When the ball falls in the middle of either halve it is easy to tell, but what if it falls on top of the net. On which half of the field did it fall? You cannot tell, but have to give an answer so you make an educated guess, or ask for a video replay in super-slow-motion in hopes it can help you. A digital system also has that problem, but than with a voltage.

Looking at some standard[3] a logic '0' is defined to be any value between 0.4 V and 0.8 V. A logic '1' is anything between 2.0 V and 2.4 V. (Please see figure 1 for a number line with the values indicated.) So far so good, but physics takes care to mess things up a bit. To change from a '0' to a '1' and vice versa takes time. The values at these "ramps" is undefined (please see figure 2 on page 2). The hardware however is designed to work at the levels of '0' and '1' and the behaviour of these components is thus also undefined, but by design they either output a '0' or a '1'[4]. So their output should be defined. Should, because the recovery takes time as well and, if the next activating clock edge comes too soon, the output will also be metastable. To prevent that from happening at least two registers are needed to ensure stable data. With a combined recovery rate less than the period of the controlling clock[5].

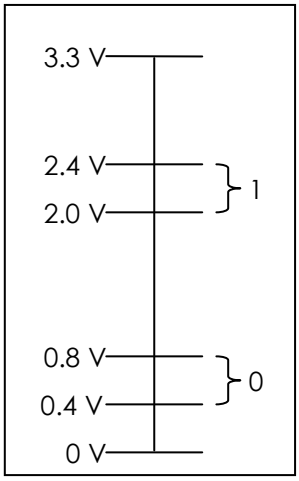


Figure 1. Voltage levels for 3.3 V LVTTL [2].

1 Although the different domains can be "separated" only by a phase-shift.
2 The Mars sidereal day, as measured with respect to the fixed stars, is 24h 37m 22.663s, as compared with 23h 56m 04.0905s for Earth [1].
3 3.3 V LVTTL/LVCMOS.
4 Except tri-state components.
5 With the setup and hold times ignored.

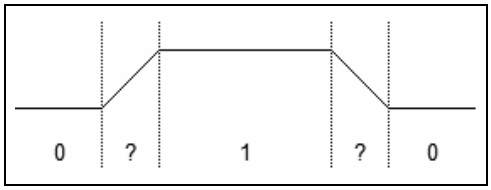


Figure 2. Simple example of metastability.

[1] Authors: NASA Goddard Institute for Space Studies; Michael Allison and Robert Schmunk
Title: Technical Notes on Mars Solar Time as Adopted by the Mars24 Sunclock
Created on: unknown
Updated on: 30 June 2015
Available at: http://www.giss.nasa.gov/tools/mars24/help/notes.html
Viewed on: 16 November 2015

[2] Author: Texas Instruments
Title: Voltage-Level-Translatation Devices
Created in: September 2002

Appendix VI. List of test equipment

Number of pages: 1
Description: This document contains details of all test equipment used.

List of test equipment

Oscilloscope

Manufacturer	: Tektronix
Type	: DPO 4104 Digital Phosphor Oscilloscope
Calibrated	: 05-05-2006
Code	: [TNO] 41120939
Note	: Not used with original probes, all four probes are from different manufacturers and are different types.

Logic analyzers

Manufacturer	: Hewlett Packard
Type	: 16500B Logic analysis system
Calibrated	: 11-10-1994
Code	: [TUI] 16827

Manufacturer	: Xilinx
Type	: Chips Scope Pro Analyzer
Version	: 14.7 P.20131013
Core version	: 1.04a
Calibrated	: N/A
Code	: N/A

Cameras

Manufacturer	: Bauer
Type	: HXC13
Calibrated	: N/A
Code	: N/A

Manufacturer	: Photon focus
Type	: Lingo
Calibrated	: N/A
Code	: N/A

Manufacturer	: Adie
Type	: 4000m/D
Calibrated	: N/A
Code	: N/A

Computer (receiving image over RS-232)

Manufacturer	: Dell
Type	: Optiplex 990
Code	: PC-11879
OS	: Windows 7 Enterprise 64-bit SP1
CPU	: Intel Core i5 2400 @ 3.10 GHz Sandy Bridge 32 nm Technology
Motherboard	: Dell Inc. 0VNP2H (CPU 1)
Serial (RS-232)	: unknown

Development board

Manufacturer	: AVNET
Type	: Xilinx Spartan-6 LX150T PCI Express Development Board
Chip	: Xilinx Spartan-6 XC6SLX150T Fgg676BIV1D37
Serial (RS-232)	: Cypress CP2102

Appendix VII. Sequence of test images

Number of pages: 2
Description: This document contains all the test images next to one another in chronological order. This way the progress can be easily seen.

Sequence of the test images

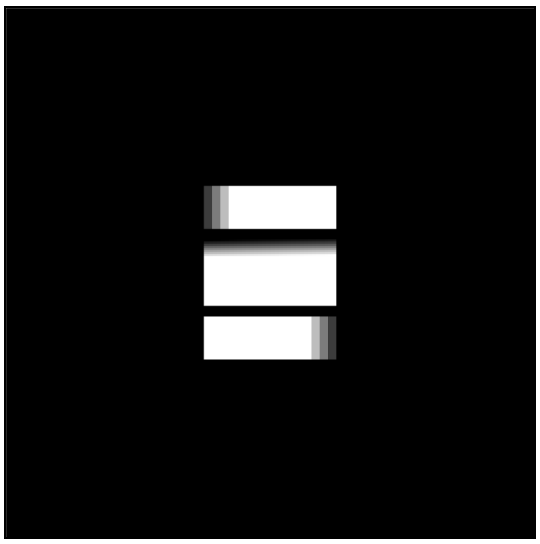
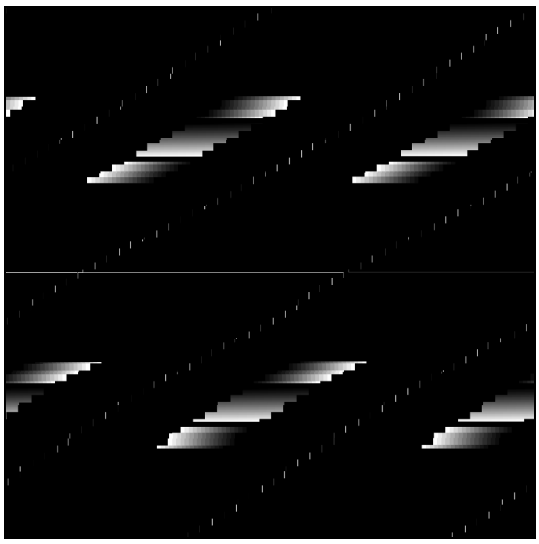
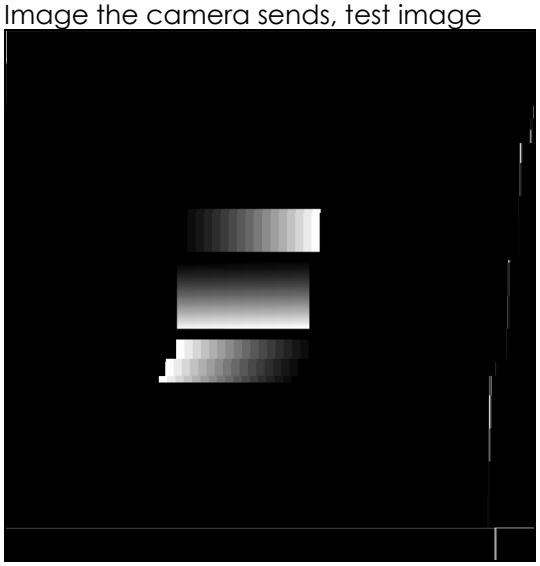


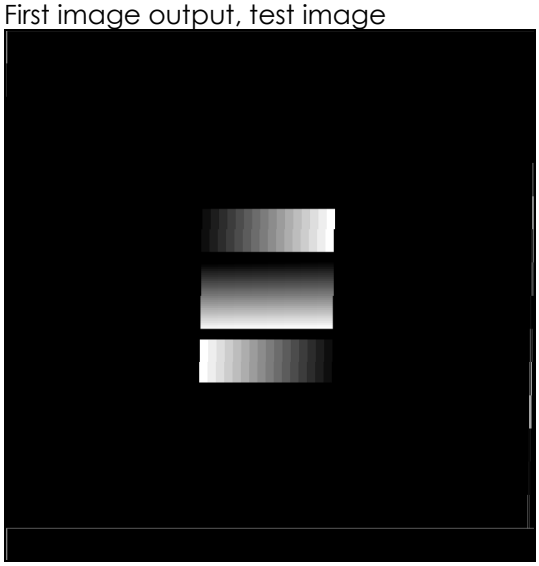
Image the camera sends, test image



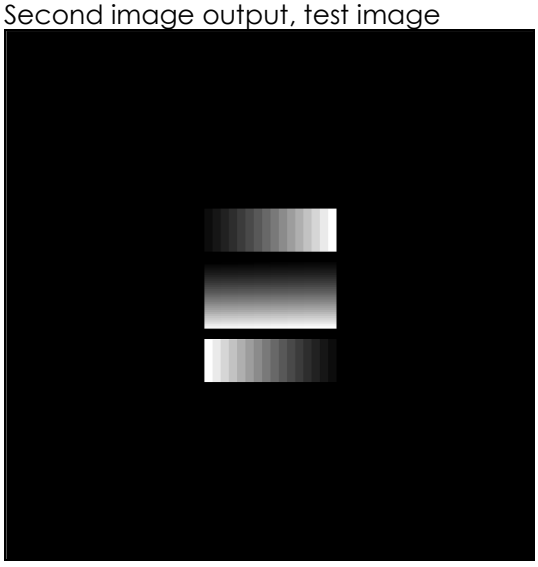
First image output, test image



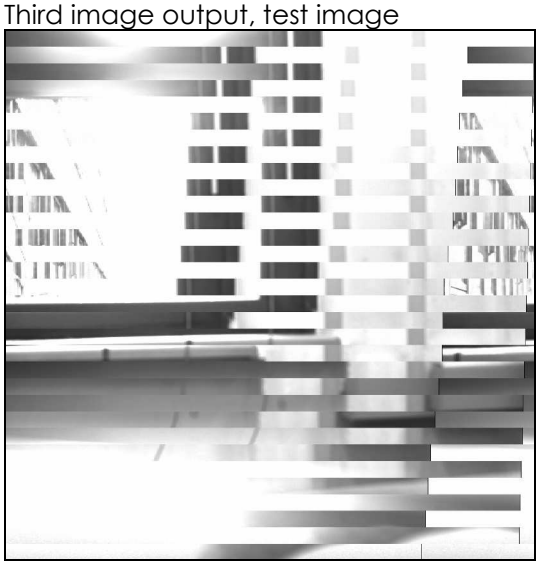
Second image output, test image



Third image output, test image



Fourth image output, test image



Fifth image output, real image


Sixth image output, real image


Seventh image output, real image


Eight image output, real image