NTNU

Norwegian University of
Science and Technology

Embedded Demonstrator for Video Presentation and Manipulation

Cato Marwell Jonassen

Master of Science in Electronics
Submission date: June 2010
Supervisor: Per Gunnar Kjeldsberg, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

In many situations, like school visits at NTNU,
Forskningstorget and Elektronikk-& telekommunikasjonsdagen,
it is desirable for Department of Electronics and Telecommunication to demonstrate
good examples of electronic systems.
Embedded systems are well suited as demonstrators since the combination of hardware
and software gives both flexibility and wide possibilities for optimization.

In this task, the student will implement a specified system for demonstration of
topics related to courses provided by the department.
In order to make a good demonstration, necessary presentation material is also to be made,
together with a plan on how to demonstrate the embedded system.

Assignment given: 15. January 2010
Supervisor: Per Gunnar Kjeldsberg, IET

Abstract

In this master thesis there has been implemented an embedded demonstrator for video presentation and manipulation, based on the specification presented in the project thesis written last semester. The demonstrator was created with the intention of being used by Department of Electronics and Telecommunication in situations where the department needed good examples of electronic systems. These systems can be used to motivate, educate and possibly recruit new students.

By combining the use of video as a motivational medium with a practical approach to the theory, the demonstrator is designed to emphasize the importance of hardware/software codesign in electronic systems. The demonstrator is designed with a combination of dedicated hardware modules and the Nios II/f embedded soft processor from Altera. Video is processed in both hardware and software to demonstrate the difference in obtainable video quality. A measured frame rate of 25 fps in hardware and less than 1 fps in software is considered to be a good demonstration of the difference in processing power. An additional color processing demonstration is also created to visually demonstrate the performance differences when processing colors using software versus using custom floating-point instructions. It is concluded that an average performance increase of 300% is archived when using custom instructions, which is concidered to be noticeable visually. A poster with the necessary theory, usage guidelines and results has been created to support the demonstration together with a plan of how the demonstration should be performed based on the age and educational background of the observer.

The embedded demonstrator was implemented using the Altera DE2 platform in combination with the TRDB D5M camera and hardware description from Terasic.

Preface

One of the most important reasons for choosing this task for my master thesis was that it gave me the possibility to use the theory and knowledge I have gained over the years here at NTNU to make something useful and practical. I also believed that the design task would be a good preparation for the challenges I will meet when I start working. I have always been fascinated by electronics and the wide area of application this technology presents. Since my interests include both computers and different forms of visual media, this project became a great opportunity to combine my interest and education. This combination is in my opinion the most effective and motivating way to learn new things.

The task of implementing and creating an embedded demonstrator system has been both challenging and rewarding. It is always frustrating when not even the simplest things work as they are supposed to, but it is an even bigger reward when one finally solves the problem. One of the biggest challenges this past semester has been time. It is not easy to predict how much time it will take to get things to work the way they are are supposed to. This has been a very intense semester, especially the last couple of months. But now, when I am finished, I look back and see it all as a very exciting time from which I have learned a lot, and even got to create a new system. I have learned a lot about embedded systems, hardware/software codesign, cameras and video and much more. The Altera DE2 platform was a very good platform to work with and it gave room much creativity. I have created the embedded demonstrator by using Terasic's existing camera hardware platform, and I have also used much of the theory from the project in this master thesis.

This project would not have been possible without all the help and support I have received. I would like to thank my supervisor, Professor Per Gunnar Kjeldsberg for all the support, help and fresh perspectives I have gotten. I would also like to thank my two best friends Jarle Larsen and Kai Andre Venjum, for sticking with me, through thick and thin. Thanks for all the constructive discussions, shared knowledge and all the fun we have had together when creating our systems. Finally, I would also like to give my thanks to my better half, Tia. Without your great support, optimism and interest, this would not have been possible.

Cato Marwell Jonassen
NTNU, Trondheim
June 2010

Contents

List of Figures

List of Tables

Definitions

ASIC : Application-Specific Integrated Circuit - a hardware circuit of the application in silicone

BSP : Board Support Packages - A NiosII BSP project is a specialized library containing system-specific support code

CCD : Charge-Coupled Device - a device for movement of electric charge. When combined with a image sensor and a Bayer filter it serves a great purpose for digital imaging

CFA : Color Filter Array - a mosaic of tiny color filters placed over the pixel sensors of an image sensor to capture color information

FIFO : First-In First-Out - refers to a way of queuing and organizing data by giving highest priority to the data arriving first (First-Come First-Served - principle). A FIFO buffer is a buffer organized in this way

FPGA : Field-Programmable Gate Array - a programmable device for realizing hardware

FPS : Frames Per Second - the rate or frequency of which unique pictures or frames is produced

HAL : Hardware Abstraction Layer - an abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on it

HDD : Hard Disk Drive - a none volatile storage device for digital data

HDL : Hardware Descriptive Language - a language for describing hardware

IP : Intellectual Property - a term referring to a number of distinct types of creations of the mind for which property rights are recognized and owned

IRE : Institute of Radio Engineers - is a unit used in the measurement for composite video signals. It is a relative measurement (procent). 100 IRE was originally designed to be the range from black to white video signal

LE : Logic Element - a basic element used in FPGAs. The number of LEs is often used as one of the paramters for an indicator of total area

MMU : Memory Management Unit - a unit that is responsible for handeling all communication between CPU and memory

MPU : Memory Protection Unit - a unit that protects the memory. Sectors that are critical would be secured against unauthorized alterations

PLL : Phase-Locked Loop - a unit that is used to create a new signal with equal or different phase and frequency by using a reference signal

RISC : Reduced Instruction-Set Computing - a processor architecture which uses a reduced number of different instructions

SDRAM : Synchronous Dynamic Random Access Memory - a type of volatile memory which needs to be periodically refreshed

SRAM : Static Random Access Memory - a type of volatile memory that does not need to be refreshed, but still needs power to hold data

VGA : Video Graphics Array - can refere to both the analog computer display standard for the d-sub connector or the 640x480 resolution

Chapter 1

Introduction

Today, in our modern society, people are surrounded by many different forms of multimedia every day. Video, images and sound have become an important part of how people communicate and gain knowledge, as well as an important medium for entertainment and a part of culture. Thanks to great improvements in the field of technology and electronics many things that were deemed impossible has become possible. People can talk and see each other in real-time across continents with use of video conference tools in a quality that give the sense of talking to a person right next to you. The mobile phones today are filled with sophisticated electronics that let people watch TV, listen to music streamed from the Internet, take photos and share them with friends, surf the Web, talk to people with or without video and one can even watch movies. The possibilities are endless, but in order to make enhancements in the field of technology and electronics there is a need for engineers that possess the knowledge and interest to make those things come true. Universities, like NTNU, are always searching for new students and to recruit them. In order to promote the field of electronics, Department of Electronics and Telecommunication desires to demonstrate good examples of electronic systems that can be used on occasions like school visits at NTNU, Forskningstorget and Elektronikk- & Telekommunikasjonsdagen. Hopefully, this could result in an increased number of applicants for this field of study and as a consequence get clever students who have the ability to take part in making the next generation of electronic systems.

Most part of the electronic systems people surround themselves with have some form of computational properties. These systems are ”small” computers, dedicated to perform the task they where designed to do. Such systems are often referred to as embedded systems. Embedded systems consist of a combination of hardware and software which gives both great flexibility and wide possibilities for optimization. Systems with such properties are great platforms for making demonstrators and let the designer be creative.

Most forms of multimedia processing are very complex and puts a high demand on available computational power. Many devices that do this form of computation is running on battery, and battery lifetime is important. Not being able to watch a whole movie on a de-

vice before it runs out of battery would be unacceptable to most people. As performance costs power, the designer needs to be smart when creating the device. Hardware/Software codesign is about meeting system objectives by exploiting the synergism between hardware and software trough their concurrent design [48]. Performing the calculations where it's most efficient, both in relation to performance and power, would be a HW/SW codesign choice. Such choices are an important part of the designers job and could mean the difference between meeting system objectives or not.

FPGA or Field-Programmable Gate Array is a programmable device that allows the designer to create almost any form of hardware from a description made in a hardware language. It is also possible to create software processors on the device and run them side by side with custom designed hardware. In other words, it is a great platform for hardware/software codesign.

1.1 Preliminary Work

Prior to this thesis there has been done some research. This research was a study on different educational challenges related to demonstration of electronic systems. In addition there was considered many different types of applications in relation to demonstration before one final system was specified.

From research and discussions in the project report it was concluded that the application of the demonstration should focus on a specific topic in order to keep the focus on what the demonstrator should demonstrate. In addition there was also concluded that the demonstrator should demonstrate something the observer could relate to in order to motivate him or her. It was also considered that a visual demonstration on a technical topic would help bring interest and also be effective when trying to get the observer to learn something from the demonstration. Also, by letting the observer take an active part in the demonstration it would support the observer in coming to his or her own conclusion about the demonstration. This is considered as a good pedagogical approach.

Many different systems were considered, but two prevailed themselves as the most promising candidates. One of the ideas was an embedded demonstrator for visual demonstration of audio properties. The other idea was an embedded demonstrator for video presentation and manipulation. Both candidates had the ability to become good demonstrators, but idea number one presented some additional challenges. The challenges were related to creating a visual interface in the available timeframe. Idea number two was selected and a specification of a possible implementation was suggested.

1.2 Description

In this master thesis an embedded system for presenting and manipulating video will be described and implemented. In addition, the necessary presentation material will be made. The system's purpose is to emphasize the importance of hardware/software codesign when designing an electronic system. Without the unique combination of hardware and software it might not be possible to achieve system objectives and specification. The demonstrator is going to demonstrate this importance by processing video from a camera through hardware or software. The observer is then able to select between the two and evaluate the difference in obtainable video quality. In addition the observer will be able to select between normal color representation and inverted color representation. The system is also going to visually present the time difference by doing color manipulation in software or by using dedicated hardware by means of custom instructions.

This system presents many design challenges. One of the main challenges this project brings to surface is the available performance of the embedded software processor and the uncertainty of its ability to present some form of viewable video. The final solution offers a demonstration of video processing and manipulation. The ability to observe camera-video, processed in either software or hardware in real-time is considered to be a good concept for a demonstration. In addition, the platform shows great promise for further development and can easily be adapted to be used in other educational situations.

This report is divided into ten main chapters. The first seven chapters (Chapter 2 - Chapter 8) present theory about different topics related to the demonstrator. Chapter 9 presents the implementation of the complete system and a discussion of the challenges, different solutions and other topics related to the design process. Chapter 10 presents a walk through of a hypothetical demonstration of the system. Presented in the final chapter (Chapter 11) is a conclusion and suggestions for future work.

1.3 Main Contributions

The main contributions of this master thesis will be:

- A fully functional embedded demonstrator that processes recorded video in both hardware and software
- A hardware video processing performance of up to 25 frames per second
- A system that demonstrates the benefits of using custom floating-point instructions to accelerate color processing, with up to 300% increase in performance compared to software processing
- A poster with necessary material to give a full demonstration of the embedded demonstrator

Chapter 2

Embedded Processors

First, a little introduction as to what an embedded system actually is. Most people are familiar with the desktop computer. The desktop computer is a very versatile computing machine which can be used for numerous applications, like playing games, checking mail and much more. An embedded system can also be defined as a computer system, but which is designed to perform a specific task [30]. This as opposed to the desktop computer which is designed with the purpose to be much more generic. As a result the embedded system can be constructed to perform the individual tasks in a much more effective manner, both in relation to size, power consummation and performance. Embedded systems are actually much more common than the desktop computer, although they are not always as easy to spot. About 99% of all the microprocessors being made is used in embedded systems [29]. You can find embedded systems in almost any kind of electronic consumer products, like cars, washing machines, coffee makers, toys, TVs, Blu-Ray players and the like. Embedded systems are also very common and important in industry and are often used to control mechanical systems many of which have hard demands in relation to responsiveness and speed. These systems are often referred to as a real-time system [29]. Many of these systems have hard demands which would not have been possible to achieve with a conventional desktop computer. Possible reasons could be insufficient processing power or to slow response time. Embedded systems usually consist of a combination of hardware (e.g. I/O, memory, peripherals, accelerators, co-processors) and software (some kind of program to tell the system what to do). This combination gives flexibility and wide possibilities for optimization and customization. For an embedded system to be able to run software a microprocessor for executing instructions is needed.

2.1 About the Embedded Processor

The main function of a processor is to manipulate data in a way specified by a sequence of instructions [30]. The series of instructions is what constitutes a software program, written in some kind of language (e.g. C and C++). The software is stored in a memory (e.g. Flash, SRAM, SDRAM, HDD) and the processor fetches the instructions sequentially by use of extra logic for translating the instructions into control signals and a program counter to point to the next instruction. A microcontroller is a processor which incorporates all necessary hardware in order to function as a computing unit. Consisting of a CPU (Central Processing Unit) and different kinds of peripherals including memory. The peripherals could be user-designed or generic (standardized). Some examples of generic peripherals are: SPI (Serial Peripheral Interface Bus), UART (Universal Asynchronous receiver/transmitter), JTAG(Joint Test Action Group - for debugging and test), OnChipMemory (Flash, SRAM or something similar) and I^2C (Inter-Integrated Circuit - serial bus system). The peripherals are used to give the processor purpose. A processor with no means of communication to the analog world would not be very useful.

There are two main types of embedded processors: Soft processors and hard processors, as stated in [34]. The soft processor is typically made out of logical elements found in FPGAs and described in a Hardware Descriptive Language (HDL), like VHDL or Verilog. The hard processor is, on the other hand, not possible to alter once it has been created in silicone. This is because the processor does not consist of reconfigurable elements like the soft processor does. Hard processors can appear in both ASIC (Application-Specific Integrated Circuit) and as a part of a FPGA. One example of a FPGA which includes a hard processor is the Xilinx FPGA Vertex-V FXT family [4]. It includes the PowerPC 440 embedded processor architecture from IBM.

There are both advantages and disadvantages in using a soft or a hard embedded processor [51, 63, 34]. In [34] there is described four advantages by using the soft processor: Customization, increased lifespan, component and cost reduction and hardware acceleration. The hard processor's main advantage is its high performance and low unit cost.

The soft processor is very customizable and can easily be altered to fit the current need. This gives the designer the possibility to be flexible when selecting a combination of peripherals and controllers. The designer can even invent unique peripherals that can be connected directly to the processor's bus, as opposed to the hard processor where the designer must choose between the off-the-shelf embedded processors that are currently available, many of which are too large or too small for the task the designer wants to solve. If the hard processor is combined with a FPGA, like the Xilinx Vertex-V FXT, one can benefit from both and gain some customization, but the processor core itself can not be altered.

Peripherals can be constructed by the designer or bought from a vendor. The available cores are often distributed with an IP (Intellectual Property) which means that someone owns the source code for the block [9]. The use of IP cores can greatly reduce the overall

design time and reduce the time-to-market. The FPGA vendor Altera offers a wide range of IP cores from Altera and other third-parties [9]. For implementing these cores Altera users can use Quartus and SOPC builder, which is described in more detail in Section 5.1. Hardware cores are also available as a free and open source solution under the free Lesser General Public License (LGPL) at `www.OpenCores.org` [50].

Since the hardware is described in a HDL like language, the lifespan of the product is much longer. It can be re-implemented on future FPGA platforms and can easily be altered to become an even better product. Another advantage in using a soft processor is component and cost reduction. The versatility of the FPGA enables the designer to use only one FPGA to replace a system that required multiple components. By reducing the number of components, the company can also reduce board size and inventory management, both of which will save design time and money.

Hardware acceleration is one of the most compelling reasons to choose an FPGA embedded processor [61]. This gives the designer the unique ability to combine and make trade-offs between hardware and software to maximize performance [48]. More theory about this subject is presented in Section 3.

There are a lot of advantages in choosing a soft processor architecture, but there are still cases when a hard processor might be a better choice. While the hard processor can be bought in a completely ready-to-use state, the soft processor has to be constructed and the designer has to be able to both construct hardware as well as software. Luckily, tools like SOPC builder (Section 5.1) makes this job much easier and it requires less knowledge about hardware contruction. Another disadvantage is that the soft processor needs more complex tools and the design methodology needs more attention. Another aspect to consider is device cost. If the designer is about to solve a ”simple” problem that does not require much processing power or unique features, an off-the-shelf processor could be a much cheaper and better choice. If there already is an FPGA present in the system, the designer might want to consider moving much of the external hardware into the reconfigurable chip and also include a soft processor.

2.2 Nios II

Nios II, as stated in [19], is a 32-bit soft embedded processor architecture designed for the Altera family of FPGAs. The older Nios was introduced in 2001 [10] and was the industry's first viable commercial processor created specifically for embedded system design in FPGAs. Nios II displayed several enhancements over the older Nios, thus making it more suitable for a wider range of embedded computing applications [19]. Since Nios first was introduced, tens of thousands of FPGA users have adopted Nios or Nios II processors from Altera. Xilinx's MicroBlaze is an example of a very similar product [3]. This is also a 32-bit soft embedded processor. The processor is designed to be used on Xilinx's family of FPGAs.

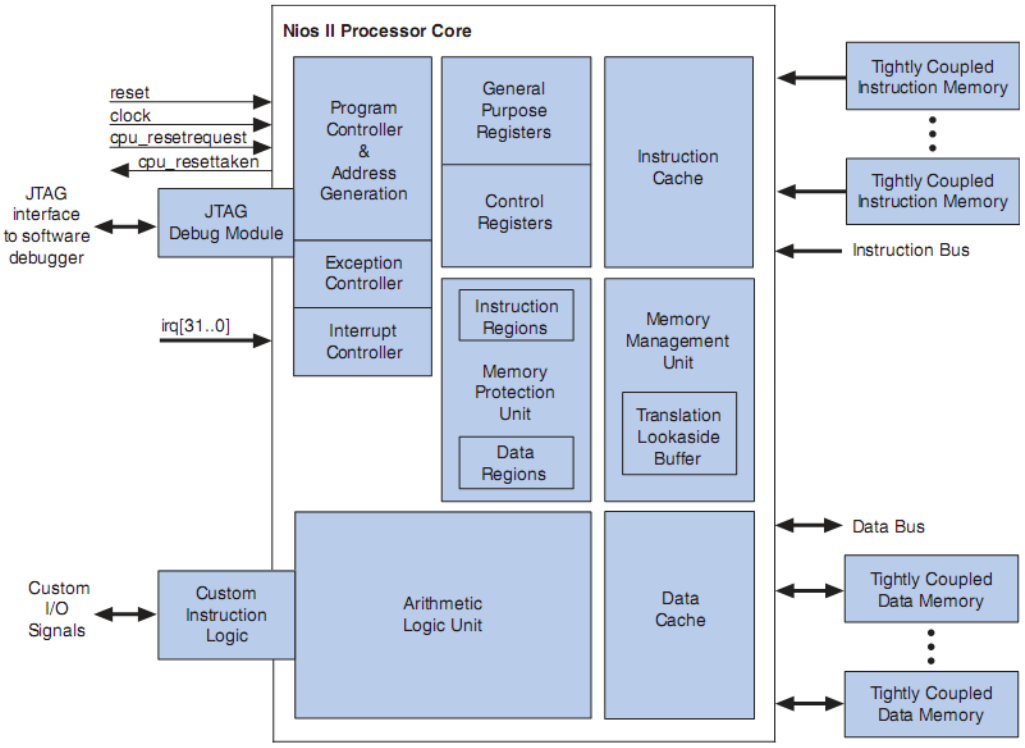

Figure 2.1: NIOS II processor core - from [19]

The processor core of the Nios II is displayed in Figure 2.1. It provides a full 32-bit instruction set, data path and address space. This makes it possible to access up to 4 GB of external memory. In addition the core has 32 general-purpose registers which can be used to store and alter time critical data. The core also supports 32 external interrupt sources which can be used by external components to get the processor's attention. More information about interrupt can be found in Section 2.3. The core can also execute complex instructions like 32x32 multiplication or division, and instruction barrel shifter. For

debugging the core supports JTAG, which is a standard for hardware-assisted debugging. This enables possibilities like start, stop, step and trace execution in run-time. The software development environment is based on GNU C/C++ tool chain and Eclipse IDE. Code creation, simulation, execution and debugging can be done in this environment. More information about this embedded design suite can be found in Section 5.3.

Processor core implementations

Feature		Core		
		Nios II/e	**Nios II/s**	**Nios II/f**
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz *(1)*	0.15	0.74	1.16
	Max. DMIPS *(2)*	31	127	218
	Max. f_{MAX} *(2)*	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	Without MMU or MPU: < 1800 LEs; < 900 ALMs With MMU: < 3000 LEs; < 1500 ALMs With MPU: < 2400 LEs; < 1200 ALMs
Pipeline		1 stage	5 stages	6 stages
External Address Space		2 GBytes	2 GBytes	2 GBytes without MMU 4 GBytes with MMU

Table 2.1: NIOS II core implementations comparison part 1 - from [19]

The Nios II core is available as three different types. All cores support the Nios II instruction set architecture and the implementation differs in some of the feature they support and their main objective. The implementations are Nios II/e, Nios II/s and Nios II/f. Their objectives are minimal core size, small core size and fast execution speed. Table 2.1 gives an overview of some of the differences between the three. Some of these differences are related to: Maximum obtainable clock frequency, maximum DMIPS/MHz (performance, se more in Section 2.4) and the amount of area the core consumes. The Nios II/s uses approximately 1400 LEs (Logic Elements) as opposed to the Nios II/f which uses around 1800 LEs (without MMU or MPU). The difference in area consumation is about 20%, but the increase in performance if chosing Nios II/f would be about 40%. Nios II/e uses about half the size of the Nios II/s, but is still able to support the whole Nios II instruction set. Depending on the designer's need it should be a simple choice to select the right core for the job. All values presented in Table 2.1 are based on choosing the fastest options and using Altera's fastest FPGAs. If the designer uses an FPGA with a lower speedgrade, this will result in some speed reduction.

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Instruction Bus	Cache	–	512 bytes to 64 KBytes	512 bytes to 64 KBytes
	Pipelined Memory Access	–	Yes	Yes
	Branch Prediction	–	Static	Dynamic
	Tightly-Coupled Memory	–	Optional	Optional
Data Bus	Cache	–	–	512 bytes to 64 KBytes
	Pipelined Memory Access	–	–	–
	Cache Bypass Methods	–	–	■ I/O instructions ■ Bit-31 cache bypass ■ Optional MMU
	Tightly-Coupled Memory	–	–	Optional
Arithmetic Logic Unit	Hardware Multiply	–	3-cycle *(3)*	1-cycle *(3)*
	Hardware Divide	–	Optional	Optional
	Shifter	1 cycle-per-bit	3-cycle shift *(3)*	1-cycle barrel shifter *(3)*
JTAG Debug Module	JTAG interface, run control, software breakpoints	Optional	Optional	Optional
	Hardware Breakpoints	–	Optional	Optional
	Off-Chip Trace Buffer	–	Optional	Optional
Memory Management Unit		–	–	Optional
Memory Protection Unit		–	–	Optional
Exception Handling	Exception Types	Software trap, unimplemented instruction, illegal instruction, hardware interrupt	Software trap, unimplemented instruction, illegal instruction, hardware interrupt	Software trap, unimplemented instruction, illegal instruction, supervisor-only instruction, supervisor-only instruction address, supervisor-only data address, misaligned destination address, misaligned data address, division error, fast TLB miss, double TLB miss, TLB permission violation, MPU region violation, hardware interrupt
	Integrated Interrupt Controller	Yes	Yes	Yes
User Mode Support		No; Permanently in supervisor mode	No; Permanently in supervisor mode	Yes; When MMU or MPU present

Table 2.2: NIOS II core implementations comparison part 2 - from [19]

Nios II/f

As mentioned before, the Nios II/f focuses on fast execution speed. Compared to the Nios II/e which uses a minimum of 6 cycles per instruction, the Nios II/f uses an average of 1 cycle per ALU instruction [19]. The performance of the Nios II/f is achieved by maximizing the max frequency of the processor core and a high instructions-per-cycle execution efficiency. This core employs a 6-stage pipeline to achieve a high DMIPS/MHz as seen in Table 2.1. A pipeline is often used to increase data-throughput in a computing system [40]. This is done by inserting a chain of data-processing stages. Nios II/f also has separate cache (fast memory) for data and instructions. The size of these two can be adjusted to fit the designer's need. The data and instruction bus can be seen in Figure 2.1. A possible memory for this job could be the fast on-chip memory on the FPGA, described in more detail in Section 6.2.5. In addition the Nios II/f provides an optional MMU and MPU to support operating systems that require this. If an MMU is selected, the processor can access up to 4GB of external memory. Another important property of the Nios II/f is the support for custom instructions, described in more detail in Section 3.2.1. Custom instructions can seriously increase the overall performance. Table 2.2 shows more of the options and properties of the available core implementations of Nios II.

2.3 Interrupt

Interrupts are a technique of diverting the processor from the execution of a current program so that it may deal with some event that has occurred [30]. The events could be many things. It could be an error from a device or simply an I/O device that has finished a result. Interrupts can be divided into two groups: Hardware interrupts and software interrupts. Hardware interrupts are interrupts from hardware like an I/O device. There are two ways of generating this interrupt. One is by using the processor itself to do the checking of the I/O device, to check if there are any new data. This is called polling. Another way to do it is to let the device itself generate an interrupt signal which "pulls" the processor away from its current job and into an Interrupt Service Routine (ISR). When such an interrupt occurs, the processor saves its state and loads the interrupt vector for the specific interrupt. The interrupt vector is an address to where the ISR can be found. The processor then executes the routine with its series of instructions before it resumes its previous state. Solution number one would demand very much processor time just for checking and waiting on the I/O device. This isn't very efficient use of processing power. The other solution would therefor be a much more efficient solution. Software interrupt is the other type of interrupt and is usually generated by an instruction, or it could be an exception occurring in a program.

When operating with interrupts there is a chance that multiple interrupts can occur, maybe two at the same time or one being followed by an other. In such cases the system

must be able to handle this. One way of handling it is to give the different interrupts priorities. This is how it is solved by the Nios II embedded processor. It supports up to 32 separate interrupts and each interrupt is given a specific number, between 0 - 31 [19]. The lower the number the higher the priority. In this way, if two interrupts with different priority occurs the processor will first finish the one with the highest priority and then do the second. It is also possible to deactivate interrupts on the processor. This could be done inside an important ISR. This would result in a rejection of all interrupts while it is turned off. These are considerations which must be done by the designer of the system.

Nios II and Interrupt

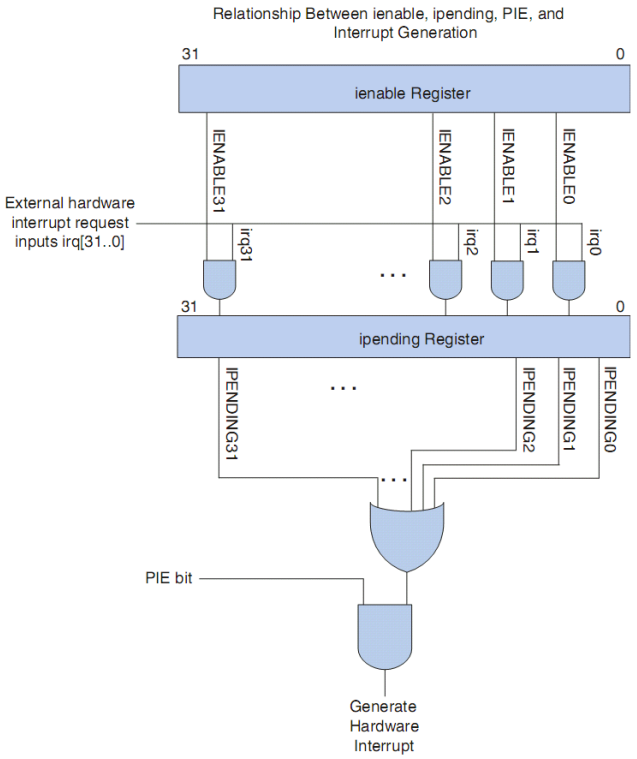

Figure 2.2: Relationship between `ienable`, `ipending`, `PIE` and Hardware Interrupts - from [19]

There are three conditions that have to be true in order for a hardware interrupt to be generated on the Nios II [19].

- The `PIE` bit of the `status` control register is one
- An interrupt-request input, `irq`n, is asserted
- The corresponding bit n of the `ienable` control register is one

Figure 2.2 shows how this relationship is constructed. A software exception routine determines which of the pending interrupts that has the highest priority and then transfers the control to the appropriate ISR. The ISR disables the interrupt before it is re-enabled after finishing the ISR. The interrupt can be re-enabled in the ISR by writing one to the `PIE` bit. This allows the current ISR to be interrupted and the designer must take this into account when designing the system.

Interrupt Vector Custom Instruction

The Nios II processor offers an interrupt vector custom instruction which reduces average and worst case interrupt latency. The interrupt vector custom instruction improves both average and worst case interrupt latency by up to 20% [19].

2.4 Performance

There has been suggested many different metrics and benchmarks for measuring performance over the years [42]. Some of them give better measurement of "true" performance than others. In many advertisements for computer systems, the most prominent factor is often the frequency of the processor core. This could lead to the assumption that a processor running at 250MHz has a greater performance than a system running at 200MHz. This is not always the case. Performance depends on many factors. The clock rate says something about how fast the processor runs, but it does not say anything about how much computation that is done per clock cycle. Even though clock rate is not a very good metric for performance, an increase in clock rate on a specific processor would result in an increase in performance on the same processor.

MIPS is one simple form of metric of performance. It is a measure of throughput or execution-rate and it defines the computer system's unit of "distance" as the execution of an instruction. MIPS, or Millions of Instructions executed Per Second, is defined in Equation 2.1.

$$MIPS = \frac{n}{t_e \times 10^6} \tag{2.1}$$

t_e is the time required to execute n total instructions. One of the problems with MIPS as a performance metric is that the amount of computation per instruction is not equal on all processors. Some of the instructions are far more complex than other.

$$MFLOPS = \frac{f}{t_e \times 10^6} \tag{2.2}$$

MFLOPS is a performance metric that tries to correct some of the shortcomings of MIPS. MFLOPS, or Millions of Floating-point Operations executed Per Second, defines

an arithmetic operation on to floating-point quantities to be the basic unit of ”distance”, see Equation 2.2. The main problem with this metric is when used on a system that executes a program with no floating-point calculations. This would give the result zero.

Very often with embedded computers the system is going to be used to execute a specific program. In this case the ultimate performance metric would be to measure the time the processor uses to execute this program. The result, however, would just be accurate for that specific program, but it could be used to compare execution time on other processors executing the same program. To be able to measure time one would need very precise type of measurement, like the number of clock cycles.

Nios II can use the Performance Counter Core [21] to measure performance. The core can be included with SOPC builder, discussed in Section 5.1.1. Performance Counter Core returns the number of clock cycles the specified piece of code needed to execute.

Dhrystone

DMIPS (Version 1.1) also called Dhrystone MIPS (Million Instructions Per Second) is a synthetic computing benchmark program developed in 1984 by Reinhold P. Weicker [58]. Dhrystone is a benchmark completely based on integer operations and does not support floating point operations. At this time the integer performance predominated, with little or no floating-point calculations. The Dhrystone model was viewed as a ”typical” application mix of mathematical and other operations. In 1988 Dr. Weicker created Dhrystone Version 2.1 which remains in the original format today. This version was improved over the earlier version to give a more accurate measurement.

Some of the strengths with Dhrystone are: Easy to report score - single figure of merit, easy to implement and run on most platforms and architectures, and since it’s ”integer-only-code” it is potentially very useful for simple 8- and 16-bit embedded processors. Dhrystone does have a lot of weaknesses too. Some of these are: Cannon hope to mimic the breadth of applications encountered on a processor based system, it is based on only one set of functions, does not measure multiply-accumulate, floating-point,SIMD or any other type of operations and there is a possibility that the designer can cheat to get good benchmarks. Dhrystone represents a more meaningful performance representation than MIPS, but it does not give an accurate and true performance measurement. It is more usable as an indication of performance.

In Table 2.3 there are listed some DMIPS/MHz results. It shows a comparison between the soft embedded processor Nios II, the hard embedded processor PowerPC 440 from IBM and a common dedicated hard CPU (Intel Core 2 Duo) from Intel, located in many of today’s desktop computers. The IBM PowerPC 440 embedded core is a 32-bit RISC CPU providing a performance of about 2.0 DMIPS/MHz [2, 35] and about 1000 DMIPS at the max clock rate of 555MHz on nominal silicone. In comparison with the performance of a PC based CPU it is listed that an Intel Core 2 Duo at 2,4 GHz with 32 bit integers can perform at 8094 DMIPS, which gives a 2.28 DMIPS/MHz. The Nios II/f

Name	Type	DMIPS/MHz	f_{max}
Nios II /e	Soft	0.15	200MHz
Nios II /s	Soft	0.74	165MHZ
Nios II /f	Soft	1.16	185MHz
PowerPC 440	Hard	2.0	555MHz
Intel Core 2 Duo 32bit	Hard	2.28	2.4GHz

Table 2.3: Embedded Processor Performances Using Dhrystone Version 2.1 - from [43, 18, 19, 2, 35]

is said to be able to achieve up to 250 DMIPS with use of the fastest FPGA and highest possible clock rate. The performance gap between the hard and soft embedded processor is quite large, but as research on FPGAs and on different types of optimizing is made, this gap is becoming smaller and smaller. The research made in [45] suggest a speedup using a partitioning algorithm to move "heavy" computations from software to hardware and combine them by using custom instruction. They demonstrate that the soft-core based processor achieves average speedups by a factor of 5.8 and energy reductions of 57% compared to the soft core alone. More on custom instructions can be seen in Section 3.2.1.

Chapter 3

Hardware/Software Codesign

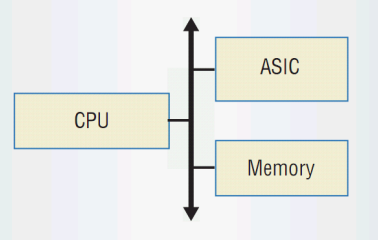

Figure 3.1: Target architecture for hardware/software partitioning - from [60]

Hardware/Software codesign is a term that first emerged in the 1990's to describe a confluence of problems in IC design [60]. At the time the processor-based, or software executable, systems were mostly on board-level, but it was foreseen that in the near future technology would give room to combine a microprocessor together with an ASIC, like Figure 3.1 shows. Now, 20 years later, hardware/software codesign has gone from being an idea to becoming a very important mainstream science design methodology. Giovanni and Rajesh [48] say that: "hardware software co-design means meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design". In other words, HW/SW codesign is a design methodology used to meet the increasing demands to performance, cost and design-time with use of a combination of hardware and software. HW/SW codesign focuses on utilizing more of the available transistors by benefiting from reuse of HW and SW macro blocks [48]. This results in better cost/quality, flexibility, better performance and shorter development time. Creating an embedded computer that meet such goals is typically a codesign problem [59]. The design of HW and SW components influence each other and the codesign methodology requires intimate knowledge of the interactions between the hardware and software components. At present, there is a much deeper understanding of the HW and SW disciplines separately than about codesign. As ICs become more and more complex, the challenges with codesign will also increase, making HW/SW codesign a vibrant field for a long time [60].

3.1 Embedded Systems and HW/SW Codesign

The goal when making an embedded system is to find the right combination of HW/SW resulting in the most efficient product within the specifications [46]. Basing the embedded system on an existing platform, like the Nios II core from Altera [19], will greatly reduce the time-to-market when designing complex systems. Following is a list of design activities that should be used when designing an embedded system by use of the HW/SW codesign methodology.

Design activities:

1. Task level concurrency management
2. High level transformations
3. Hardware/software partitioning
4. Compilation
5. Scheduling
6. Design space exploration

1.: At the task level the designer should regroup the task from the specification to maximize the implementation efficiency. This is often done by either merging or splitting tasks. When merging the overhead can be reduced by combining tasks for fewer context switches. While splitting task has the purpose of maximize the use of resources.

2.: The goal with the high level transformations is to improve efficiency of embedded software. One example of this is to use fixed point instead of floating point, which gives a dramatic increase in performance. This could reduce the accuracy, but in most cases it is not noticeable. Other possibilities is to do transformations of loops generated by software so that the access to memory is adjacent. If not, this would result in very few data being in the cache and give bad performance. Loop-tiling-and-blocking exploits memory hierarchy efficiently, and focuses on keeping the relevant data in cache to increase performance.

3.: In the HW/SW partitioning stage the designer should try out different partitioning of HW/SW. To find the best partitioning of which task to solve in HW and which to solve in SW to get the best result.

4.: Using an optimized compiler may reduce energy consumption, increase speed and reduce software size. The compiler exploits knowledge about the underlying processor and there are often special hardware-aware compilers

5.: Scheduling is about mapping of operations to start time.

6.: In most cases there are several designs that meet the specifications. Design space exploration is the process of analyzing the set of possible designs, among those that meet the specifications, and choose one of them.

3.2 Hardware Acceleration and Custom Instructions

In [16, 48, 28, 19] hardware acceleration and instruction set extension are presented as two ways to accelerate processes or tasks in hardware.

3.2.1 Custom Instructions

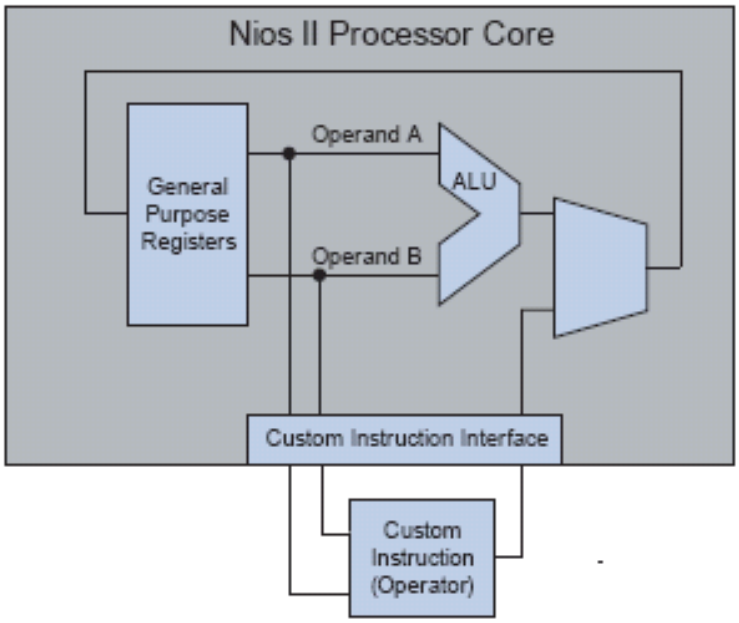

Figure 3.2: Custom Instructions with Nios II soft processor - from [19]

Instruction set extension is a inclusion of additional functional units to speed up common operations [28]. The unit is connected to the CPUs register file. One of the challenges is to decide which operations goes into hardware and which is performed in software. One way of choosing is to test all possible combinations of HW and SW partitions, which is a lot of work. ISEGEN is a suggested deterministic algorithm that tries to mimic a human designer to come up with the best partitioning solution. When partitioning, a common rule is to perform tasks that are very repeatable in HW.

All the Altera Nios II processors support instruction set extension [19]. Similar to native Nios II instructions, custom instruction logic can take values from a source register and optionally write back the result to a destination register. The custom instruction logic block can be seen in Figure 2.1 and Figure 3.2. If the designer demands more flexibility of the dedicated hardware, a separate hardware accelerator could be a better choice.

3.2.2 Hardware Acceleration

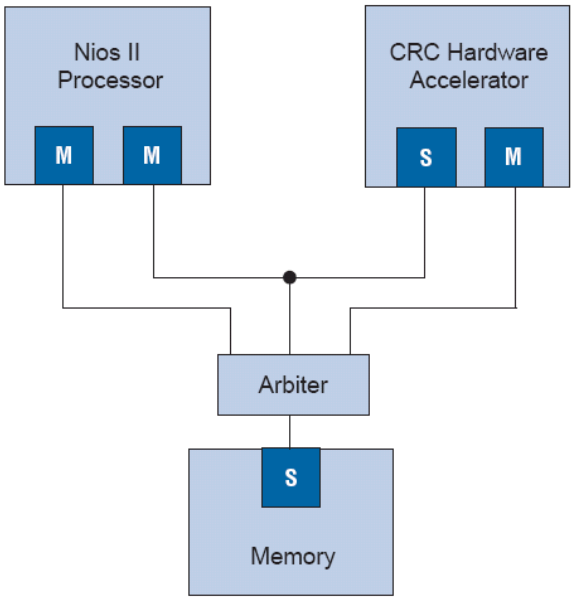

Figure 3.3: Example of a CRC Hardware Accelerator with Nios II - from [19]

A hardware accelerator is a piece of hardware designed to accelerate the execution of a specific task [61]. It does not execute instructions and is more functionally equivalent to an I/O-device. The goal of the hardware accelerator is to increase the performance/cost by performing specific functions that are more efficient here. The accelerator could be implemented as an ASIC, by use of a standard component, an FPGA or designed hardware with use of PEs on an FPGA. A typical target for acceleration would be an application with repeated computations and low latency I/O functions, operations not fit to CPUs, highly responsive I/O or streaming data, like video. In order for an accelerator to be beneficial, the payoff should be considerably more than the cost of implementing it. Therefore the designer should perform an analysis of the cost of performing the task in software or hardware before deciding to use an accelerator. The hardware accelerator is often attached to CPU/memory buses or via shared memory, like in Figure 3.3.

Figure 3.4 illustrates an example of a hardware accelerator, which could be connected to a processor, like Nios II. Nios II can use the hardware accelerator by activating it to

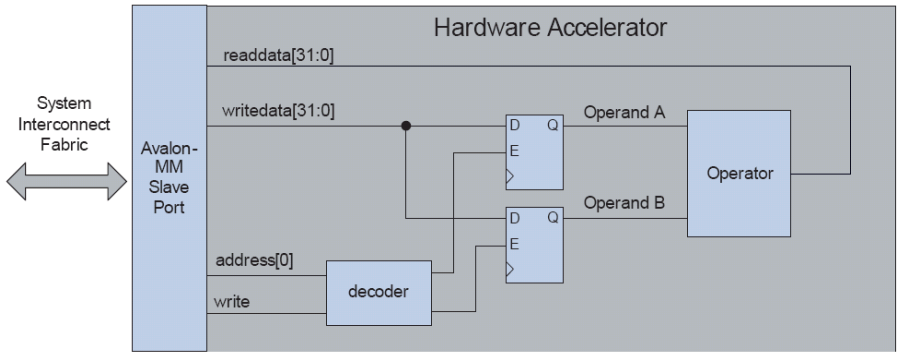

Figure 3.4: Example of a Hardware Accelerator - from [19]

work on data in memory. The processor can then concurrently process other tasks as the accelerator does its job. An advantage of creating an accelerator on an FPGA is that the accelerator itself is easily scalable [16]. Figure 3.3 shows an example of using an accelerator together with a Nios II soft processor. The hardware accelerator is performing a Cyclic Redundancy Check (CRC) in memory, which is much more effective in HW than in SW. To summarize: The hardware accelerator can provide better performance, is less expensive, provide real-time performance and it can fit larger applications that cannot fit on a single chip processor. The cost of implementing it could be a considerable effort and the developer may need a different skill set.

Chapter 4

Video and Images

Video is a medium for displaying pictures in motion and it consists of a series of pictures or frames. These frames are usually displayed in a high enough rate so that the human eye perceives it as fluid motion [39]. The video signals are either analog or digital and can be represented in many different ways. One of the most common ways is by using RGB color representation in combination with some form of compression scheme. Pictures or frames on a TV screen is typically refreshed at a rate of 50-60 times per second and 70-90 times per second on a computer display. For timing control, information called *vertical sync* is used to indicate when the new frame is beginning. Each of the image-frames consists of *scan lines* which are lines of data that occur in vertical sequence on the screen. Additional information called *horizontal sync* is used to indicate when a new scan line is starting. This information can be transferred in different ways. The three most common are:

1. Separate horizontal and vertical sync signals
2. Composite sync signal embedded within the video signal
3. Separate composite sync signal not embedded within the video signal

The composite signal is a combination of both vertical and horizontal sync. Each image from the D5M Terasic camera is surrounded with both vertical and horizontal blanking data around the valid image [56]. This data is used to generate the separte FVAL and LVAL signals used to indicate when the pixel data is available.

4.1 Resolution, frame rate and compression

Based on [39, 31] image resolution is defined as the product between the number of pixels of the picture's height times its width. This product is the total number of pixels available. A standard VGA video at 640 x 480 pixels gives 307 200 separate pixels.

This is equal to a 0,3 megapixel image. Video resolution is often measured in "*lines of resolution*". In essence this is how many distinct black and white vertical lines that can be seen on the display. This number is normalized by dividing the number of lines with the aspect ratio of the display, examples of aspect ratios are 4:3 and 16:9.

Some of the most common video resolutions are:

- Standard VGA: 640x480 = 307 200 = 0,3 Mpixel
- Standard Definition: 720x480i = 345 600 = 0,34 Mpixel or 720x576i = 414 720 = 0,41 Mpixel
- 720p HD: 1280x720 = 921 600 = 1 Mpixel
- 1080p HD: 1920x1080 = 2 073 600 = 2 Mpixel

The resolution is often defined as 720p, 576i or 1080p. This indicates only the number of vertical lines. The "i" and "p" defines how the different picture frames are "painted" on the screen; interlaced or progressive. Progressive means that each frame is completely "painted" on the screen in each refresh. Interlaced video reduces the amount of information sent by half because only half the lines are sent per frame. The complete frame is "painted" by first doing the odd-numbered lines and secondly the even-numbered lines which are then merged together. This can result in a degrading of the quality observed as flickering around sharp edges.

To give an indication of the amount of data a video signal contains, some bit rates based on information from [31] is presented:

- Image (low resolution): 512 x 512 pixel color image x 24bits/pixel(3 x 8 bit per color channel) = 6,3 Mbits/Image
- Video (standard VGA): 640 x 480 pixel color image x 24 bits/pixel x 30 images/second = 221 Mbps
- HDTV (720p): 1280 x 720 pixel color image x 24 bits/pixel x 60 images/second = 1,3 Gbps

Which means that a single layered DVD could only hold about 3 minutes of uncompressed video. This is a motivation for considering using some kind of compression scheme to be able to save even more video of similar quality. Also, if transferring large amount of data on a limited bandwidth channel, compression might be the only way to make it possible.

4.2 Color Spaces, Transformation and Properties

In [39] a color space is defined as a mathematical representation of a set of colors. Some of the most popular color models are RGB and YUV or YCbCr. RGB is an additive color space in which a combination of red, green and blue are put together to create

different colors. Most of the the different types of image output devices today (e.g. TVs, computer displays etc.) uses RGB when presenting color. The RGB color space is not very good when dealing with "real world" images though. To alter the intensity of one color component, all three colors have to be read before calculation and modification. On a computer system this would cost access time and computation time. YCbCr is a color space that consist of the luma component Y, Cb and Cr. Cb and Cr are the blue-difference and red-difference chroma components. This color space is much used in digital image or video representation, e.g. MPEG and JPEG. YCbCr makes it easy to alter the light intensity of the image. This is done by increasing or decreasing the Y channel. Typically, a black and white system would only use the Y, luma information.

Based on the RGB information all the other color spaces can be derived. The basic equations to convert between RGB and YCbCr are:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.1}$$

Equation 4.1 is for transforming between the RGB and YCbCr color domains (as defined in ITU-R BT.601 and [39]). When 8 bits are used to represent each channel, the range is $16 - 235$ for the Y channel and $16 - 240$ for the chroma channels.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0.000 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} (Y - 16) \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix} \tag{4.2}$$

Equation 4.2 is for transforming between the YCbCr and RGB color domains (as defined in ITU-R BT.601 and [39]). When 8 bits are used to represent each channel, the range is $0 - 255$ for each RGB color channel.

4.2.1 Inverting colors

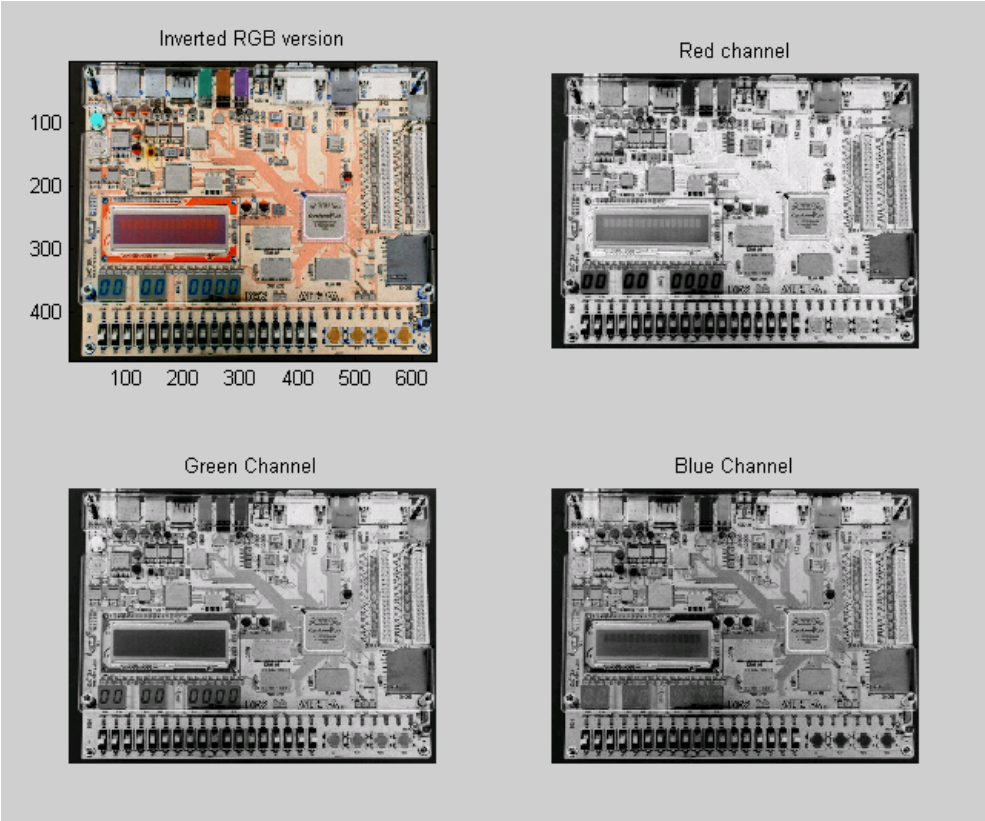

Figure 4.1: An inverted color representation of an image

A common digital resolution of the color components are 8 bits per channel [39, 32]. An RGB picture consists of three color channels, Red, Green and Blue. If the resolution is 8 bits, then the total number of values for each color channel is $2^8 = 256$. By taking 255 and subtracting the current color channel value, we get the inverted color value. Do this for each channel and the color of the picture becomes inverted. Another way of doing this operation would be to invert all 8 bits per channel in hardware. This would result in the same color transformation of the picture. Figure 4.1 shows what a typical inverted RGB image looks like.

4.2.2 Grayscale

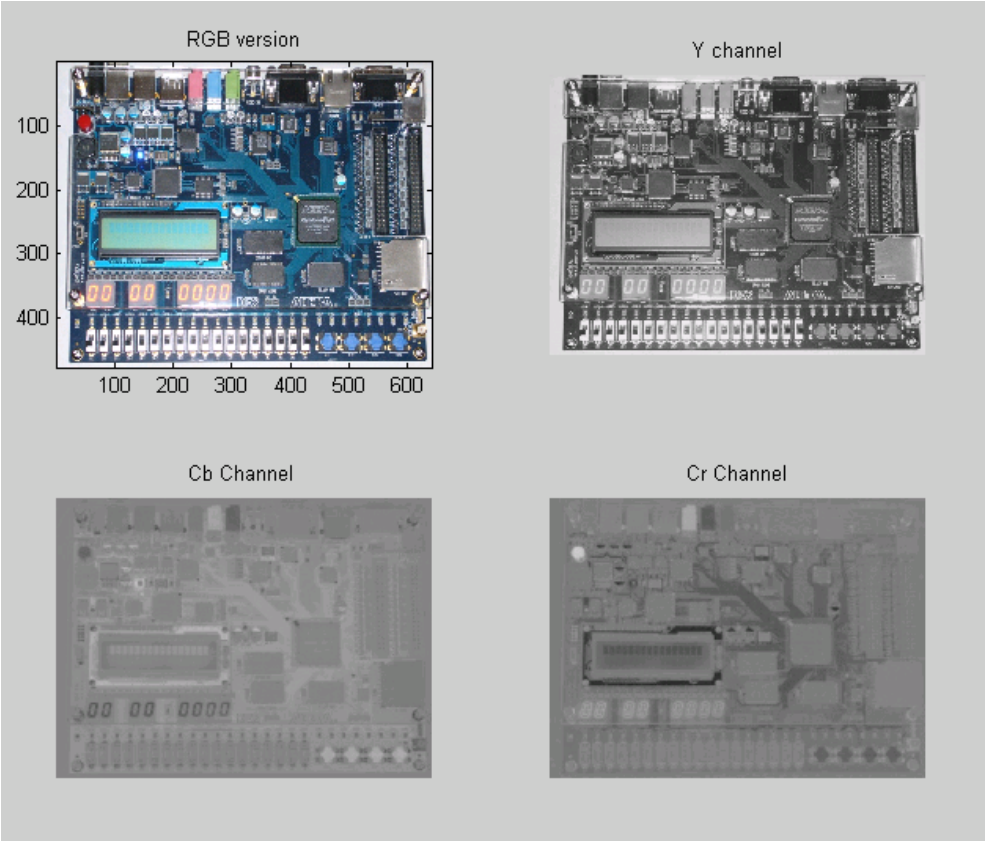

Figure 4.2: A YCbCr color image and its channels

To get a grayscale based video stream one has to do a grayscale conversion. One approach is to use just one of the color components and transmit only one color channel to the VGA DAC as mentioned in [26]. Another approach could be to transform the video from the RGB color space to the YCbCr color space and just transmit the luma (Y) channel. The second approach is much more complex and would increase processing time in a software implementation. In order to be able to display grayscale on the display, the VGADAC has to either alter its hardware configuration, which is not possible in our case, or the luma (Y) value could be transmitted on all RGB channels. This would result in a grayscale RGB representation as seen in Figure 4.2.

4.2.3 Brightness and Contrast

In [39] it is said that working in the YUV or YCbCr color space simplifies the implementation of brightness and contrast control. The contrast in the picture is altered by

multiplying the YCbCr data by a constant. If Cb and Cr are not adjusted, a color shift will result when adjusting the contrast. Brightness control is implemented by adding or subtracting a constant to the luma (Y) component. Brightness adjustments are done before contrast adjustments to avoid introducing a varying DC offset.

4.3 The Processing of Raw Image Data

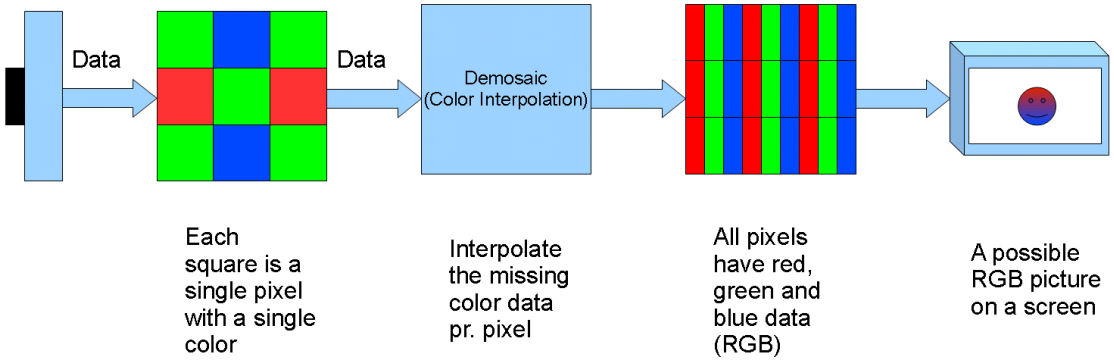

Figure 4.3: The process of transforming raw image data to viewable RGB image data

Figure 4.3 shows the process of transforming raw image data into viewable RGB data. Many cameras today use a CCD image sensor in combination with a color filter array (CFA), which results in a data structure as seen in the figure in [49]. In order to get a full range color image, each pixel needs all three color values. These values are interpolated by use of some form of algorithm. This results in a final image viewable on a display.

4.3.1 Bayer Color Pattern

As introduced in [41, 49] most digital camera sensors use Color Filter Arrays (CFA) to gather the necessary color information. The human eye needs more than light intensity to obtain a full-color image. This would result in an image sensor that needed to carry at least three pieces of information such that the intensity of three independent colors (e.g. Red, Green and Blue) can be deduced. Therefore, to reduce cost, CFA was introduced. Although the resulting image is not full-color, the CFA can be arranged in such a way that the missing color information can be obtained with a reasonable degree of accuracy. This process is known as color interpolation or demosaic. An alternative would be to use three separate sensors to obtain a full RGB range without the need to interpolate, but this would be expensive.

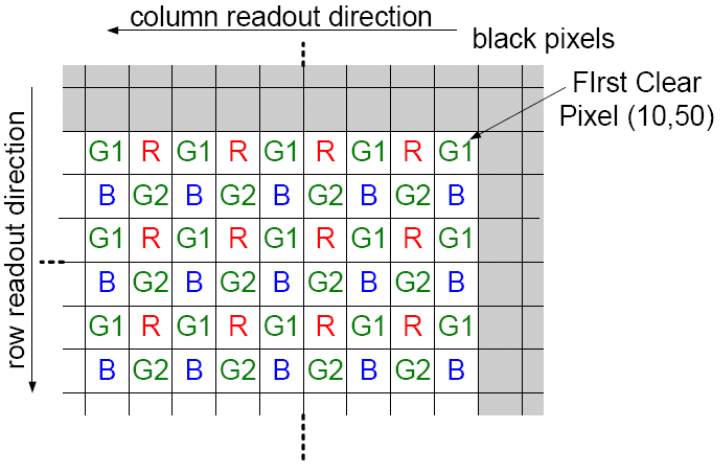

Figure 4.4: Pixel color readout pattern (Bayer) - from [56]

In [56] it is said how the pixel data from the camera is outputted in a Bayer pattern format consisting of four ”colors” - Green1, Green2, Red, and Blue (G1,G2,R,B), thus representing three filter colors. When no mirror modes are enabled, the first row output alternates between G1 and R pixels, and the second row output alternates between B and G2 pixels. The pattern can be seen in Figure 4.4.

4.3.2 Interpolation and Demosaic Algorithms

To display an image from the sensor data, a demosaic algorithm has to be run to gather all three color components for each pixel [49, 32, 52]. There are many types of demosaic algorithms. At the low end there are the so called fill-in-the-missing-data algorithms, in the middle there are linear interpolation algorithms and at the high end there are algorithms that choose an interpolation rule based on image contents (adaptive). The simplest interpolation algorithm, nearest-neighbor, copies the information on a pixel of same color. Another simple interpolation algorithm, bilinear, uses two or four similar colored pixel to calculate an average sum for the unknown pixel. The first and simplest interpolation algorithm is not usable when quality matters, but the second one is better.

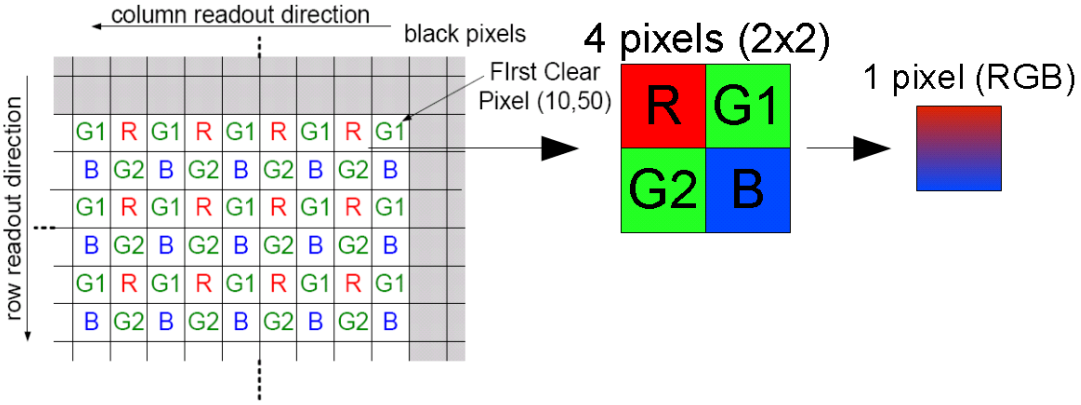

Figure 4.5: Color interpolation by using neighboring pixels to determine one full range RGB pixel - combined with figure from [56]

These algorithms can be implemented in either hardware or in software. The video or image quality depends on how good the algorithm is, which often depends on the avilable processing power. The result is a conversion from a RAW Bayer image pattern to a displayable RGB image. Figure 4.5 shows how pixels are read from the sensor on the camera and color interpolation is done to get a full range RGB pixel.

$$h(x) = \begin{cases} 1 & 0 \leq x \leq 0.5 \\ 0 & 0.5 < x \end{cases} \tag{4.3}$$

Equation 4.3 describes one of the simplest interpolation methods; Nearest Neighbor. It assigns each interpolated output pixel the value of the nearest pixel in the input image [52].

$$h(x) = \begin{cases} 1 - x & 0 \leq x \leq 1 \\ 0 & 1 < x \end{cases} \tag{4.4}$$

Equation 4.4 describes the bilinear interpolation algorithm. It is similar to the ”Nearest Neighbor” algorithm, but in addition this one takes an average sum of the surrounding pixels. This reduces potential for aliasing and image distortion [52].

The Terasic D5M camera supports a function called binning, described in Section 6.3. This function reduces the resolution of the image by averaging pixels together. Figure 4.6 shows how this is done when the camera is set to 2X binning. This also reduces aliasing and gives the possibility to use a less complex interpolation algorithm to get sufficient

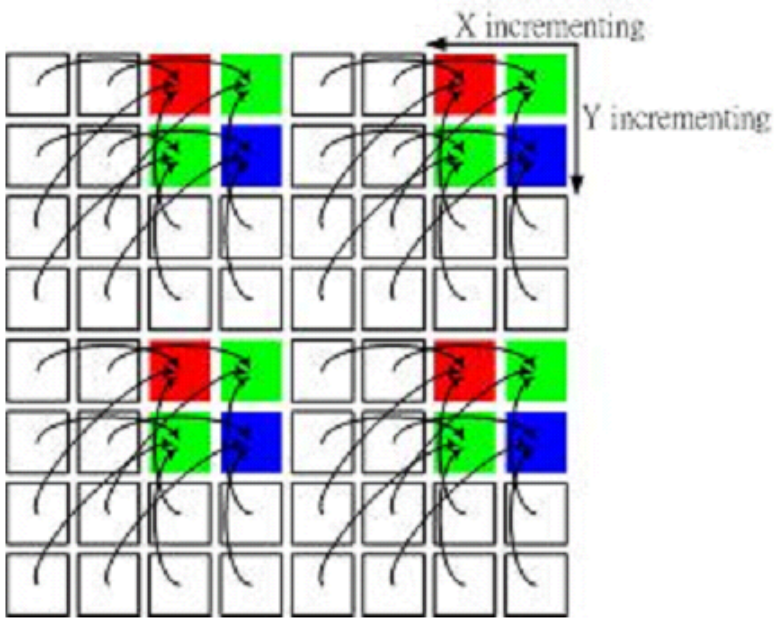

Figure 4.6: 2X binning on Terasic D5M - from [56]

results. Using binning in combination with nearest neighbor would give similar results as with the bilinear algorithm. In [52] it is concluded that the bilinear interpolation algorithm was best when comparing the highest picture quality with best performance.

4.4 Frame Buffer

The memory has many areas of application, one example is as a buffer for temporary storage of data. A frame buffer is a buffer that holds all the information about a picture frame [53]. It is mainly used in a picture or video based system. The picture frame is always available in the buffer, which is read, possibly transformed, and transferred to a display. If the buffer always holds the next frame, and possibly even future frames, this would help against stuttering and picture errors.

Chapter 5

Tools and Software

Thanks to corporations like Altera, Xilinx and Atmel it is now much easier to develop embedded systems. In the past, when the tools and software weren't available, this could have been a much more complex and time consuming job. Tools and software like Quartus II, SOPC Builder and MegaFunctions have been made with the purpose of reducing the design time for the developers. This results in a much shorter time-to-market, which makes using FPGAs, Microcontrollers and the like for development much more attractive. Good tools gives the possibility to greatly improve profit.

5.1 Quartus II

Quartus II is a design software from Altera that provides a complete, multi platform design environment for FPGAs [17]. A typical design flow is: Design Entry, Synthesis, Place and Route, Timing Analysis, Simulation and Programming and Configuration. The Quartus II software includes solutions for all these design phases and a graphical user interface. In the Design Entry process the hardware can either be described completely by using a hardware descriptive language like VHDL, see Section **??**, or in combination with a "drawing" environment, where the designer can place and connect different blocks by using the PC mouse. This tool is very useful and makes interconnection between the modules easy. Synthesis and place and route can be done with either Altera's own tools or the designer can select a different tool like Synplify [8] from Synopsys. Such dedicated tools can provide additional improvements of the design. For simulation, Altera provides a build-in environment. In addition Altera also support the possibility to use other simulation tools inside Quartus II. ModelSim, see Section 5.2, is one of these.

5.1.1 SOPC builder

The SOPC builder is a part of the Quartus II design environment and it lets the designer define and generate a complete system in much less time than usual by including available IP (Intellectual Property) modules in the design[22]. The tool offers both the possibility to include the IP cores in the design as well as the ability to configure them to fit the designer's needs. The IP cores that are available can be found in [9]. In addition to configuring the cores, the SOPC builder also supports a connection function which allows the designer to simply click on how the cores should be connected to each other. SOPC builder then automatically generates a top-level HDL file that connects the different modules together. Altera Nios II soft processor core is implemented by using this tool.

5.1.2 MegaFunctions

MegaFunctions are functional blocks of common off-the-shelf digital design functions that are ready-made and pre-tested [11, 14]. They can easily be included in a digital design, done in Quartus II. The objective of these IP cores is to help the designer save time and energy by not redesigning common functions, but rather let him focus on improving and differentiating his system-level product. There is a variety of different types of cores, e.g. shift-registers, filters, video scalers and others. All cores can be simulated alone or as a part of a bigger system by using e.g. ModelSim (see Section 5.2). Another strength with the MegaFunctions cores is that they are easy to change. Once they have been created by the tool inside Quartus II, the designer can edit the implemented core and change its properties at any time.

5.2 ModelSim

ModelSim is a unified debug environment for hardware descriptive languages (HDL) from Mentor Graphics [7, 47]. The tool uses single kernel simulator (SKS) technology which is said to give it good performance. In the verification process of a design the designer can use ModelSim to simulate and verify the behavior of a hardware description. This is an important step in order to get rid of potential errors and a great way to ensure that the system behave as intended. ModelSim also runs any testbench code written in a HDL. The simulation is displayed as a digital timing diagram like the one in Figure 6.4. Altera delivers a version of ModelSim called ModelSim-Altera Edition that comes with a pre-built library many of the megafunctions available through Altera's MegaFunctions 5.1.2. The ModelSim-Altera comes in three different solutions: Web edition, starter edition and Altera edition. It is the two latter versions that currently are under future development. The main differences are simulation performance, design size limitations and price. Two of the main advantages of using ModelSim over Quartus II simulation tool is its performance, especially in large designs, and its ability to run testbench code.

5.3 Nios II Embedded Design Suite

The Nios II EDS is a fully integrated development environment for developing software to Altera's Nios II embedded processor [20]. The environment is based on the industry's Eclipse IDE. The Nios II specific functionality is included as plug-ins. Following is a list of these plug-ins:

- Nios II Project Manager
- Nios II Software Templates
- Nios II Flash Programmer
- Nios II BSP Editor
- Quartus II Programmer
- Nios II Command Shell

The Nios II EDS provides two distinct development flows and includes many proprietary and open-source tools for creating Nios II programs. Among these tools are the GNU C/C++ tool chain for compiling the C or C++ software language. The Nios II EDS also automates the creation of BSP, see Section 5.3, and functions as an editor for altering this configuration. There are also provided different software templates for examples on how to use the Nios II. The Nios II command shell is used to display messages and it also functions as a terminal when running code on a Nios II processor. The terminal will display any message printed using C library functions like `printf()`. Quartus II programmer makes it possible to program and run code on an implemented processor on an Altera FPGA.

BSP and HAL

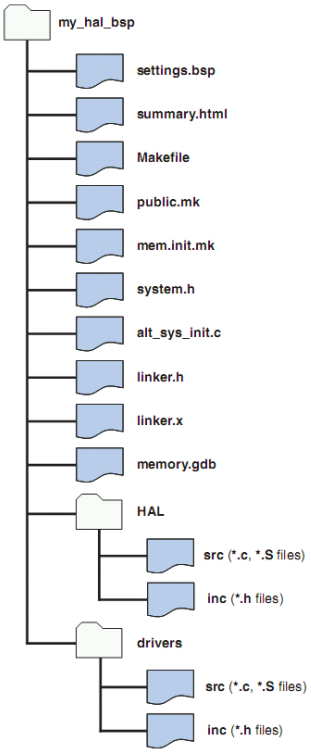

Figure 5.1: HAL BSP After Generating Files - from [20]

BSP or Board Support Packages is a specialized libary that contain system-specific support code for a Nios II project [20]. The BSP isolates the application from the system-specific details. In Nios EDS, se Section 5.3, the BSP editor can be used to look at these specifics and possibly change the system configurations. Inside the BSP editor it is also possible to see the memory map and a list of the peripheral devices connected to the Nios II processor. Figure 5.1 shows file hierarchy inside a generated BSP project and how they are structured. If a design uses more than one Nios II processor, then each processor would generate its own BSP project. A BSP contains the following elements:

- Hardware Abstraction Layer
- Newlib C Standard Library
- Device Drivers
- Optional Software Packages
- Optional Real-Time Operating System

As mentioned, a element of the BSP library is HAL. HAL is a lightweight runtime envi-

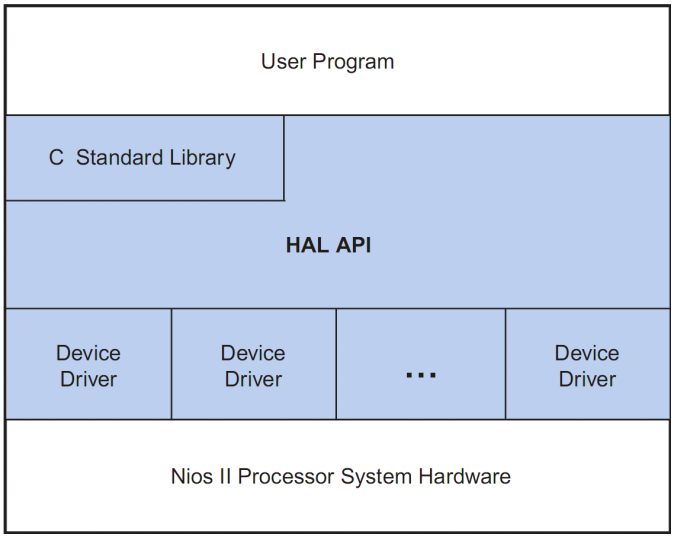

Figure 5.2: The layers of a HAL-Based System - from [20]

ronment that provides a simple driver interface for programs to connect to the underlying hardware. Figure 5.2 shows how this software communication layer is implemented between the application program and the Nios II processor hardware. The HAL API is integrated with the standard C library which allows the use of familiar C library functions like `printf()`, `fopen()`, `fwrite()`, etc. to access devices and files in the underlying hardware. The tight integration between SOPC builder and the Nios II software development tools provides an automatic construction of HAL instances for the hardware generated by SOPC. This makes it very easy for the developer to alter a hardware configuration and then be able to run the same application on the new hardware by just updating the Nios II BSP project (which also will update HAL).

Chapter 6

Development Platforms and Hardware

There are several corporations that provide programmable circuitry like microcontrollers and FPGAs. Altera, Xilinx and Atmel are examples of this [25, 62, 1]. All of these corporations also provide development tools and platforms for their customers. The tools and platforms have many purposes. One of these purposes is to demonstrate potential area of application, another important purpose is making development easy.

6.1 Available platforms

Both Altera and Xilinx provide a range of different development kits [5] and [6]. The kits differ in complexity, hardware components and area of application. Many of the development kits provided are created by third party companies like Terasic [57]. Terasic is the developer of the Altera DE2 platform and many other boards with Altera FPGAs. In addition to the main boards they also supply a range of daughter boards with specific areas of application. Examples of such boards could be cameras, touch screens, network cards and much more. A similar platform to the Altera DE2 from Xilinx would be the Spartan-6 LX150T FPGA Development Board. It includes many of the same hardware components and it is well suited for video purposes. Avnet Spartan-6 FPGA Industrial Video Processing Kit includes both the LX150T FPGA development board, image sensor and more.

6.2 DE2 and Components

The Altera DE2 is a platform intended for development and education. It provides many different kinds of features as shown in Figure 6.1, which makes it a versatile board with

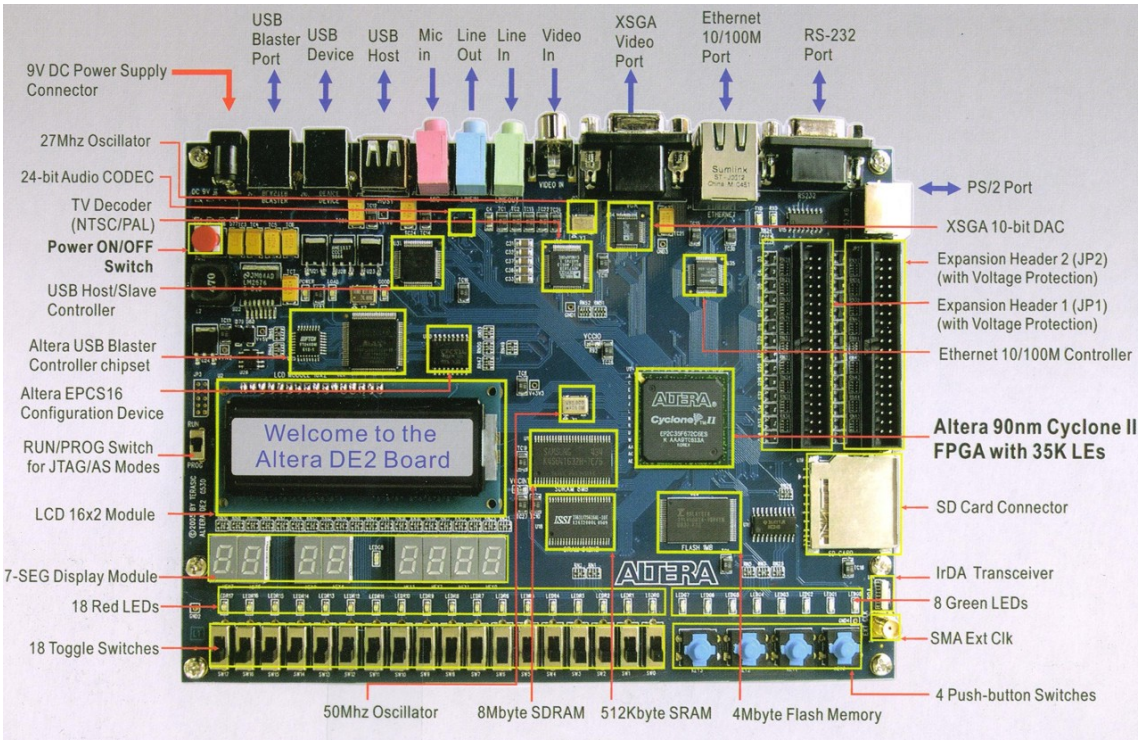

Figure 6.1: Altera DE2 development board - from [12]

many possibilities [12]. The heart of the board consists of an Altera Cyclone II 2C35 FPGA which is one of Altera's low-cost FPGAs [13]. The FPGAs output pins are all connected to the important components on the board allowing the designer full control. For simple communication with the environment or for user-interaction there are several LEDs and switches. There are two types of switches: Push buttons and stationary toggle buttons. To display characters and numbers there are a 16x2 character display and eight 7-segment displays. For experiments that need some kind of digital communication there are several communication interfaces RS-232, PS/2, USB 2.0, 10/100 Ethernet and an infrared (IrDA) port. For making systems that use sound or video the DE2 board also have connections for microphone, line-in, line-out (24-bit audio CODEC), video-in (TV Decoder) and VGA (10-bit DAC) for connecting to displays. It also has an SD memory card connector and separate SRAM, SDRAM and Flash memory chips for storage. In addition there are expansion slots for connection of other user defined boards.

6.2.1 The FPGA - Cyclone II EP2C35

In the Altera product catalog [15] three different types of FPGAs are described. The Stratix series, Arria series and the Cyclone series. The Cyclone series focuses on low-cost and low-power and is therefore well suited for high-volume applications. In [13]

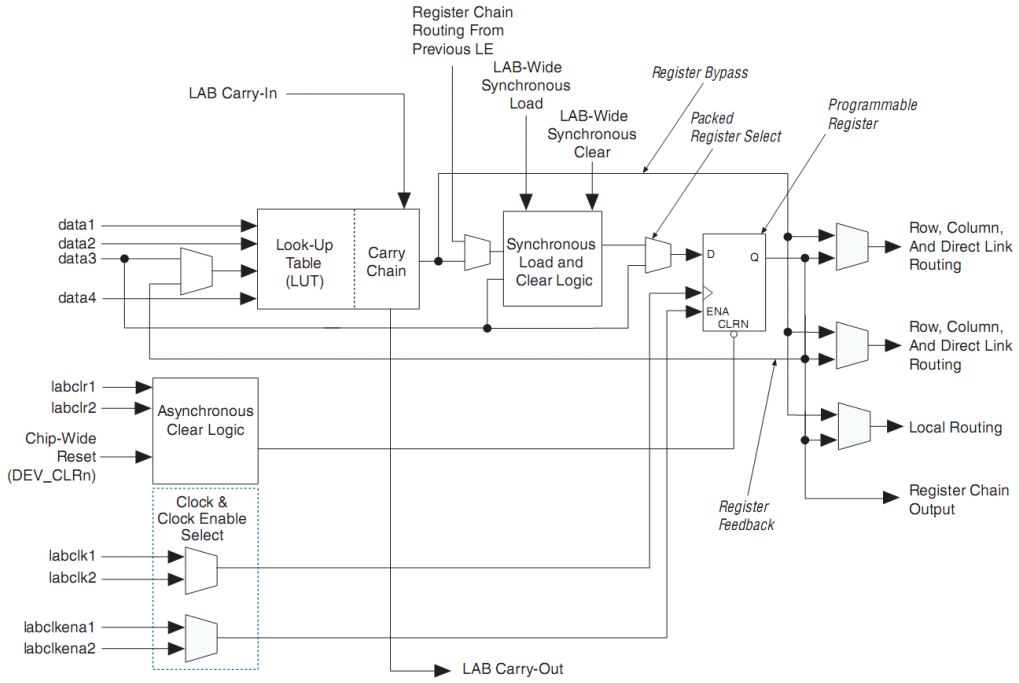

Figure 6.2: Logic Element in Cyclone II - from [13]

the Cyclone II series is described in detail. From this data sheet it is gathered that the Cyclone II EP2C35, located on the DE2 board, has 33 216 Logic Elements (LEs). These elements, as shown in Figure 6.2, are the fundamental logic building blocks of the FPGA. The logic elements include a 4-input look-up table (LUT), programmable register, and a carry chain connection. The LUT can implement any type of combinatoric logic and the programmable register can be used to hold information and implement different kind of flip-flops. The carry chain connection can be used when implementing an adder which needs to send carry out. The Logic Elements are distributed on the FPGA into Logic Array Blocks (LAB). Each block consists of 16 LEs. The LABs are placed in a two dimensional row and column based grid and interconnected with different connections which support different speeds. In addition to the LAB there are also embedded memory blocks and embedded multipliers present. The EP2C35 has a total of 105 M4K on-chip RAM blocks and 483 840 total RAM bits, and 35 embedded multipliers. The multipliers are 18 x 18 bit each configurable as two independent 9 x 9 bit multipliers with up to 250 MHz clock performance. The device also supports four Phase Locked Loops (PLLs). The PLLs provide clock multiplication and division, phase shifting, programmable duty cycle and external clock outputs, allowing system-level clock management and skew control. For connection, the 2C35 has 475 user I/O pins and are made in a 672-pin FineLine BGA package.

6.2.2 The VGA DAC - ADV7123

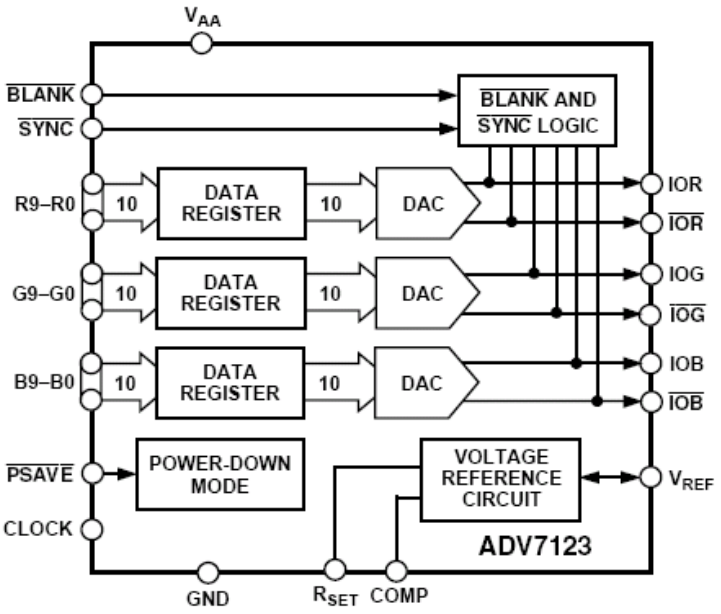

Figure 6.3: Functional Block diagram of the VGA DAC - from [26]

The ADV7123 device features high speed conversion of triple 10-bit digital video [26]. As input, see Figure 6.3, it takes three color components Red, Green and Blue (RGB, see Section 4.2 for more details), each with a digital resolution of 10-bit. This data is converted to the analog domain by three DACs and sent to a 15-pin high-density D-sub connector. This connector can be connected to a common LCD or CRT screen with a standard VGA cable. To synchronize the analog output the chip uses logic in combination with the inputs $\overline{BLANK}$ and $\overline{SYNC}$. Every time the CLOCK has a rising edge the chip latches the RGB data inputs, $\overline{SYNC}$, $\overline{BLANK}$ and control inputs to registers. The screen resolution is based on the frequency of this CLOCK signal. This chip supports a wide range of resolutions, all the way up to 1600 x 1200 at a 100Hz refresh rate.

In [26] all the functions of the VGA DAC is described. The most important ones like $\overline{BLANK}$, $\overline{SYNC}$, $\overline{PSAVE}$ and CLOCK will be described in more detail. Inputs like Vref and connection of external passive components to the chip is taken care of on the DE2 board.

$\overline{BLANK}$ and blanking level

The blanking level is the level separating the sync portion from the video portion of the signal. If the input on $\overline{BLANK}$ is equal to logical zero, the RGB inputs will be

disregarded and result in an ignorance of the RGB inputs and the analog output will be driven to the blanking level. If this level is 0 IRE the result will be the darkest possible picture.

$\overline{SYNC}$ and sync signal

The sync signal is the position of the composite video signal that synchronizes the scanning process. A logical zero on the $\overline{SYNC}$ input switches off a 40 IRE current source. This is internally connected to the IOG analog output. The $\overline{SYNC}$ signal does not override any of the other control signals, so it should only be asserted during the blanking interval. The sync level is defined as the peak of the $\overline{SYNC}$ signal.

$\overline{PSAVE}$

This is the power save pin and can save power when activated.

CLOCK

The rising edge of this clock signal latches the color and control inputs. It is typically the pixel clock rate of the video system. To calculate the required clock for a defined resolution one has to use Equation 6.1.

$$Dotrate = (HorizRes) \times (VertRes) \times (RefreshRate)/(RetraceFactor) \tag{6.1}$$

In Table 6.1 and Table 6.2 there are listed some of the possible resolutions and their horizontal and vertical timing specifications. These are gathered from [12].

6.2.3 The I/0

The Altera DE2 board has a total of 18 toggle switches which, for instance, can be used to activate functionality or as user inputs [12]. When a switch causes logic 0 it is stated in the DOWN position, which is closest to the edge of the board. The logic 1 state is when the switch is in the UP position. In addition there are four push-buttons. All buttons are debounced by a Schmitt trigger circuit, transferring a clean logic 0 or 1 to the input of the FPGA. The button is normally high and takes an active low state when pushed. The buttons can be seen in Figure 6.1.

VGA mode		Vertical Timing Spec			
Configuration	Resolution (HxV)	a(lines)	b(lines)	c(lines)	d(lines)
VGA(60Hz)	640x480	2	33	480	10
VGA(85Hz)	640x480	3	25	480	1
SVGA(60Hz)	800x600	4	23	600	1
SVGA(75Hz)	800x600	3	21	600	1
SVGA(85Hz)	800x600	3	27	600	1
XGA(60Hz)	1024x768	6	29	768	3
XGA(70Hz)	1024x768	6	29	768	3
XGA(85Hz)	1024x768	3	36	768	1
1280x1024(60Hz)	1280x1024	3	38	1024	1

Table 6.1: VGA vertical timing specification - from [12]

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(Mhz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25 (640/c)
VGA(85Hz)	640x480	1.6	2.2	17.8	1.6	36 (640/c)
SVGA(60Hz)	800x600	3.2	2.2	20	1	40 (800/c)
SVGA(75Hz)	800x600	1.6	3.2	16.2	0.3	49 (800/c)
SVGA(85Hz)	800x600	1.1	2.7	14.2	0.6	56 (800/c)
XGA(60Hz)	1024x768	2.1	2.5	15.8	0.4	65 (1024/c)
XGA(70Hz)	1024x768	1.8	1.9	13.7	0.3	75 (1024/c)
XGA(85Hz)	1024x768	1.0	2.2	10.8	0.5	95 (1024/c)
1280x1024(60Hz)	1280x1024	1.0	2.3	11.9	0.4	108 (1280/c)

Table 6.2: VGA horizontal timing specification - from [12]

6.2.4 The SDRAM - S29AL032D

There are different types of memory located on the DE2 card, as stated in [12]. This memory is a chip named S29AL032D and it consists of 4 banks with 1 Mega x 16 bits (equal to 2 Mega bytes) which gives a total of 8 MB storage capacity. The memory type is called SDRAM (Synchronous Dynamic RAM) and it is accessible as memory for the Nios II processor.

6.2.5 The FPGA-OnChip Memory

All Cyclone devices features embedded memory blocks which can be configured to be used as "OnChip" memory [13]. In Section 6.2.1 it was stated that the EP2C35 has a total of 105 M4K on-chip RAM blocks and 483 840 total RAM bits. The M4K memory blocks are very fast and flexible and can be used to support a wide range of system requirements. The memory blocks can be used in many different modes, including single-port, dual-port, shift-register, as ROM (Read Only Memory) and as FIFO. On-chip memory can be used to save and run Nios II software instructions.

6.2.6 Megafunction - Shift Register

Shift registers are a cascade chain of flip-flops that are connected to each other together with a common clock signal [40, 14]. When the shift register is activated it will shift the data from the input through the chain of flip-flops. The data will be shifted in the rate of the clock signal. The first data of the output will be delayed equal to the number of flip-flops in the chain. Shift registers can have both parallel and serial inputs and outputs. They are often configured in either SISO (Serial-In Serial-Out), SIPO (Serial-In Parallel-Out) or PISO (Parallel-In Serial-Out). Shift registers can serve many purposes. They be used to convert between serial and parallel interfaces, as memory or delay element, multiplicator or divider (by shifting data right or left the data would be multiplied or divided by two).

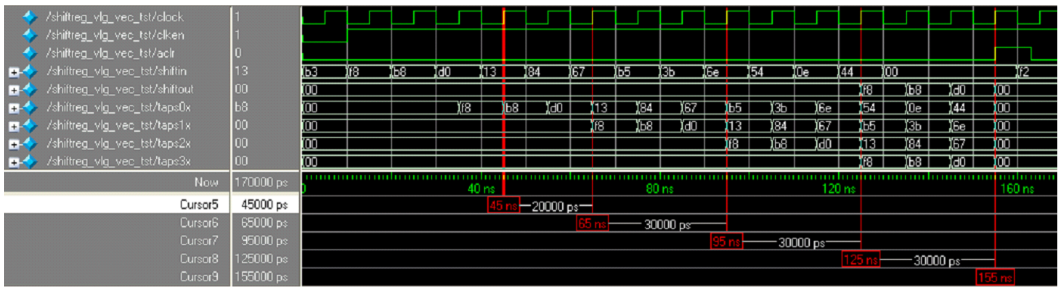

Figure 6.4: Megafunction shift register (ALTSHIFT_TAPS) - from [14]

Figure 6.4 shows a timing simulation of the RAM based shift register from Altera (Megafunction ALTSHIFT_TAPS). The shift register is implemented by using dual-port M4K memory elements in shift register configuration (see Section 6.2.5). The configuration of the shif register depends on the `TAP_DISTANCE`, `NUMBER_OF_TAPS` and `WIDTH` parameters. All parameters can be selected inside the Megafunction Wizard in Quartus II. Figure 6.4 is configured with `TAP_DISTANCE=3`, `NUMBER_OF_TAPS=4` and `WIDTH=8 bits`. The data is shiftet in serial with a bit width of 8 bits, on input `shiftin` in the rate equal to the `clock` signal. When `clken` is logic "1" the input is read and the shifting begins. The signals `taps0x to taps3x` outputs data with a distance equal to the `TAP_DISTANCE`. Since all taps are internally connected in cascade `taps3x` will result in a delay of `TAP_DISTANCE` $\times$ $tap - number$. In this case the delay is 9 clock cycles.

6.2.7 Nios II Floating-Point Unit

$$significant\ digits \times base^{exponent} \tag{6.2}$$

Many numbers are too large or too small to be represented by integers. One solution is to use a system called floating point [40, 36, 24]. Every floating point can be described as a combination of the variables in Equation 6.2. The term *floating point* refers to the fact that the radix point or decimal point is floating. The point is set based on the value of the exponent. In order to be able to describe decimal numbers with integers it is very common to use a form of fixed point representation. This means that the radix point is predetermined and fixed. This would allow integers to represent a range of decimal numbers, but the range is fixed by the radix point.

	Sign	**Exponent**	**Fraction**	**Bias**
Single Precision	1[31]	8[30-23]	23[22-00]	127
Double Precision	1[63]	11[62-52]	52[51-00]	1023

Table 6.3: Storage Layout of IEEE floating point numbers - values from [36]

Table 6.3 displays how the storage layout of floting point number is based on IEEE [36]. The sign bit indicates the sign of the number. Logig "0" is a positive number and logic "1"

is a negative one. The exponent field represent both negative and positive exponets. To represent this the actuall exponent is a combination of the bias and the stored value. If the stored value is 200 the actual exponent is equal to $(200-127)$ 73 for single precission. In genereal this gives that the "real" exponent is $value - bias$. Double precision uses a total of 64 bits and single precision uses 32 bits. The composition of the fraction bits and an implicit leading bit is called mantissa or significand and represents the precisions bits of the number.

Target FPGA Device	Addition	Subtraction	Multiplication	Division
EP3C120	20 times	18 times	17 times	12 times
EP3SL150	18 times	19 times	12 times	13 times

Table 6.4: Sample Floating-Point Custom Instruction Acceleration Factors - values from [24]

The Nios II embedded processors, presented in Section 2.2, supports floating-point operations [24]. The floating-point operations can either be emulated in software by the GNU C/C++ compiler or processed as a custom instruction acceleration (see Section 3.2.1). The architecture supports single precision floating-point instructions as specified by the IEEE Std 754-1985 [36]. The floating-point custom instructions can be added to the Nios II processor inside SOPC builder (see Section 5.1.1) and gives support to floating-point addition, subtraction, and multiplication. Floating-point division is also available, but as an extension to the basic instruction set. It can also be added inside SOPC. If double precision floating-point is used this would be emulated in software and would give reduction in performance. Table 6.4 shows some sample performance increase factors when using the custom instructions instead of only software emulation.

6.3 The Camera - Terasic TRDB D5M

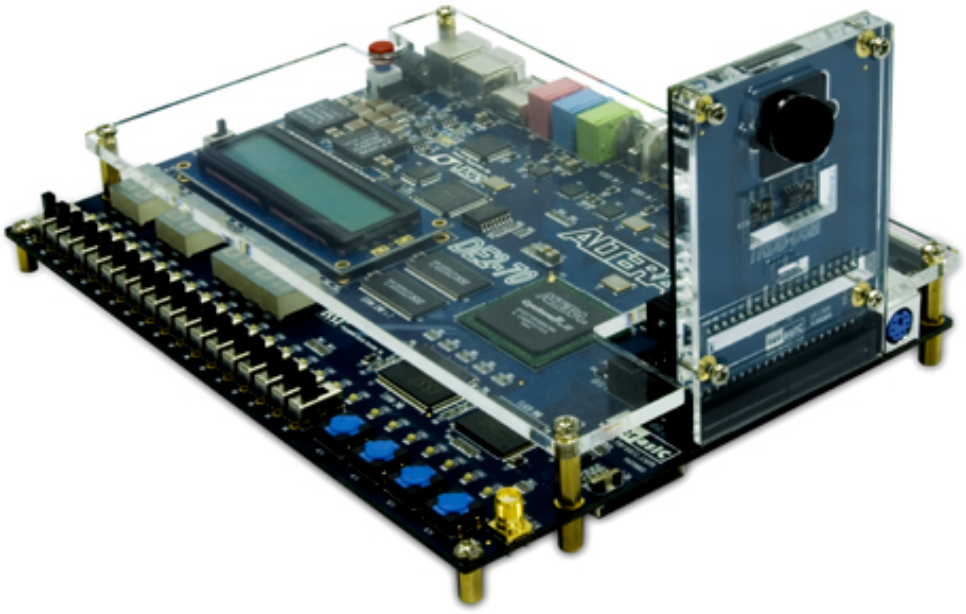

Figure 6.5: The Terasic Digital Camera with the Altera DE2 - from [57]

This device, as shown in Figure 6.5, is made by Terasic for use in combination with Altera's DE3/ DE2_70/ DE2/ DE1 and Cyclone II starter boards. General information about this camera can be found in [55], and a more detailed description about hardware is found in [56]. Only main features are going to be described in this section. More details will be presented in Section 4, which describes how the raw image data is presented. Together with the D5M camera Terasic has given a totally functional hardware description in Verilog that demonstrates how the camera can be used together with the DE2 board to show captured image data. The demonstration also supports a VGA screen to show the pictures taken while the camera is running in the default mode called ERS Continuous. In this mode picture frames are produced continuously at the frame rate defined by the parameter ${}^{t}FRAME$. Electronic Rolling Shutter (ERS) is used, and the exposure time is electronically controlled to be ${}^{t}EXP$.

$$\begin{aligned} W &= 2 \times ceil((Column\_Size + 1)/(2 \times (Column\_Skip + 1))) \\ H &= 2 \times ceil((Row\_Size + 1)/(2 \times (Row\_Skip + 1))) \end{aligned} \tag{6.3}$$

Following is a summary of some of the important features of the camera:

- The ability to run many different resolutions and frame rates in ”video” mode
- Supports widescreen resolutions, 720p60 and 1080p30, as well as resolutions up to 2592x1944 at 15 fps
- Very configurable and flexible
- Programmable controls: Gain, frame rate, frame size and exposure.
- Uses a two-wire serial interface (I^2C) for configuration
- Connects directly to the DE2 board in the expansion slot.
- Both hard and soft reset. Soft reset enables quickly reset of most registers
- Integrated PLL for clock frequency multiplication and division, see Section 6.3.
- Support for different readout modes: Skipping and Binning.
  - Skipping - reduces the output resolution without affecting the field-of-view. It does this by not sampling entire rows and columns of pixels. A skip 2X mode skips one Bayer pair of pixels for every pair of output. See Equation 6.3 for the resulting width (W) and height (H) after skipping
  - Binning - Can reduce the effect of aliasing introduced by use of the skip modes by averaging of 2 or 3 adjacent rows and columns (adjacent - same-color-pixels). More about this can be found in Section 4.3.2

Pixel clock and PLL

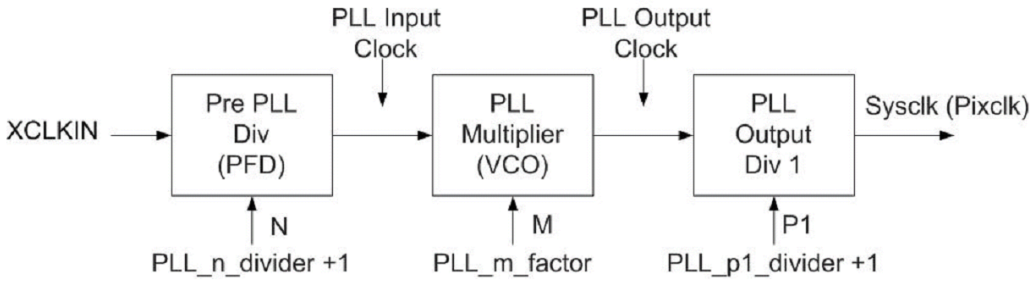

Figure 6.6: Shows how the camera's PLL is implemented - from [56]

$$PIXCLK = \frac{(XCLKIN \times M)}{(N \times P1)} \tag{6.4}$$

The PLL (Phase-Locked Loop) in [56] is used to generate a pixel clock signal which corresponds to the rate of the readout of pixel data. To alter this frequency the values

of the correspondence data registers are changed. This is described in further detail in Section 9.3. Equation 6.4 describes what the final pixel frequency will be based on the values of the adjustable parameters.

Chapter 7

Communication

The Oxford Dictionary of English defines communication as: ”the imparting or exchanging of information by speaking, writing, or using some other medium”. For an embedded system, communication is a very important part of the system. Without the possibility to talk to the world ”outside” or to communicate with different peripherals inside, the system would not serve any purpose.

7.1 Altera Avalon PIO

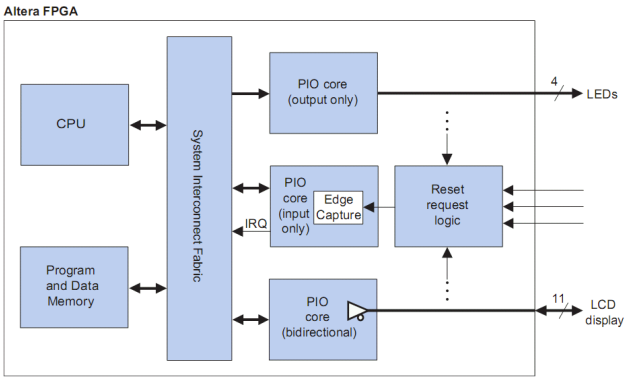

Figure 7.1: NIOS II sample system with multiple PIO cores - from [21]

The Altera Avalon Parallel Input/Output (PIO) core is an interface that easily can be connected to the Nios II embedded processor [21]. This is done by using SOPC builder as described in Section 5.1.1. The core itself provides one to 32 I/O ports. Figure 7.1 shows an example of how the PIO core connects the system interconnection fabric together with the other logic. The PIO core can be used to connect to peripheral logic inside the FPGA (on-chip) or to connect to other devices outside the FPGA (through external

pins). The PIO core is connected to the Nios II processor by memory mapping the PIO core in its address space. This means that Nios II does not make any distinction between memory devices and I/O devices. Accessing the I/O device is done by accessing the specific address that the I/O device occupies [30]. When an access is made with Nios II the communication is done through the system interconnection fabric. The PIO core can be used in many ways, e.g. controlling LEDs, acquiring data from switches, configuring and communicating with off-chip devices and more.

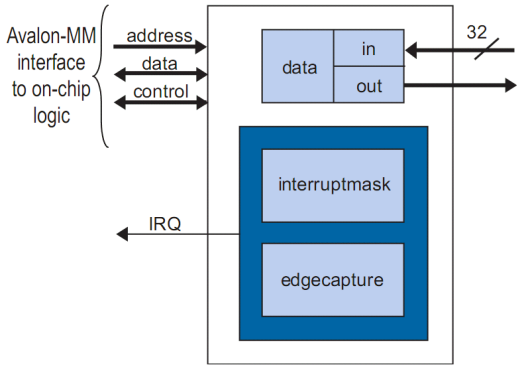

Figure 7.2: PIO core - from [21]

SOPC builder, described in Section 5.1.1, can be used to add a PIO core to a Nios II system. When adding the core it is possible to choose between different options on how the core should operate. Figure 7.2 displays a core for input or output operation. It is also possible to select a core for bidirectional operation (in-out). The core in Figure 7.2 displays a dataregister that can be used to read or write data to the ports. In addition there is also a separate part of the core to support Interrupt (see Section 2.3). Interrupt can be either ”level-sensitive” or ”edge-sensitive”. When it’s ”level-sensitive” it means that it will generate an interrupt whenever a ”high” level is registered. If its set to ”edge-sensitive” the core can generate an interrupt on either rising edge, falling edge or either edge. The `interruptmask` register is used to select which of the ports should trigger an interrupt. The bits in the register that are logic ”1” will activate sensitivity on the corresponding port.

7.2 I2C

From [30] I^{2}C is described as a very cheap yet effective bus-network used for intercommunication between peripheral devices in small-scaled embedded systems. It has been used for more than 20 years and there are many devices supporting this bus-network. The camera, described in Section 6.3, uses I^{2}C to configure the camera by altering values in the data registers.

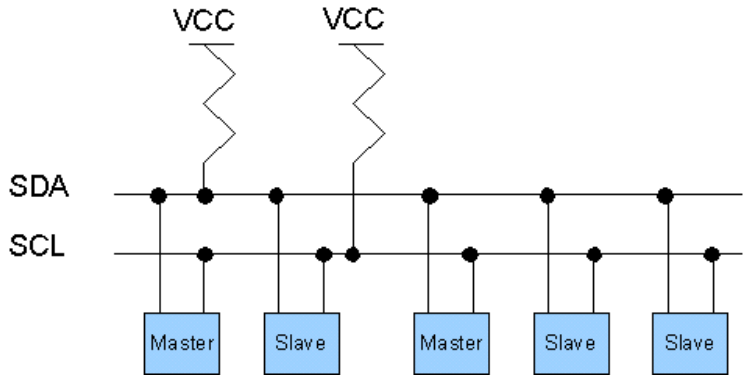

Figure 7.3: I^{2}C bus system - from [30]

The I^{2}C is based on a two-wire system, one wire for synchronization SCL (Serial Clock) and the other wire for data SDA (Serial Data). Both lines are pulled up by two resistors to Vcc (the supply voltage) which gives them the default digital value of logic ”1”. The different devices are connected as shown in Figure 7.3. One of the advantages in comparison with other serial communication interfaces is that a device can be connected and disconnected without interfering with the other devices that are communicating. This property gives the system flexibility. The devices that are connected can either be a master (I^{2}C supports multiple masters), which can initiate a transmission, or a slave which can only receive data.

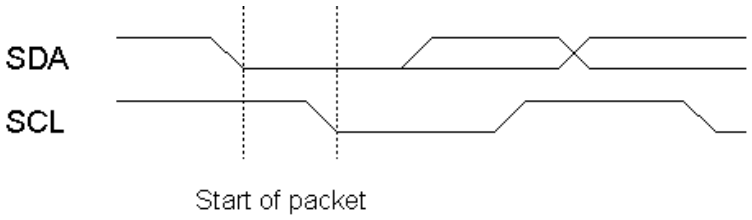

Figure 7.4: I^{2}C ”start condition” - from [30]

The idle state for the system is when both SDA and SCL is high. This means that none of the devices are pulling the lines towards ground. An I^{2}C connection is initiated by the master by first making SDA low followed by SCL. This is called ”start of packet”.

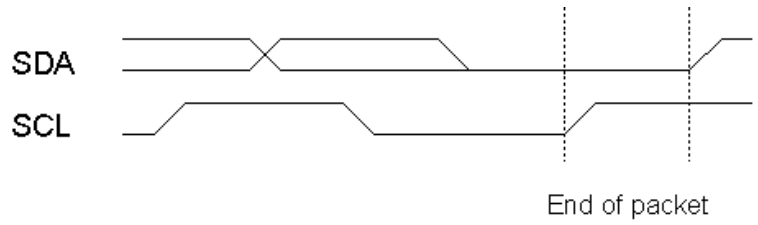

Figure 7.5: I^{2}C "stop condition" - from [30]

To stop an I^{2}C transmission the steps are simply reversed. SCL are let high followed by SDA. This is called the "stop condition". The two conditions are shown in Figure 7.4 and Figure 7.5.

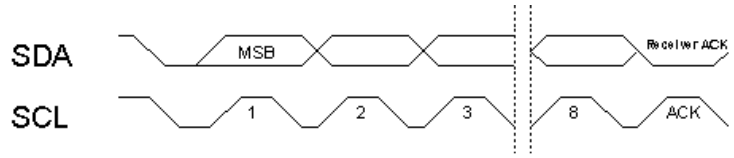

Figure 7.6: Single I^{2}C packet with receiver acknowledge - from [30]

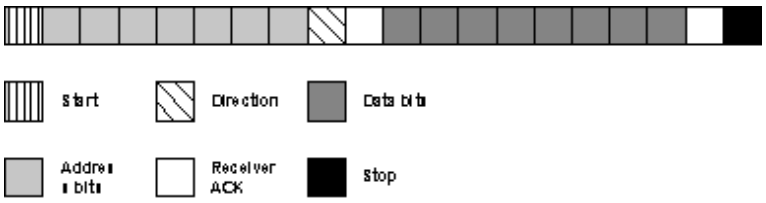

Figure 7.7: Complete I^{2}C packet, slave address, direction and data - from [30]

A typical I^{2}C packet consists of the above mentioned "start of packet", a unique slave address, a direction bit, receiver ACK, the data, a new receiver ACK and last a "stop condition". Figure 7.6 shows a single I^{2}C packet with a receiver acknowledge. Figure 7.7 shows a typical complete packet. This includes the "start of packet", the address of the slave, direction bit - read or write, receiver ACK, data bits, receiver ACK and finally the "stop condition". If the address is chosen as 0x00 this means that the message can be received by all devices. This is called a "general call address". This could be used by a master to find out which slaves are available. More information can be found in [30].

Chapter 8

Pedagogics

In any kind of teaching and learning environment the main objective should be to present knowledge in a way that has the best learning effect on the one receiving it. The theory around this field is called pedagogic [38] and it is an important tool in this environment. By changing the way knowledge is presented and taught it is possible to get very different results in relation to what the observer has learned. When demonstrating electronic systems, the use of these tools combined with the experience of others can greatly improve the success of the demonstration.

8.1 Embedded Systems in Education

There are several people researching the field of how to teach about embedded systems and microprocessors [33, 44, 27]. One of their common goals is to find an efficient way to provide students with sufficient knowledge of the topic.

In [33] it is suggested to change a curriculum in order to make it possible for computer engineers to embrace the field of embedded systems. Pedagogically, the courses about embedded systems allow the students to explore the trade-offs between different design modalities (i.e., hardware versus software, digital versus analog circuitry, dedicated versus reconfigurable hardware, etc.) and investigate optimal solutions given a set of constraints placed on the system. It is also said that in order for the students to develop their skills in the embedded systems discipline it is not sufficient with single courses, but a more extensive program-level curriculum is needed to adequately educate students in embedded systems design. This emphasizes that the field of embedded system is very extensive.

The two courses described in [44] uses both soft and discrete microprocessors to support their teaching in their introductory course to microprocessors and embedded systems. The soft processor in combination with the reconfigurable nature of the FPGA provides great flexibility to demonstrate many different aspects to the field of embedded systems.

The courses use projects and a practical approach to get the students to learn.

Presented in [27] are experiences about important teaching skills and concepts for embedded system design. These experiences have been gathered from courses provided in the field of embedded systems. The courses have been given great feedback from students that followed them. What they have experienced is that a theoretical approach is not sufficient. The hands-on sessions are indispensable to get the student to really understand the theory. Although the course focuses on complex system design, the practical examples are kept small and simple enough to be able to explain the basic concepts without losing focus because of the complexity of the problem. The focus is always held on one small design aspect at the time with additional explanation of how the small parts can be used in a more complex manner. Key factors to obtain this good learning environment are enthusiasm and motivation from the teacher, interactivity and real world examples.

8.2 Pedagogic in Teaching and Learning

In [38] and [37] knowledge of pedagogic and psychology in the field of teaching are presented. [38] focuses on how the teacher should perform the educating to get the best results, and [37] focuses more on the psychology of the pupils. This information is important to consider in any form of educating environment.

One part of [38] presents some general guidelines and principles to consider when teaching. It is important that the teacher motivates the pupils before teaching. A form of motivation could be to give the pupils some examples of how the theory could be used in everyday life. This might give the pupil a meaning as to why it is important and thereby motivate him or her. Another important thing to be aware of when teaching, is that the pupils learn best when they can be an active part of the education and not just passive listeners, for instance by doing exercises, which gives a practical approach to the theory. The importance of teaching and learning with use of an active and practical approach is supported by [54]. In order to maximize the student's achievement the instructors should not allow them to remain passive while they are learning. One way to get them more actively involved is to structure a cooperative interaction. It is also important to present the theory in a concrete and clear fashion to avoid mix-ups [38]. In addition it is also important to follow the principal of focus. This is done by emphasizing the central parts of the subject. The focus is defined as what binds the part we understand with the new theory. Hence, it is important to individualize the teaching - making it suitable for the individual subject to keep the focus. It is also important to remember that the pupils have different background and basis for understanding the theory, and take this into consideration when teaching.

In [37] it is presented pedagogical psychology based on the pupils' perspective. One important theory from this book is about differentiating. Differentiating is about dividing the pupils into groups based on their age before adapting the teaching. This emphasizes

the importance of age and that it should be an important factor to consider when choosing how and what kind of theory to present.

Chapter 9

Implementation and Discussion

In the pre-study described in the project thesis there was made a specification of an embedded demonstrator on the Altera DE2 platform. It was decided that the chosen demonstrator should present and manipulate recorded video, both in hardware and in software. The demonstrator's purpose is to educate about the importance of hardware/-software codesign in addition to presenting an example of an electronic system. The demonstrator that was chosen was based on the study of many different alternatives. The two most promising candidates were the video demonstrator and a demonstrator for visual demonstration of audio properties. Both demonstrators used a visual concept to educate and were supposed to be able to interact with the participants. The reason for choosing a visual and interactive demonstrator was based on pedagogical aspects. These aspects are presented in Section 8. From the field of pedagogics it is said that a practical approach to education is much more effective than theory alone. In addition, by letting the observer take an active part of the demonstration and visually see the results it is easier for the participant to come to his or her own conclusion. It was also suggested that the practical examples demonstrated should be kept small and simple enough to be able to explain the basic concepts without losing focus because of the complexity of the problem. The embedded video demonstrator uses video as a medium for demonstration, which makes it easier for the participant to relate the demonstrator to real world examples. The embedded video demonstrator presented a better solution altogether, it had more available documentation and a functional hardware description which would help finalize it in the available timeframe.

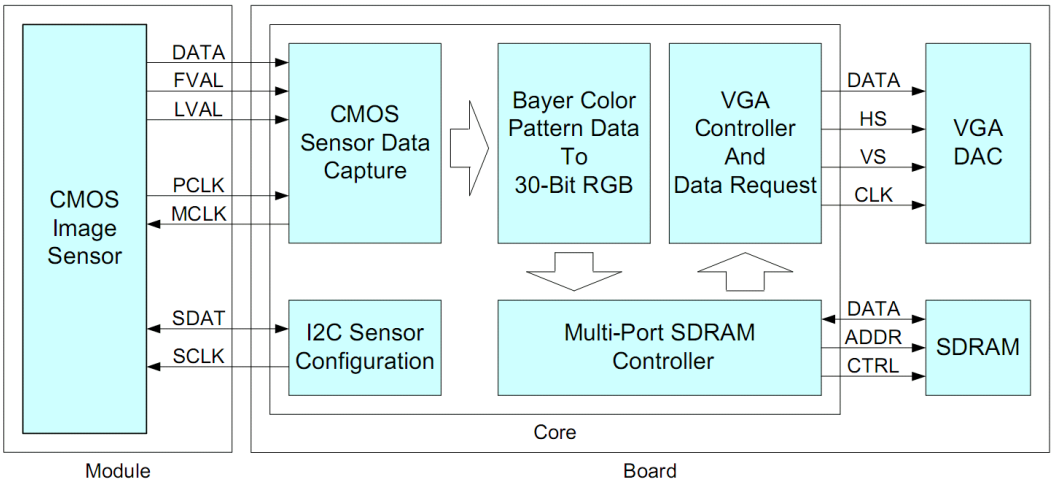

Figure 9.1: A block diagram of the Terasic camera system - from [55]

The chosen demonstrator presents two demonstrations. One is to demonstrate the difference in obtainable video "quality" by processing it in both hardware and in software. The other is to demonstrate how video can be manipulated by doing modifications to the color information and demonstrate the difference between doing this in hardware or in software. The specification of the embedded video demonstrator was based on a combination of reusing existing hardware modules and possibly modify these or designing new ones. An existing system was designed by Terasic [57] and came with the camera (see Section 6.3) and consists of several useful modules. The system is displayed in Figure 9.1 and was used as a base for putting together the final specification of the system. In order for the Terasic system to serve the wanted demonstrational purpose, it had to become much more flexible and dynamic. This flexibility was achieved by introducing a soft embedded processor used for control, configuration and video processing and manipulation.

In this chapter, the process that took place when designing the implementation of the specified embedded demonstrator is going to be presented and discussed. The system consists of several different modules, both in hardware and in software. These were described in the C software programming language and in the hardware descriptive language Verilog. Some of the software code is available in Appendix A and some of the hardware modules code can be found in Appendix B. The complete system with all software and hardware code is delivered in a .zip-file together with this thesis.

9.1 System Overview

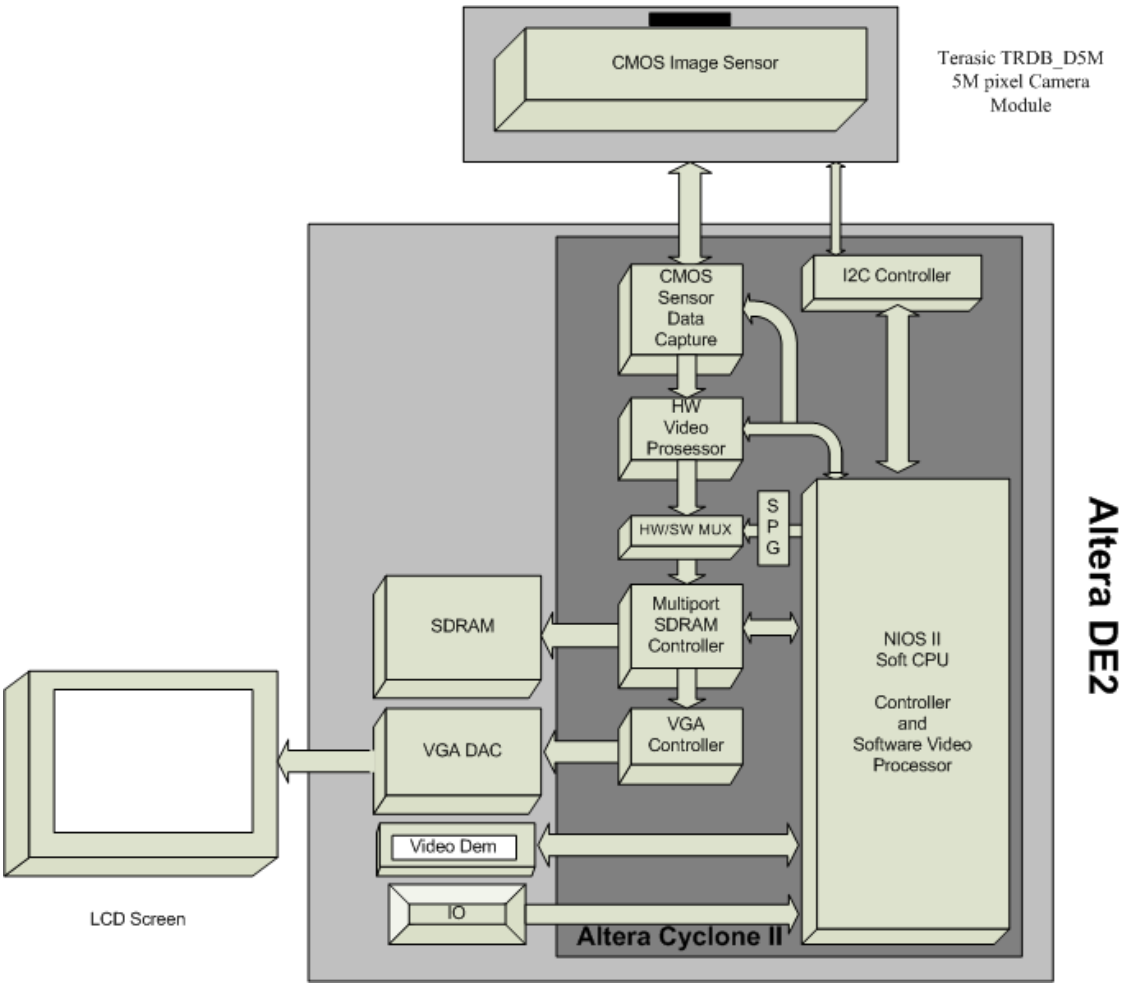

Figure 9.2: A block diagram of the system and the communication flow

Figure 9.2 gives an overview of the finalized system implemented on the FPGA and the main communication flow between the different modules. A similar block diagram was made in the specification in the project thesis. The differences are the location of the HW/SW MUX, which was originally placed above the hardware video processor, and the introduction of an extra module called SPG. The system utilizes many different hardware and software components. It uses the FPGA's on-chip memory to make shift-registers, to store software and to create FIFO buffers. The dedicated SDRAM is used as a frame buffer that holds a single frame of video at the time. The SDRAM is controlled and accessed by using the four-port controller hardware from Terasic, implemented on the FPGA, which can be used to read or write to the SDRAM through its four FIFO buffers. The external LCD screen receives analog RGB data thorough its VGA cabel. The VGA

DAC transmits this analog data by conversion of the digital data it receives from the VGA controller on the FPGA. The fastest soft Nios II processor is used at a high operating frequency of 150MHz and it is combined with different types of peripherals and custom instructions (PIO (with and without interrupt), performance counter, custom floating-point instructions, JTAG, character LCD driver and an interval timer). The I/O buttons (e.g. SW0-SW7) are used for system reset, switching between demonstration modes and for activating/deactivating other functionality. More information about how the demonstrator can be operated will be presented in Section 10 and in the poster. In the following tables, Table 9.1 and Table 9.2, the different modules in the finalized system are given a short description of their functions and origins. The tables also describes which of the modules have been modified in order to meet the spcifications. The most essential modules will be described and discussed further in the following sections.

The top module of the design, the file which describes how the different modules are connected to each other, is made by using the same top Verilog file that Terasic used for their system. An alternative would be to make a schematic top file, which can be created inside Quartus II. In the beginning of the design phase a schematic top file was used to implement only single modules, like Nios II. A schematic description of the whole system was abandoned because it would require to much time and work. It was much easier to use the existing top file and simply modify it to fit the new design. In addition, by using a code based top file it was considered much easier to add parameters and make simple changes to the system. Simple changes would be adding registers, counters or other functions in the top file itself. It was also considerably easier to keep an overview of the complete system, especially on a small screen, but this is most likely a personal preference. A part of this top file is available in Appendix B.1 and shows how the different modules are connected to each other.

Module Name	Description	Origin
CMOS Sensor Data Capture or "CCD Capture"	A module for starting or stopping the capturing of image data from the camera. It creates a DVAL signal to indicate when pixel data is valid and it creates X and Y signals that correspond to the coordinate of the valid pixel data. The pixels arrive at the rate of the pixel clock, generated from the camera. In addition, this module counts the current number of captured frames and displays it on the seven segment displays on DE2. The module was modified to be started and stopped by Nios II	Terasic
HW Video Processor or "RAW2RGB"	This module demosaics the RAW image data to viewable RGB. The module was modified to support control by Nios II and additional functionality. An important part of this module is the sub-module called "Line Buffer". This module is a MegaFunction shift-register. The shift-register functions as a line buffer so that it is possible to read from two image data lines simultaneously. This is essential for the demosaic interpolation process	Terasic
HW/SW MUX	This module was created to serve the purpose of switching between sourcing video data from hardware or software to the SDRAM frame buffer (Multiport SDRAM Controller). The MUX is controlled by the Nios II processor and sources the hardware video data as its default operation	Self
SPG or Single Pulse Generator	This module was created as a solution to a timing problem. The module takes in two signals, a clock signal and an input signal. When the input signal goes high, the SPG generates a logical high pulse on the output with a width of one clock period.	Self
SDRAM Multiport Controller or "Sdram Control 4Port"	This module is a generic 4 port SDRAM controller. It uses two FIFO buffers for inputs (writing) and two FIFO buffers for outputs. The FIFO buffers are made using Altera MegaFunctions and are 16 bit wide and can hold up to 512 words of this size. In this system the SDRAM controller is used as a frame buffer to hold a single picture frame. The SDRAM controller manages all writing, reading, sorting and synchronization.	Terasic

Table 9.1: Overview of the hardware modules on the FPGA and their origin part 1 of 2

Module Name	Description	Origin
SDRAM PLL	This is a MegaFunction from Altera and it is used to generate several clock signals based on a reference clock (50MHz). It is used to create a clock signal to the SDRAM controller and to create a similar clock to the external SDRAM. The external signal is phase shifted with $-3ns$ from the controller clock. Both clocks are running at 125MHz. The PLL have been modified by using MegaFunction tool to add an extra clock signal for the Nios II/f (150MHz) (see Section 9.1.4 and Section 9.2)	Altera
VGA Controller	This controller generates the necessary synchronization and blanking signals based on an input reference clock and a set of parameters. The parameters use the values presented in Section 6.2.2 in Table 6.1 and Table 6.2. The default parameters and reference clock puts the screen connected to the VGA DAC in 640x480x60HZ. To alter this a different reference clock must be provided and the parameters must be changed	Terasic
I^2C Controller or "I2C CCD Config"	This module is used to configure the operation of the camera. It consists of two modules: The configuration module and a sub-module for transmitting data following the I^2C standard as specified in Section 7.2. The module was modified to support receiving a configuration from the Nios II/f.	Terasic
Reset Delay	This module is not in Figure 9.2, but it is used to reset the system. When a user pushes Key[0] this will trigger a reset in this module. The module will then reset all hardware modules and wake them up, one by one with a little delay in between. This is done to make sure that the essential modules are ready before others that rely on them. The reset delay module has been modified so that Nios II can reset all hardware without being reset itself	Terasic
SEG7 LUT	This is a module for translating binary values into seven segment control signals. It is used by the CCD Capture module to display current frames	Terasic
Nios II/f embedded processor	Composed and customized by using Altera SOPC builder tool. This module will be further described and discussed in Section 9.2	Altera

Table 9.2: Overview of the hardware modules on the FPGA and their origin part 2 of 2

9.1.1 Hardware Video Signal Flow

This mode is accessed either by resetting the whole system (Key[0]) or by user selection (SW2). When this selection occurs the Nios II processor "switches" the MUX into hardware mode and it feeds the appropriate configuration to the I^2C controller. The I^2C controller requests this data by triggering an interrupt in the Nios II processor when changing its configuration data index. This configuration is described in further detail in Section 9.3. The camera is now ready and is operating as configured. Before any processing begins, the CCD Capture device has to be started. The user can start or stop this module by using SW0 or SW1. When the capture device is started, it waits for the beginning of a new image frame from the camera before it starts to count the pixels and deliver the data together with a data valid signal (DVAL) in addition to the X and Y coordinates. This data is received by the RAW2RGB module which uses a shift-register to buffer up two rows of image data. This shift-register delivers two taps of data so that the data on two separate rows can be read simultaneously and the necessary interpolation/demosaic is done to create the RGB data. This data together with a new data valid signal is then transferred through the MUX and into the two input FIFO buffers of the SDRAM multiport controller. The FIFO buffers read the input data when they get a write request. This write request is connected to the data valid signal from the video processor. The image data is then stored in the frame buffer. The VGA controller is already configured by static parameters to operate at 640x480x60Hz. This means that the controller tries to get a new frame 60 times per second. This is done by making a read request in the two output FIFO buffers and the data is read at the rate of the VGA control clock (25MHz). The digital RGB data is then converted into analog RGB signals which then can be presented on the LCD display.

9.1.2 Software Video Signal Flow

The main difference between this mode and the hardware mode is that the data is read through the Nios II processor, processed here and then delivered to the frame buffer. The software mode is accessed by user selection (SW3) which triggers an interrupt in the processor. The control signals controlling the HW/SW MUX is then changed and the data connected to the SDRAM controller is now connected to Nios II. Next, the I^2C controller receives the new configuration from the processor and transmits it to the camera. The camera now operates at a much lower frame rate, taking into consideration Nios II's limitations. As a result, when activating software mode, the processor also activates interrupt on the input called "PIX_VALCLK". This signal is gathered from the hardware video processing module (RAW2RGB) which has been modified to deliver a clock signal which only ticks as long as the data valid signal is logic "high". This means that each time the clock signal changes value to logic "1", it means that there are new image data ready to be accessed. This transaction to logic high (rising edge) triggers an interrupt that reads data from the shift-register inside the hardware module, and interpolates it to RGB data. This data is then transmitted to the RGB outputs on Nios

II, including a write signal. It is essential that this write signal just gives a single pulse of the same length as one clock period of the Nios II clock (150MHz). The reason for this is that the FIFO input buffers also use this clock, because of software mode. Since Nios II might use more then one clock cycle to pull this signal low-high-low, the SPG (Single Pulse Generator) was created. This module eliminates the possibility that the FIFO buffers should read the same pixel data more than once, which would result in corrupted pixel data. Finally the data is displayed on the screen. A successful processing of video in software.

9.1.3 System and Platform Discussion

The platform used for this system is the Altera DE2. This platform is versatile and can be used to implement a wide variety of systems to fit many different applications. Based on the embedded video system implemented here, the DE2 has all the necessary components to successfully create the application. The heart of the DE2 board is the Altera Cyclone II FPGA which has sufficient capacity and hardware performance. Since the whole idea of the FPGA is to be customizable and reconfigurable, this leaves the designer with the big task of utilizing its power by partitioning hardware and software in the most effective manner. One way to increase hardware processing performance is to use more area and do more calculations in parallel. One could argue that why not use a simple microcontroller to do the same job? As presented in [52] a microcontroller would possess the properties to do pixel color interpolation, but it would result in a bad frame rate for video purposes. In addition, with a microcontroller alone it would be difficult to demonstrate the potential of using dedicated hardware to accelerate some calculations. The microcontroller would also not be as adaptable and customizable as the FPGA is. In the design process of this system, it has been very important to be able to adapt hardware to fit a very specific need and also the possibility to create new hardware. This would not have been possible with a microcontroller.

Two of the biggest producers of FPGAs are Altera and Xilinx. Both of them have a wide range of different products, many of which are suited for different areas. Low power, large capacity and high performance are some of their properties. These producers also offer different development platforms, like the Altera DE2 or the Xilinx equivalent, the Spartan-6 LX150T development board. Both also offer a large variation of daughter boards that can be connected to the platforms, like cameras and the like. One of the main reasons for choosing Altera over Xilinx to implement the current system was the availability of the board. NTNU already had a large supply of DE2 and Altera/Terasic also supplied a sufficient camera with a functional hardware platform that was designed to fit the DE2. Xilinx also delivers a kit with a camera, FPGA development board and other components which also could prove to be a good platform for this system.

Another discussion to be had is about the use of either hard or soft processor. A presentation of these processors was given in Section 2.1. The combination of an FPGA and a hard processor, like the one Xilinx provides (Xilinx FPGA Vertex-V FXT), could provide

the system with the necessary processing power to be able to do color manipulations on the ”live” video stream. This was not the case when using Nios II/f as will be discussed in Section 9.6. The hard PowerPC 440 processor could have delivered almost the double of DMIPS compared to the Nios II/f, as shown in Table 2.3, 2.0 DMIPS/MHz versus 1.16 DMIPS/MHz. In addition would the hard processor support a much higher clock speed than the Nios II/f processor. This would also have provided additional performance. The PowerPC 440 does not implement a hardware floating-point unit, but it is possible to add such functionality by using the APU (Auxiliary Processing Unit) interface. This interface enables a direct processor connection to high-speed FPGA logic for a wide variety of coprocessing options. This could be an alternative to the custom floating-point instructions used in the implemented system. However, using this hardware could have provided additional design challenges as opposed to the easy way of implementing it using the custom instructions and the tools from Altera. The benefits of using a soft embedded processor also gave the designer wide customization possibilities which would not have been possible with the hard processor core. The Nios II/f was fitted with the exact needed peripherals and was also possible to alter to try out other solutions. This flexibility was very important in the design of the embedded demonstrator and was considered to outweigh the performance increase by using a hard processor. Further discussions of the software performance of the soft core Nios II/f will be presented in the following sections.

The specification from the project's report was based on using the Terasic hardware description. One could argue that there might be other and more efficient ways of doing the implementation, but since the designer was not experienced in implementing such platforms, it was believed that using an existing platform as a basis would be a smart way to go. It was also believed that this would increase the chances of success in implementing the demonstrator in the available time frame.

9.1.4 Programs and Tools

Programs and tools	Usage
Quartus II 9.1 SP1 Web Edition	This program and its features was used to do the hardware implementation of the system
SOPC builder tool (In Quartus II)	This tool was used for implementation of the Nios II processor, its peripherals and the custom instructions
MegaFunction tool (In Quartus II)	This tool was used for modification of the PLL and shift-register MegaFunctions
Nios II EDS 9.1	This program and its features was used to create, debug and program the software running on the Nios II processor core
ModelSim-Altera 6.5b	This simulation and debug program was used to simulate and verify hardware behavior
Matlab R2008a	This program was used to make a software simulation and verification of the color manipulation functions and the implementation of different demosaic interpolation algorithms

Table 9.3: Programs and tools used to implement the system

Table 9.3 gives an overview of the different programs used to implement this system. The main design program was Quartus II 9.1 SP1 which also gave access to the SOPC builder tool and the MegaFunction tool. All the hardware, including the Nios II/f processor was implemented using a combination of these tools. SOPC was used to implement and edit the Nios II/f processor and its peripherals. A description of these peripherals and Nios II is given in Section 9.2. SOPC was also used to add custom instructions, connect the peripherals together and defining interrupt priorities (IRQ). The MegaFunction tool was used to edit existing MegaFunctions like the "Line-buffer" (shift-register) which was used as an important part of the interpolation process, and also to edit the PLL to support additional clock signals. A clock signal could be created by using a combination of multiplier and divider options based on the reference clock signal. It was also possible to change the phase between the new clock signal and the reference clock.

Nios II EDS was used to create, debug and program the software on the Nios II/f processor. The environment lets the designer create standard C code with its library and compile the code to be executable with Nios II's instructions set. Nios II EDS structures the hardware and software into BSP projects, which includes a specialized library containing system-specific support code. It also consists of the HAL software communication layer for easy access to the underlying hardware and linking between standard C functions, like the link between `printf()` and JTAG with a console inside Nios II EDS. When the designer does alterations on the Nios II hardware, a new BSP can be generated with a simple click. And the designer can compile his code with the new hardware

configuration. One of the important discoveries made when using Nios EDS and the BSP projects was the ability to modify the BSP and HAL driver library inside the BSP editor to reduce the size of the library. This modification was essential when trying to fit the whole program into the on-chip memory (32kB memory).

9.2 Implementation of Nios II

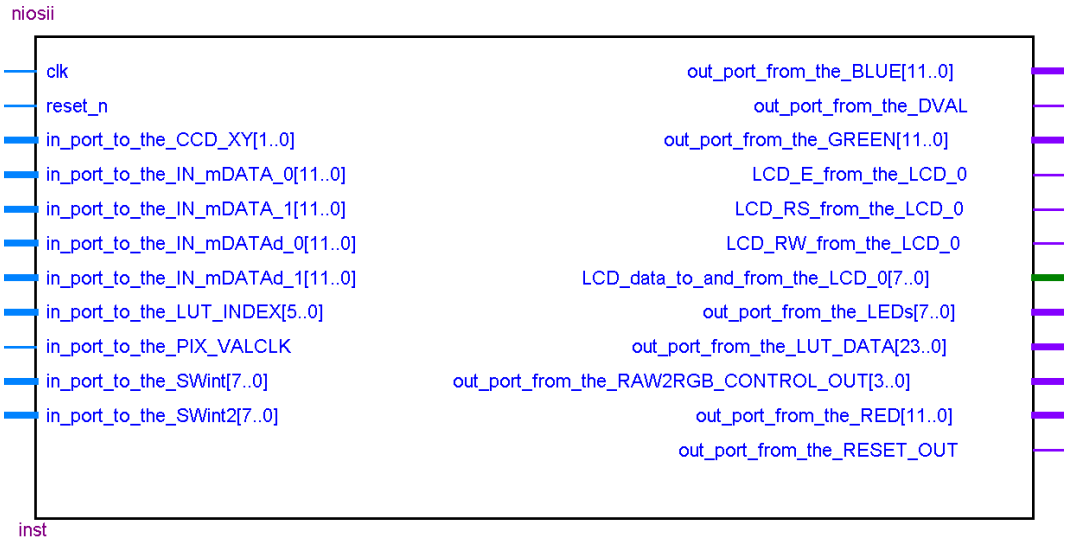

Figure 9.3: The Nios II/f-module created for this system

Nios II comes in three different core implementations, as described in Section 2.2, and was implemented using SOPC builder in Quartus II. Since video processing is a demanding task it was necessary that the fastest implementation was chosen; Nios II/f. The slowest implementation, Nios II/e, uses six clock cycles and more to execute a single instruction, but Nios II/f can execute a single instruction in one clock cycle. Table 2.3 shows a list of the difference in DMIPS performance between the core implementations. If the pixel clock was 25MHz (the default value from Terasic) and the Nios II/f was operating at a clock rate of 150MHz, this would give six clock cycles to do the potential work between each arriving pixel. In other words, not a lot of time to do calculations. This is also the reason for lowering the pixel clock significantly when processing video in software. See Section 9.3.2 and Section 9.5 for more information. In order to get the maximum performance, Nios II/f was set to run at the maximum operating frequency. This was about 150 MHz on the current FPGA (Cyclone II). An even higher clock rate could be possible by using an FPGA with a higher speed grade.

Figure 9.3 gives an overview of the implemented processor and all its inputs and outputs. Most of them are results of PIO peripheral cores (I/O) for communication with the other

hardware modules on the FPGA. Table 9.4 and Table 9.5 gives overview of the inputs and outputs and specifies where they are connected and their purpose.

Name	Usage	Module
clk, reset	Nios II clock and reset signal	SDRAM PLL 150MHz, KEY[0]
CCD XY	Pixel coordinates, use just the LSB to indicate odd or even columns and rows	CCD Capture
mDATA_0, mDATAd_0, mDATA_1, mDATAd_1	In total, all these data signals give a group of 4 pixels (2x2). Taken from the shift register and two additional delay registers	Line Buffer inside RAW2RGB
LUT DATA	The signal is used to give 24 bits of configuration data to the I^2C controller	I^2C controller
LUT INDEX	Is an interrupt driven input, on any edge. When the index value changes, this means that the I^2C configuration module needs new configuration data from the camera. This configuration data is immediately put on LUT DATA	I^2C CCD Config
DVAL	A write/valid signal to the input FIFO buffer on the SDRAM controller. This signal is meant to be a single pulse so that the FIFO just reads the current pixel once. This is done by running the signal through the SPG module	SPG->HW/SW MUX->SDRAM controller input
PIX VALCLK	This is a signal that combines the data valid signal from CCD Capture (DVAL) and the pixel clock (PIXCLK). Whenever DVAL is high, the signal will follow PIXCLK (DVAL AND PIXCLK)	RAW2RGB

Table 9.4: Nios II processor's inputs and outputs, their purpose, and to which modules they are connected to, part 1 of 2

Name	Usage	Module
SWint, SWint2	Both are 8 bits interrupt driven inputs. SWint are connected to SW0-SW7, SWint2 are connected to SW8-SW15. They trigger an interrupt on rising edge. In other words the switch has to be toggled	Switches (SW)
RED, GREEN, BLUE	The 12 bits RGB color components used to transmit processed pixel data to the frame buffer when in software video mode	SDRAM controller through HW/SW MUX
LCD E, LCD RS, LCD RW, LCD DATA	Signals connected to the onboard 16x2 character LCD from the LCD peripheral driver. The driver enables easy access to the different registers on the LCD panel. It is used for giving information about the current demonstration mode	16x2 LCD panel
LEDS	The two LSBs of this signal is used to start and stop the CCD capture device. The 6 MSBs are connected to the LEDR2-LED7	CCD Capture, LEDR
RAW2RGB CONTROL	These signals are used to control the RAW2RGB module. Now their only purpose is to activate or deactivate inverting the color data, but they can also be used to activate further hardware color processing (if implemented)	RAW2RGB
RESET	This signal is used to activate the reset of all modules except Nios II. Each time the system is reset, the I^2C controller wants new data. In other words this is the way to change the camera's behavior	RESET DELAY

Table 9.5: Nios II processor's inputs and outputs, their purpose, and to which modules they are connected to, part 2 of 2

9.2.1 Hardware Peripheral Selection

Use	Module Name	Description	Clock	Base	End	IRQ
☑	cpu_0	Nios II Processor	clk	0x00010800	0x00010fff	
☑	onchip_memory2_0	On-Chip Memory (RAM or ROM)	clk	0x00008000	0x0000ffff	
☑	jtag_uart_0	JTAG UART	clk	0x00011190	0x00011197	3
☑	timer_0	Interval Timer	clk	0x00011040	0x0001105f	10
☑	sysid	System ID Peripheral	clk	0x00011198	0x0001119f	
☑	performance_counte...	Performance Counter Unit	clk	0x00011000	0x0001103f	
☑	LCD_0	Character LCD	clk	0x00011130	0x00011131f	
☑	SWint	PIO (Parallel I/O)	clk	0x00011090	0x0001109f	4
☑	SWint2	PIO (Parallel I/O)	clk	0x000110a0	0x000110af	5
☑	LUT_INDEX	PIO (Parallel I/O)	clk	0x000110b0	0x000110bf	1
☑	PIX_VALCLK	PIO (Parallel I/O)	clk	0x00011180	0x0001118f	0
☑	CCD_XY	PIO (Parallel I/O)	clk	0x000110d0	0x000110df	2
☑	IN_mDATA_0	PIO (Parallel I/O)	clk	0x00011110	0x0001111f	
☑	IN_mDATAd_0	PIO (Parallel I/O)	clk	0x00011160	0x0001116f	
☑	IN_mDATA_1	PIO (Parallel I/O)	clk	0x00011120	0x0001112f	
☑	IN_mDATAd_1	PIO (Parallel I/O)	clk	0x00011170	0x0001117f	
☑	LUT_DATA	PIO (Parallel I/O)	clk	0x000110c0	0x000110cf	
☑	RED	PIO (Parallel I/O)	clk	0x000110e0	0x000110ef	
☑	GREEN	PIO (Parallel I/O)	clk	0x00011140	0x0001114f	
☑	BLUE	PIO (Parallel I/O)	clk	0x000110f0	0x000110ff	
☑	DVAL	PIO (Parallel I/O)	clk	0x00011150	0x0001115f	
☑	LEDs	PIO (Parallel I/O)	clk	0x00011080	0x0001108f	
☑	RESET_OUT	PIO (Parallel I/O)	clk	0x00011100	0x0001110f	
☑	RAW2RGB_CONTROL_...	PIO (Parallel I/O)	clk	0x00011060	0x0001107f	

Figure 9.4: Nios II's peripherals with memory map and interrupt

Figure 9.4 shows the final Nios II peripheral configuration. It includes several different cores. In the beginning of the design phase the Nios II was first implemented with only one simple interrupt based PIO. This was done to make it possible to test out the functionality and to learn how to get interrupt up and running. After managing the PIO core other peripherals were added and tested in the same way. It took some time to find out how to use the different peripherals and how to access them in software, but it worked out after gaining more knowledge on how to use the available commands in HAL for accessing registers and similar. The file named "`system.h`" in the BSP project contains a list of all the devices currently added to the Nios II core, and it was found that instead of using the physical base address when accessing peripherals, one should rather use the name from this file. E.g., the address for LUT_INDEX would be called LUT_INDEX_BASE equal to 0x110BO (see Figure 9.4). This makes sure that the correct device is accessed even if the memory address changes, since this would change in the file as well.

At the top of Figure 9.4 one can see the Nios II core together with the on-chip memory. This memory is 32kB and provides both instructions and data storage. The JTAG

UART core gives the necessary functions to debug the software and the ability to pause the execution and step one instruction at the time. In addition, it also connects the Nios II to the console inside Nios II EDS, which can be used to send messages from Nios II to the PC. Next is the timer core. This core was added with the purpose of being used for synchronization or to perform some kind of periodical function, but since the reading of pixel data is based on interrupt, it was not necessary to use it. It could have been removed, but it was decided to keep it in case the functionality was needed. Another important core is the performance counter. This is used to count the number of cycles a specific section of code takes to execute. It was used to gather data about how much time different sections of code took and to get a number of the performance increase by using custom floating-point instructions versus floating-point in software. The results are discussed in Section 9.6. The character LCD peripheral makes accessing and controlling the LCD panel much easier and is used to write messages about the demonstration mode and other useful information to the user. The sysid core is used to name the Nios II system with an ID. And last thre are the PIO cores. They are used for internal communication in the FPGA between the different modules, as described in Table 9.4 and 9.5. Those of the cores that are interrupt based are each given an IRQ number. This number indicates their priority. The lower the value, the higher is the priority. It was decided that the most crucial and important interrupt was the interrupt that received new pixel data when running in video mode. The second most important was decided to be the configuration of the camera, which only occurs when the hardware has just recently been restarted (not in video mode). The interrupt on the X and Y coordinates are not used, but was thought of as an alternative to the PIX_VALCLK signal to indicate new pixel data. The rest of the IRQ values are evenly distributed.

9.2.2 PIO and Interrupt

When trying to get the PIO core to function with Nios II/f, an important discovery was made. Usually, to read or write from the PIO core, one could simply make a volatile pointer to the same address as the PIO core occupies and read or write to this pointer. This was the way it was done in several of Altera's tutorials on the two other processors, Nios II/e and Nios II/s, but it did not work on Nios II/f. One of the differences between the implementations of Nios II/e and Nios II/s, and Nios II/f was the use of cache; A small, but very fast local memory, used to hold some of the recent used data. When a pointer was made to the address of the PIO core, the processor simply returned the cache value and not the current value of the core. In order to update the value in cache one had to access the PIO core's data register directly by using an I/O read command from the HAL library. It was also discovered that Nios II/f could support volatile pointers, but this would mean adding compile parameters. Since the system ultimately should use interrupt it was not necessary to use volatile pointers.

Each of the PIO cores support up to 32 bit width, this would allow one core to receive several signals which could then be separated in software and reduced the necessary

number of PIO cores. Instead many of the signals were decided to be separated into individual PIO cores with the exact bit width. This was also the case for the received pixel data when running in video mode. The reason for this choice was that the splitting of one 32 bit signal would need to use some kind of shifting mechanism, which would need extra processing time. Since the video processing was very time-critical, it was decided to use separate cores for these signals because this was found to use less time.

How interrupt works is presented in Section 2.3. Each of the PIO cores that use interrupt were given an ISR (piece of code to run when the interrupt occurs). In order for interrupt to be activated, an IRQ mask was written to the PIO core to select the bit which should trigger an interrupt. By changing this mask it was possible to disable and enable the interrupt. In the default configuration, interrupts are implemented using software to handle the requests. This process takes time and it creates a delay before the interrupt can be handled. A possible improvement to this is to add support for an interrupt vector custom instruction. This is said to possibly improve latency up to 20%, but there was not enough time to try this out. If implemented, it might improve the software video processing performance.

9.2.3 Custom Instructions

The custom instructions were implemented using the SOPC builder inside Quartus II. In the option window of the Nios II core it was possible to select between several existing custom instruction extensions or one could be created from a hardware description. One of the existing custom instructions was the floating-point instructions. This was added to serve the demonstrational purpose of comparing the time it took to process floating-point color calculation with or without the use of these instructions.

9.2.4 Software Implementation

The complete software implementation of this system is available in the .zip-file delivered together with this thesis, and the name of the BSP project is ”video_demonstrator”.
Source files: camera.c, colorManipulation.c, colorManipulation_HW.c, colorManipulation_SW.c, LCD.c, main.c, raw2rgb.c, switches.c and timer.c.
Header files: camera.h, colorManipulation.h, LCD.h, raw2rgb.h, switches.h and timer.h.
The timer-files are not used by the current implementation.

9.3 Camera Configuration and Operation

The camera from Terasic is described in Section 6.3 and can be configured to operate in many different modes. The default mode is the ERS (Electronic Rolling Shutter) mode which continuously takes pictures. This is the mode used in this demonstrator to show

video. In order for the camera to be able to work in both software and hardware, it was necessary to alter some of the configurations, like the frame rate of the video.

9.3.1 Configuration

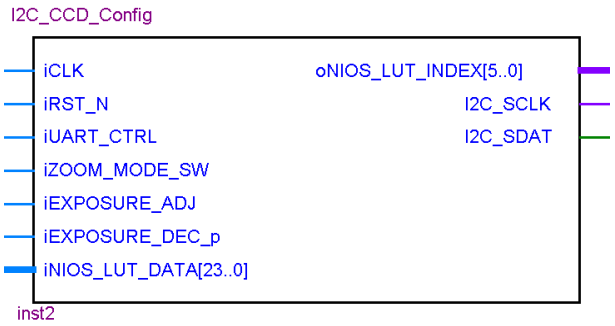

Figure 9.5: The I^2C configuration module

Figure 9.5 shows what the I^2C CCD Config module looks like. In Terasic's default configuration this module was connected to three switches: One for activating zoom, showing just some of the pixels from the sensor, and two for exposure adjustment. These switches are no longer used in this way. Instead, the Nios II processor is reading the switches and is also the one delivering the configurations to this module. When this module is reset, the first operation would be to transfer a configuration to the camera. Before the modification, this was done by counting an index through an array of configuration data stored in a static register, and sending them, one by one, using the I^2C controller's sub-module. This sub-module takes the I^2C slave address, the register address and the register data as input. Each of these packages are transferred according to the I^2C standard as described in Section 7.2. After the modification this module is no longer reading the configuration from its internal register, instead it delivers its index to the Nios II, which then triggers and interrupt, and the processor puts the corresponding data on the LUT DATA output so that the I^2C controller can read it. Because of the big difference in clock speed, there is no need for further synchronization since Nios II has the data ready before the I^2C transmission is complete.

If there had been more time, it would have been better if the processor could trigger a transmission request without resetting all the other hardware modules. One possibility would be to work directly with the internal sub-module (I^2C controller) and use it to directly transmit the data sent to it. This would be a much more elegant solution. Instead of transmitting all configurations it would be possible to just alter one register. When deciding to use this module there was also the possibility of designing one from scratch, but it was decided that the time would be better spent on other modules. Another solution could be to use another existing IP core together with the Nios II. In the Altera SOPC builder IP library there was an SPI core available, but no I^2C core was found here.

9.3.2 Operation Modes

In Table 9.6 a description of all the transferred configuration data, in default operation mode, are listed. Each of the registers have their own purpose and can change how the camera operates. Some of the most essential registers for this system are the ones that influences video image size, frame rate and exposure/shutter-width.

Size

The registers setting column and row start, column and row size and column and row mode can alter the size of the image. The start addresses are set to be where the active image is. By using Equation 6.3 in combination with the default values it is calculated that the effective image height and width is 1280x960. The demosaic algorithm implemented in hardware and software takes groups of four pixels and combines them into one full range color pixel. This gives that the effective output resolution to be 640x480 (VGA). This is the same resolution used in both hardware and software.

Frame Rate

There are several ways to adjust the frame rate, some of them are:

- Adjusting the pixel clock
- Change the shutter width (SW)
- By enabling or disabling binning (disabling could increase the frame rate, but decrease picture quality)
- Adjusting image size

In the standard configurations written by Terasic the pixel clock was set to operate at 25MHz. The pixel clock can be can be calculated by using Equation 6.4. To achieve an even higher frame rate, the pixel clock was changed to 75MHz in hardware mode. In software mode it was not possible to achieve a synchronized and clean picture at this rate. By trying to lower the pixel clock significantly, down to 150kHZ a valid picture was displayed. The pixel clock was thereby increased until the picture got corrupted again. This happened around 300kHz. At this rate the Nios II would have 500 clock cycles per received data.

Exposure

The exposure can be adjusted by altering the shutter-width. The shutter-width defines how long the image sensor is exposed and would have to be altered depending on the amount of light available.

Register name	**Address(8MSB) and Value(16LSB)**	**Description**
Read Mode	24'h20c000	Mirror rows and columns
Shutter Width Lower	24'h09(exposure-value)	Set the exposure time
Horizontal Blank	24'h050000	Extra time added to the end of each row, a higher value will decrease frame rate and increase exposure
Vertical Blank	24'h060019	Extra time added to the end of each frame, a higher value will decrease frame rate, but not alter exposure (can minumum be 8, but is 25)
Pixel Clock Control	24'h0A8000	Set to capture LVAL, FVAL and DATA on rising edge
Green1 Gain	24'h2B000b	Green1 analog gain is set to $11 \times 8 = 88$ (analog)
Blue Gain	24'h2C000f	Blue analog gain is set to $15 \times 8 = 120$ (analog)
Red Gain	24'h2D000f	Red analog gain is set to $15 \times 8 = 120$ (analog)
Green2 Gain	24'h2E000b	Green2 analog gain is set to $11 \times 8 = 88$ (analog)
PLL Control	24'h100051	Powering up the PLL
PLL Config 1	24'h111801	PLLmFactor (8'h18=24) and PLLnDivider (8'h01=1)
PLL Config 2	24'h120003	PLLp1Divider (8'h03=3)
PLL Control	24'h100053	After setting up the PLL this option switches from the input clock to begin using the PLL as the system clock
Test Pattern Control	24'hA00000	Test pattern not enabled
Test Pattern Green	24'hA10000	Value used for green pixels of dark rows and columns in test pattern
Test Pattern Red	24'hA20FFF	Value used for red pixels of dark rows and columns in test pattern
Row Start	24'h010036	Sets the start row (8'h36=54, first active image row)
Column Start	24'h020010	Sets the start column (8'h10=16, first active image column)
Row Size	24'h03077F	Sets the row size (12'h77F=1919)
Column Size	24'h0409FF	Sets the column size (12'h9FF=2559)
Row Address Mode	24'h220011	Sets the row mode: bin=2x and skip=2x
Column Address Mode	24'h230011	Sets the column mode: bin=2x and skip=2x
Row Black Target	24'h4901A8	The target black level for the row BLC alg.

Table 9.6: Camera control registers and description

9.4 Hardware/Software Selection Process

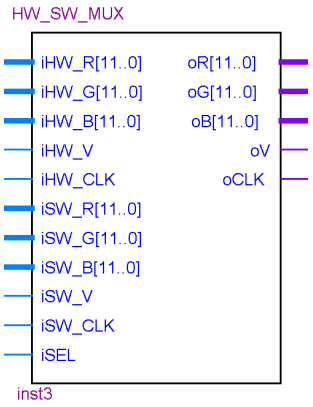

Figure 9.6: The HW/SW MUX module, used for switching between video sources

Figure 9.6 shows a block diagram of the implementation of the HW/SW MUX. This module is used to change between video processed in software and video processed in hardware. The MUX is created to function as a traditional MUX. It is connected to all the necessary signals from both the hardware module and the Nios II processor. Each of the inputs uses separate clock signals to transmit the pixel data. In hardware this is done by using the pixel clock and in software the Nios II clock is used. The input called SEL is used to switch between the two sources and is controlled by Nios II. Logic ”0”, which is default, means to source from HW input and logic ”1” means to source from SW input.

Other solutions were also considered. One of the solutions was to use the hardware RAW2RGB module to receive and transmit the software processed video. This would probably just move the location of a MUX to inside the module instead of outside. Another solution was to add extra FIFO buffers on the input of the SDRAM controller. This solution would need the possibility to start or stop the RAW2RGB module, so that video was not transmitted simultaneously from both locations. In the end it was believed that using the separate MUX to select between the sources to use the frame buffer would be the best solution. This solution is easy to control and it is believed that it gives an easier understanding of how the selection process works.

9.5 Video Processing

How this video processing works is described in detail in Section 4.3. Terasic’s implementation of this module (RAW2RGB) is still used in its original form, with just some minor modifications. These modifications are added functionality for inverting colors and

extra outputs used to transfer RAW image data to the Nios II processor. The hardware description can be found in Appendix B.2.

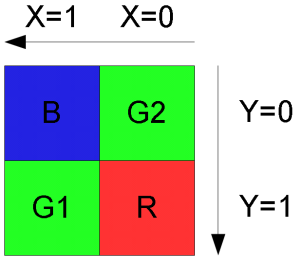

Figure 9.7: A group of four pixels in mirror readout pattern

In order to understand how this module works a form of reverse engineering was done. It started out with trying to understand how the image data was read from the camera. Since mirroring rows and columns was activated this would mean that the pixel data were arranged like in Figure 9.7.

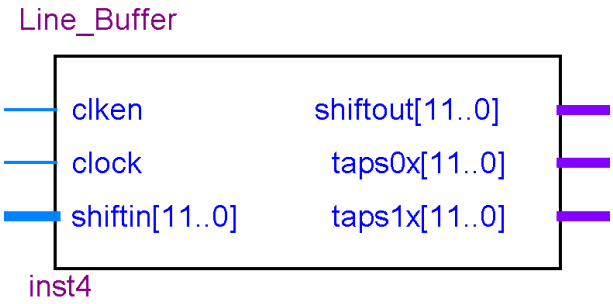

Figure 9.8: The Line-buffer used by both hardware and software for color interpolation

An important sub-module called ”Line-Buffer” was implemented inside the RAW2RGB module. This is a shift-register made with Altera’s MegaFunction tools, as seen in Figure 9.8. This shift-register has two taps from the input signal with a tap distance of 1280. Since the size of the rows, read from the camera, also was 1280, the two taps would make it possible to read from two image rows at the same time. Also, two additional delay registers were used to read a group of four pixels (2x2), like the group shown in Figure 9.7. The final approach to understand the module’s behavior was a ModelSim simulation of the whole module with 16 pixels (4x4) of image data. Each color had its own hex value. Combined with this simulation was also a Matlab software implementation of the hardware description. This gave the ability to simulate the behavior and to see how a processed image would look like. Both simulations verified the assumed behavior of the hardware module. An image from the simulation can be found in the .zip-file attached to this thesis.

9.5.1 Hardware and Software Operation and Quality

The behavior of the hardware module can be described as follows: The module samples a group of four pixels each clock cycle. Whenever the sampled group hads X and Y coordinates that correspond with both values being even, the data will be processed and transmitted as a valid pixel with a full color range (RGB). Since this is only one of four combinations of the X and Y values, the resulting resolution becomes 640x480 (VGA) and not 1280x960. The software implementation uses the same line-buffer together with the same delay registers to get the data. Without using this hardware the processor would have to buffer all this data itself. This would require much memory and would be harder to implement. It was decided that using the hardware to read data would not compromise the demonstration of video performance because the difference in performance still was significant. The software implementation of the algorithm can be seen in Appendix A.1.

Figure 9.9, on the next page, shows a comparison of a raw picture that has been interpolated in Matlab using Matlab's own demosaic (gradient-corrected linear interpolation) and the one demosaic implemented in hardware (closest neighbor algorithm). One obvious difference is the resolution, since the original image was 640x480 the interpolated result became 320x240 with the hardware based algorithm, but the linear Matlab algorithm did not leave behind any pixels. In addition, it is possible to see that, especially around sharp edges like the letters, some of the pixels have a rougher transition between the pixels when using the hardware interpolation, these transitions are less visible with the linear interpolation because it averages the closest pixel values and uses bigger groups of pixels.

When the binning mode on the camera is activated pixels are combined by taking the average of some of the surrounding pixels to represent the combined pixels. This works to the hardware interpolation's advantage and the real quality result would be similar to the one represented by using Matlab's linear interpolation. The camera can be configured to activate or deactivate binning mode, which was used to observe the quality difference. This observation showed a very small difference between the two. It might be easier to see the difference when comparing images instead of video. It also might be more obvious if the video resolution was higher and the camera was filming a surrounding containing many edges. This might provoke aliasing effects when using the nearest neighbor algorithm, as mentioned in Section 4.3 and in [52]. It might also be possible that the difference in quality would be more obvious when using higher resolutions, but this was not tested (see Section 9.5.2). Based on the observation of video quality it was concluded that the obtained image quality, when using binning, would suffice in the demonstration. The work of implementing an even more sophisticated interpolation algorithm might not be worth the effort. It was instead decided to focus more on other parts of the demonstrator.

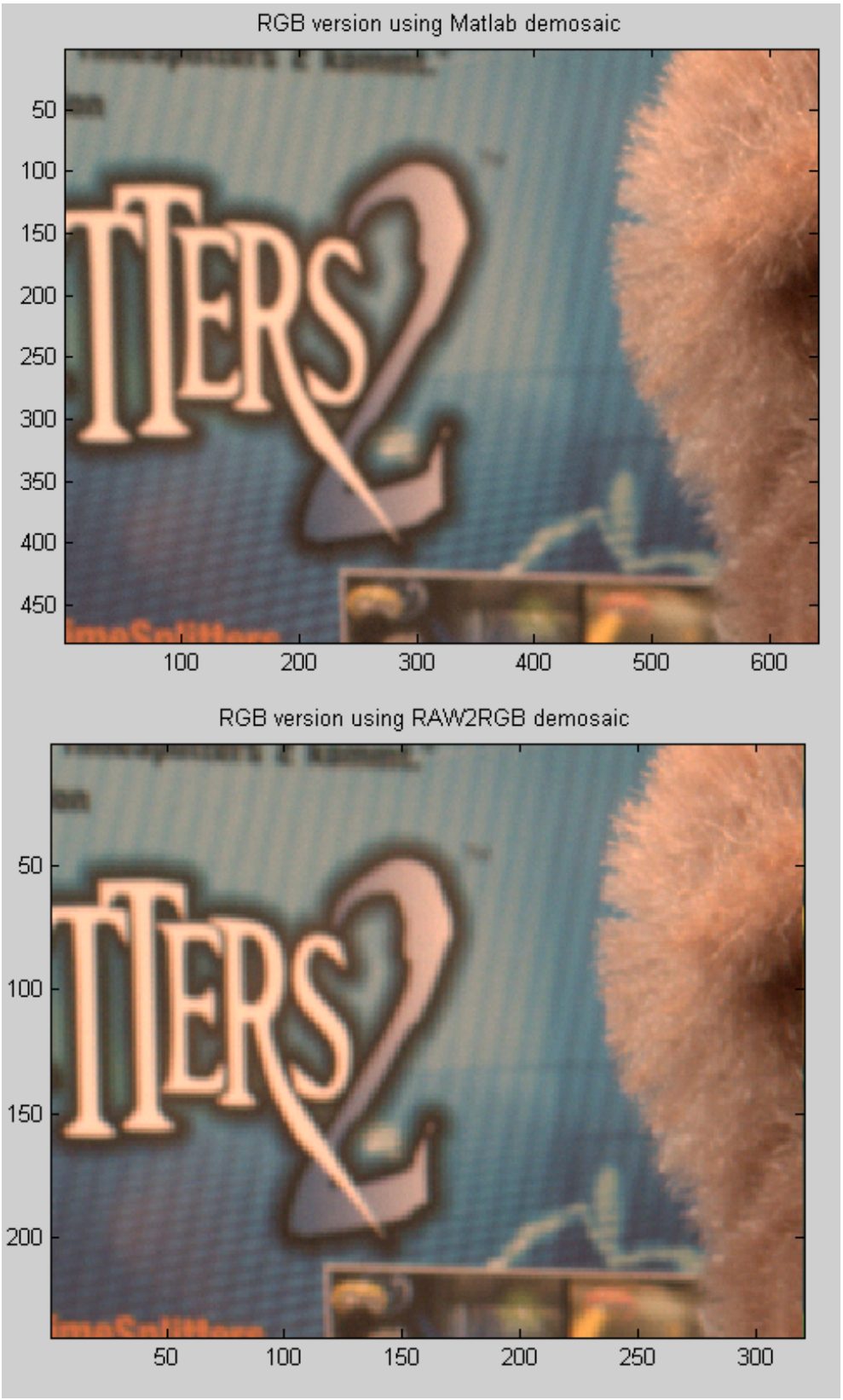

Figure 9.9: Comparison of quality of Matlab demosaic and RAW2RGB demosaic

When both the software and hardware implementation of video processing were done the main difference was the obtainable frame rate. Even though frame rate does not have anything to do with picture quality it does have an influence on the video experience. Based on this observation it was concluded that the frame rate difference would provide a better demonstration of hardware/software difference than less obvious differences in picture quality. The current system uses a pixel clock rate of 75MHz when running in hardware processing mode. At this clock rate one gets approximately 15 - 25 fps, depending on the exposure time (set by the shutter width). In contrast, when the system is running in software mode the achievable frame rate is less than 1 fps. This demonstrates a huge difference in video experience.

In order to get the video processing running in software, the simplest form of interpolation was used. It was also tried to structure the software ISR in different ways in order to get the maximum performance. One of the main reasons for not being able to achieve a better frame rate was the latency from interrupt to execution. It was later discovered an alternative solution for improving interrupt latency, custom vector interrupt. This is a way of using hardware to manage interrupts instead of software. It is said to give a 20% reduction in latency, which could increase the performance. In the process of maximizing the performance in software it was also discovered a huge difference when using optimization flags in the software compiler. This could mean the difference between being able to process video in time and not.

9.5.2 Resolutions

When the system first was specified, in the project thesis, it was proposed that the system also should be able to produce different video resolutions. In the beginning of the design phase of this system it was discovered several challenges in obtaining this system property. One of the main difficulties was the need for changing many of Terasic's hardware modules, especially the VGA controller. This module is set at a fixed resolution and both the reference clock and its static parameters would have to be made dynamic. In addition it would also require changes in both the Line-Buffer (tap distance) and the CCD Capture module (static parameters for setting the row width to create the X and Y signals). Based on the available timeframe it was decided to leave this and focus more on getting the system up and running on the default resolution. It was discovered that Altera had available IP cores for the Nios II for creating video sync signals which could be used in this context, but was not studied further. Another possibility would be to run lower resolution video inside a higher resolution window. This would mean that one a part of the screen would keep the low resolution video and the whole screen would be used when running high quality video. This option would need additional hardware and some modifications to the existing modules.

9.6 Color Processing

The processing and demonstration of different aspects of color was implemented by using a combination of software and custom instructions on the Nios II/f processor. In Section 4.2 it is presented much theory about color spaces, and different possibilities these spaces present regarding color manipulation. The initial purpose of the color processing was to transform video color data, both in software and in hardware, into YCbCr color space and do alterations to this data, before transforming them back to RGB and presenting them on the screen. In the finalized system only the color inversion function is applied to the video in both hardware and software. Color inversion is done by inverting the values of each of the color components. In hardware, this is done by inverting each component when the control signal (RAW2RGB CONTROL) from Nios II is active. In software, this function is applied by subtracting the maximum value of the 12 bits color data ($2^{12} - 1 = 4096 - 1 = 4095$) with the current color value. The color inversion fuction does not need to convert the data between the color spaces (RGB and YCbCr) in order to do the manipulation.

After getting the software video processing up and running, it was evident that the obtainable performance did not leave room for much processing between each arriving pixel. When trying to do a color space transformation this resulted in corrupted video. Based on this observation it was decided to look at other ways to demonstrate color processing. Since much of the color transformation already had been implemented in software, it was decided that the system could do the processing on a test image, which was created for this purpose. The test image is a color palette of three horizontal bars with the three RGB colors. This test image was exposed to different kinds of color processing using the software processor. In order to present a demonstration of color processing in both hardware and software, it was decided to use the custom floating-point instructions available in SOPC builder to the Nios II core. Since the color space coefficients are based on decimal values, floating-point was one option that would provide the necessary demonstration. As a result it is now possible to do the color processing in both hardware and software and to visually observe the performance increase when using the custom instructions. An alternative to processing the colors using floating-point would be to use fixed point. This would increase the performance significantly. However, this would not benefit the intended demonstration, but it could have been implemented in order to try to speed up the process enough so that it would be possible to process color data from video. Another good thing about using custom instructions in the demonstrator is the possibility to demonstrate this type of hardware/software codesign - accelerating tasks by use of custom instructions.

An alternative to the test image for color processing could be to take a still picture using the camera and process this instead. This solution would have to change the read mode of the camera to snapshot-mode and trigger a picture to be taken. Since a picture taken at 640x480 has 3 data per pixel times 8 bits (at least) each, this would mean a total of 7,372800Mbit or 921,6 kByte. This is much more than the available on-chip memory,

which means that the data would have to be put on either SDRAM or on FLASH. This was not possible to do in the available time, but could be worth further research. Grayscale color mode, as discussed in Section 4.2, was not implemented, but it would not require much work to add it to the existing color functions. This would not hve improved the demonstration of HW/SW processing, but it would be an extra functionality.

In order to implement some kind of color space transformation of video in hardware, it would be necessary to research how to implement decimal numbers using fixed point integer values. Based on the available time and the fact that color transformation of video in software did not work, it was decided to focus more on the custom floating-point instruction solution. Towards the end of the design phase it was discovered an important resource of information and tools from Altera which could have made this process much easier: Video and Image Processing Suite using MegaFunctions [23]. Among these resources it was found a color space converter module. This module would be possible to implement as an addition to the current system.

Performance

Transition	Instruction Type	Clock Cycles
RGB → YCbCr	SW	6129
YCbCr → RGB	SW	6772
RGB → YCbCr → RGB	SW	12646
RGB → YCbCr	HW	2954
YCbCr → RGB	HW	2233
RGB → YCbCr → RGB	HW	4786

Table 9.7: The performance of calculating the transition between color spaces with and without the use of custom floating-point instructions

Table 9.7 shows the number of clock cycles that was needed to perform the floating-point calculation for each color space transition and the total cycles from RGB back to RGB. The results indicate a factor of two to three in performance gain if using the custom floating-point instructions. Each of the transitions where measured using the performance counter peripheral in Nios II, and were run ten times each to get an average clock cycle usage. The functions that where used in this measurement were the same as the ones that are being used in the demonstration. By using the performance counter it was found a huge performance difference between using and not using compiler optimizations. The results in Table 9.7 are based on not using compiler optimization. This was done with the purpose of being sure that all the tests were measured equally. When measuring, it was also learned the importance of not using commands like `printf()` which rely on I/O and UART transmissions. This will give irrational results since they relay on devices that are slow and can vary much in consumed time.

The transition from RGB to YCbCr needs nine floating-point multiplications and nine

plus three floating-point additions (see Appendix A.2), while the transition from YCbCr to RGB needs, nine floating-point multiplications and additions, plus three floating-point subtractions, (see Appendix A.2). In addition, this transition might need an additional three integer additions for rounding up if values are below zero, so that the color values do not receive negative numbers.

When testing a single floating-point multiplication, this gave 308 cycles in SW and 34 cycles in HW, which is about nine times faster. This is a much higher performance increase and is more similar to the ones that Altera achieve with their custom floating-point test as presented in Table 6.4. Here they have about 12 - 17 times performance increase on custom floating-point multiplication. The reasons for not being able to achieve the same result can be many. One reason is that the FPGAs Altera use are newer and possibly faster than the Cyclone II used in this system. Compared to the results from Table 9.7 it might be possible to achieve even better results if the algorithm is further optimized. The custom floating-point instructions used support only single precision in hardware. To achieve single precision, an ”f” was added to the color space coefficients, but it might be that this has to be further specified inside the algorithm.

Chapter 10

Demonstration

The embedded video demonstrator implemented in this master thesis is most likely to be used on occasions when Department of Electronics and Telecommunication is making an appearance to show people what they do. Such occasions could be for example when schools visit NTNU, Forskningstorget and Elektronikk- & telekommunikasjonsdagen. The people that appear on these days are most likely to have different age and background. From the pedagogical perspective presented in Section 8.2 it would be advantageous if the demonstration is performed with regard to this. The main objectives of this demonstrator can be summarized in three words:

- Motivate
- Demonstrate
- Educate

At first glance, the embedded video demonstrator seems like a simple system that "just" displays video recorded from a connected camera, but after a presentation of the system, see Section 10.1, the observer gets a better understanding of what the demonstrator actually represents: A system to learn about the importance of hardware/software codesign. The embedded video demonstrator's *motivating* factor is its medium, video. Video is common and much used, and is often related to entertainment and communication. In order to *demonstrate* the importance of HW/SW codesign, a combination of visual demonstration and theoretical *education* is used. This practical approach is recommended by pedagogical theory and experiences from others that teach about this field, see Section 8.

The next sections will describe a possible demonstration of the system. The demonstration is divided into three different stages. First, there will be performed a presentation of the system giving the observer knowledge of its purpose. In addition some necessary theory will be described. Second, the interaction part of the demonstration. Here the observer is allowed to "play" with the system and go through the different modes. The

third and final stage is a discussion of the results.

10.1 The Presentation

This is the first stage of the demonstration and it begins when a possible candidate appears. This person could be interested in knowing more about what the department does or it might be that he or she is interested to learn more about the demonstrator. Either way it should be given a quick presentation of the department and some examples of all the exciting things that are possible with electronics. This should then be followed by a presentation of this system. To benefit from the pedagogical theory presented in Section 8, the presentation should include the following:

- What the purpose of this demonstrator is and emphasizing the important part (HW/SW codesign)
- Motivation by using examples of systems that would not been possible without the combination of hardware and software (e.g. PCs, mobile phones etc.)
- A suitable presentation of the demonstrator based on the age and background of the observer. If the observer is very young he or she would not benefit from a very theoretical approach. In this case the demonstrator could perform as an exciting and playful demonstration of video. This would require the one demonstrating to take an even bigger part in the interaction stage.
- Enough theory to understand how the system works and be able to understand the results, but keep it as simple as possible
- Using the poster as a tool to illustrate some of the theory presented

Presentation Material - Poster

A poster has been made as a supplement to the demonstration. The poster is divided into theory, demonstration and conclusion. Each part can be used to support all the demonstration stages. The theory gives illustrations and the necessary information to understand the main features of the demonstrator. This includes theory about what an embedded system is, the transaction from RAW image data to RBG, color planes, floating point data and the difference between dedicated hardware and software. The demonstration section of the poster gives an overview of the system and it describes how to access the different operation modes. The information should be sufficient to let the observer go through all modes by following the instructions given by the poster. At the end, when the interaction is finished, the conclusion of the poster can be used while discussing the results.

10.2 The Interaction

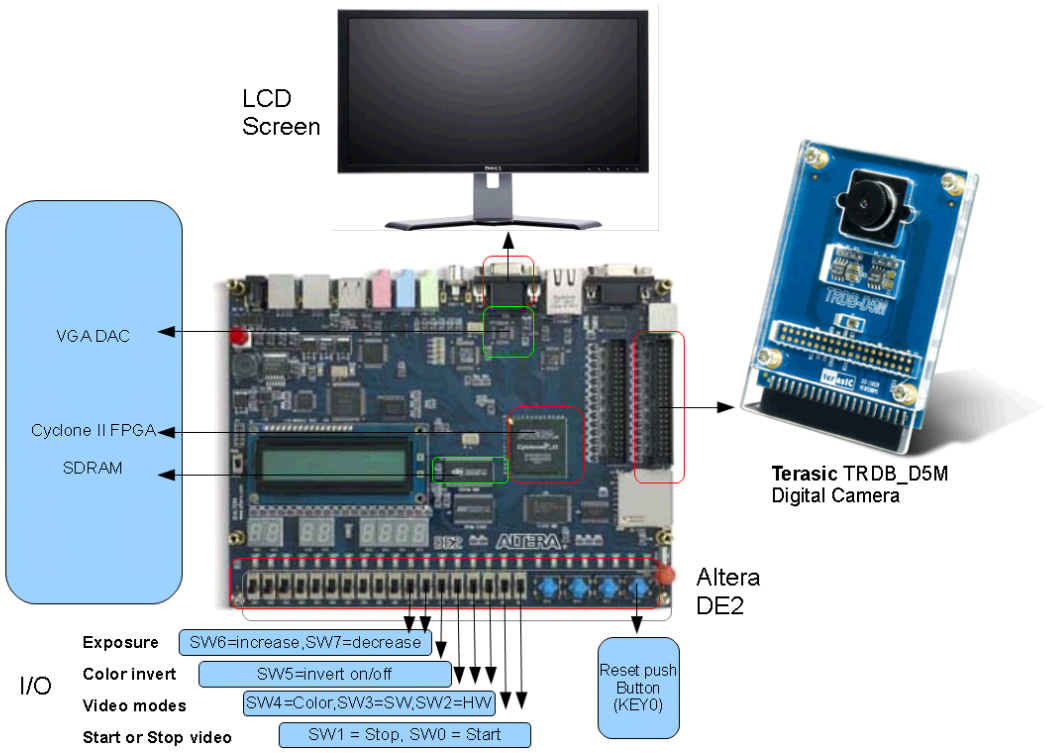

Figure 10.1: An overview of the demonstrator and its I/O

In this part of the demonstration it is time for the observer to interact with the demonstrator system. The one demonstrating should observe and give any needed assistance and possibly elaborate if anything is unclear. Figure 10.1 shows an overview of all the different components used by the system. This figure is also included in the poster and shows how the different switches can be used. The overview in the poster is followed by a step-by-step instruction of how to operate the demonstrator and how to access all the different modes. It should be possible to operate the system without the need for any help. In addition, the system uses the LEDs and the LCD to tell the observer which mode it is in and which of the buttons that are available in the current mode. From the run-through the participant should be able to come to a conclusion about the importance of a combination of hardware and software.

Switch	Purpose
Key[0]	To reset all modules
SW0	To start capturing video data from the camera
SW1	To stop capturing video data from the camera
SW2	Selecting hardware video processing (default)
SW3	Selecting software video processing
SW4	Selecting color demonstration mode
SW5	Inverting colors on/off by toggling
SW6	Increase exposure
SW7	Decrease exposure

Table 10.1: Switches and their purpose. Each switch needs to be "toggled" in order to do its function, except KEY[0]

Mode	Purpose	Active Switches
Video processing in hardware	To transform video data from RAW to RGB and interpolate missing data	SW0,SW1,SW5,SW6,SW7
Video processing in software	To transform video data from RAW to RGB and interpolate missing data	SW0,SW1,SW5,SW6,SW7
Color demonstration	Shows a color image and performs color transformations in both HW and SW	SW4

Table 10.2: Demonstration modes, their purpose and active switches

Table 10.1 lists all the available switches and what they do, and Table 10.2 gives an overview of the purpose of the different modes and the switches that are available for use in the selected mode.

10.3 The Discussion

Mode	Conclusion
Video processing in hardware	The resulting video is very viewable with a relative high frame rate
Video processing in software	The resulting video is not very viewable with a frame rate of less than 1 fps
Color demonstration	Transformation between the color spaces by using floating point requires much calculation. By using a hardware floating point multiplication this process is much faster than in software

Table 10.3: The conclusion of the different demonstration modes

This is the final stage of the demonstration and the participant should now be able to come to a conclusion about the purpose of the system, as presented in the first stage. It is important that the one in charge of the demonstration takes part in this discussion and possibly clear up any misunderstandings, and hopefully come to a common conclusion. A common conclusion would be similar to the one described in Table 10.3 and the main outcome should be to understand that hardware/software codesign is imperative in the designer's job to accomplish a system that performs within specifications.

10.4 Target Group

The embedded video demonstrator can be used for demonstration on many different occasions. On these occasions there are visitors with very different age and background. The pedagogical theory presented in Section 8 says that the education has to be fitted based on factors like age. This would suggest that it is not possible to presume that children from primary school would have the same benefit of the demonstration as high school pupils or students. With this in mind it is presented in the following table (Table 10.4) a suggestion of what the different participants can learn from the demonstration and how to present this by using the demonstrator.

Age Group	**Knowledge gained from demonstration**	**How to perform demonstration**
Primary school or elementary school (age 6-12)	It should be possible to learn that electronics is fun and that it can be used to show recorded video, andto change the colors of the video	The one demonstrating should put the demonstrator in hardware video processing mode and let the children play in front of the camera. In addition it might be fun to let the children see themselves with the colors inverted. Another possibility is to run the system in software processing mode and let the children move in front of the camera. This will result in a slow video showing a fun, deformed version of themselves because of the movement
Junior high school (age 13-16)	It should be possible to learn the difference between a software processor (PC) and dedicated hardware (graphics card), and that dedicated hardware can be used to get much better performance (better looking games and video)	The one demonstrating should present the system in a way so that the participant can relate to the difference between hardware and a software processor. With this in mind it should be possible for the observer to switch between hardware and software processing of video and come to a conclusion about the difference. Inverting of colors or adjusting exposure could also be done to show some possibilities of electronics
High school or students (age 16+)	It should be possible to learn about the difference between a software processor and dedicated hardware. It should also be possible to learn about how a camera works and how colors can be presented and processed. The participant should also be able to learn what an FPGA is and that it can be used to run both software and hardware.	This age group is this demonstrator's main target. It should be possible to perform the whole demonstration using the poster. The one demonstrating should start off by giving a quick introduction about the system using the poster to show some of the theory. Next, it should be possible to let the participant operate the system, following the instructions on the poster combined with assistance if needed. The one demonstrating should also elaborate the different demonstration modes. Finally, the participants should be able to come to a conclusion, which should be discussed with the one demonstrating for best educational effect

Table 10.4: Target Groups

Chapter 11

Conclusions

In this master thesis there has been implemented an embedded demonstrator for video presentation and manipulation. The system was first specified as part of the project thesis written last semester and was chosen for its motivational and educational properties. It has also been created a poster that can be used as a supplement to the demonstration together with a plan of how the system should be demonstrated based on factors like age and background of the observers.

The specified demonstrator is based on a functional hardware description that came with the Terasic camera. This hardware was used as a base in the design and is still an important part of the final system. In order to give the Terasic hardware system its demonstrational properties, a Nios II/f embedded soft processor from Altera was implemented. This processor's purpose is to serve as an additional processing element and to control the behavior of the demonstrator. The processor uses additional custom floating-point instructions to provide demonstrational properties when processing image color data.

In the designer's opinion, the final implementation of the embedded video demonstrator was a success. The final demonstrator is capable of processing video recorded from the camera in both hardware and software. The two processing alternatives give a performance difference of 25 fps (maximum) in hardware versus less than 1 fps in software. The image quality of the video in hardware and software is equal, but it is concluded that the difference in frame rate gives a greater demonstrational effect than the marginal difference in image quality does. In addition, there has been implemented the possibility to invert colors in both hardware and software processing mode. It is concluded that this effect on the colors can be used to give additional meaning to the demonstration, especially for observers from primary school/elementary school. The final implementation also gives a demonstration of the deference between processing color data using either software emulation of floating-point or by using dedicated custom floating-point instructions. The result shows an obtainable performance increase of a factor of three, if using custom floating-point instructions. Based on observations of this difference visually

on a screen, it is concluded that it is possible to observe the difference in performance between these two processing modes. It is also concluded that the created poster should be sufficient as a supplement to the demonstration together with the plan of how the demonstration should be performed.

11.1 Future Work

One way of improving the demonstrator further could be to implement the possibility to run a higher resolution when processing video in hardware. This might increase the effect of the demonstration. Implementation of custom instruction interrupt vectors could increase the performance of the software video processing. Since the frame rate is as low as one frame per second or less, it could be advantageous to improve the performance a little.

To improve the color demonstration it might be possible to increase the performance difference by doing further optimizations on the color space converting algorithms. Optionally it could also be possible to use real images captured by the camera when processing the colors.

It would also be recommended to perform several demonstrations for people with different age and background to get feedback on how the demonstrator is received. This would be the ultimate test to determine whether the demonstrator was a success or not.

Bibliography

[1] Atmel Homepage. http://www.atmel.com/.

[2] A Resource for Documentation About The PowerPC 440. https://www-01.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_440_Embedded_Core.

[3] A Resource for Documentation About The Xilinx MicroBlaze Soft Embedded Processor. http://www.xilinx.com/tools/microblaze.htm.

[4] A Resource for Documentation About The Xilinx Vertex 5 FXT. http://www.xilinx.com/products/virtex5/fxt.htm.

[5] A Resource for Information About Development Kits from Altera. http://model.com/.

[6] A Resource for Information About Development Kits from Xilinx. http://www.xilinx.com/products/boards_kits/index.htm.

[7] A Resource for Information About Modelsim. http://model.com/.

[8] A Resource for Information About Synopsys and Synplify Pro. http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx.

[9] A resource for IP cores optimized for Altera devices, from Altera and third-parties. http://www.altera.com/products/ip/ipm-index.html.

[10] Altera Corporation. *Nios Embedded Processor.* http://www.altera.com/products/ip/processors/nios/nio-index.html Downloaded December 3. 2009.

[11] Altera Corporation. *Introduction to Megafunction*, January 1998.

[12] Altera Corporation. *DE2 Development and Education Board User Manual, Version 1.4*, 2006.

[13] Altera Corporation. *Cyclone II Device Handbook, Volume 1*, February 2007. http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf.

[14] Altera Corporation. *Shift Register (RAM-Based) Megafunction User Guide*, July 2008.

[15] Altera Corporation. *Altera Product Catalog*, 7.1 ed., 2009. `http://www.altera.com/literature/sg/product-catalog.pdf`.

[16] Altera Corporation. *Embedded Design Handbook*, July 2009. `http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf`.

[17] Altera Corporation. *Introduction to the Quartus II Software, Version 9.1*, 2009. `www.altera.com`.

[18] Altera Corporation. *Nios II Performance Benchmarks*, June 2009. `http://www.altera.com/literature/ds/ds_nios2_perf.pdf`.

[19] Altera Corporation. *Nios II Processor Reference Handbook*, March 2009. `http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf`.

[20] Altera Corporation. *Nios II Software Developer Handbook*, March 2009. `http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf`.

[21] Altera Corporation. *Quartus II Handbook Version 9.0 Volume 5: Embedded Peripherals*, March 2009. `www.altera.com`.

[22] Altera Corporation. *Quartus II Handbook Version 9.1 Volume 4: SOPC Builder*, November 2009. `www.altera.com`.

[23] Altera Corporation. *Video and Image Processing Suite: User Guide*, November 2009. `www.altera.com`.

[24] Altera Corporation. *Using Nios II Floating-Point Custom Instructions Tutorial*, February 2010. `www.altera.com`.

[25] Altera Homepage. `http://www.altera.com/`.

[26] Analog Devices Inc. *CMOS, 240MHz Triple 10-bit High Speed Video DAC - ADV7123*, rev. A ed. `www.analog.com` Downloaded September 2009.

[27] Bertels, P., D'Haene, M., Degryse, T., and Stroobandt, D. Teaching skills and concepts for embedded systems design. In *ACM SIGBED Review Volum 6 Issue 1 Article No.:4* (New York, USA, January 2009).

[28] Biswas, P., Banerjee, S., Dutt, N. D., Pozzi, L., and Ienne, P. ISEGEN: An Iterative Improvement-Based ISE Generation Technique for Fast Customization of Processors. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 14, No. 7* (July 2006).

[29] Burns, A., and Wellings, A. *Real-Time Systems and Programming Languages*, third ed. Pearson, York, England, 2001. ISBN: 978-0-201-72988-7.

[30] Catsoulis, J. *Designing Embedded Hardware*, second ed. O'Reilly Media, Inc., Sebastopol, USA, 2005.

[31] Cibson, J. D., Berger, T., Lookabaugh, T., Lindbergh, D., and Baker, R. L. *Digital Compression for Multimedia: Principles and Standards.* Elsevier, San Francisco, 1998. ISBN: 1-55860-369-7.

[32] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing*, third ed. Pearson, New Jersey, USA, 2008. ISBN: 0-13-168728-x.

[33] Hall, T. S., Bruckner, J., and Halterman, R. L. A Novel Approach to an Embedded Systems Curriculum. In *36th ASEE/IEEE Frontiers in Education Conference* (San Diego, USA, October 28-31 2006).

[34] H.Fletcher, B. FPGA Embedded Processor Revealing True System Performance. In *Embedded Training Program Embedded Systems Conference* (San Francisco, USA, 2005), Memec.

[35] IBM. *The PowerPC 440 Core*, September 2009. `https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF778525699700634 31C/$file/440_wp.pdf`.

[36] IEEE Computer Society. *IEEE Standard for Binary Floating-Point Arithmetic, IEEE Std 754*, 1985.

[37] Imsen, G. *Elevenes Verden - Innføring i Pedagogisk Psykologi*, fourth ed. Universitetsforlaget, Oslo, NO, 2005. ISBN-10: 82-15-00737-6.

[38] Imsen, G. *Lærerens Verden - Innføring i Generell Didaktikk*, third ed. Universitetsforlaget, Oslo, NO, 2006. ISBN: 978-82-15-00874-5.

[39] Jack, K. *Video Demystified: A Handbook for the Digital Engineer.* Elsevier, Burlington, USA, 2005. ISBN: 0-7506-7822-4.

[40] Kjeldsberg, P. G., Hartmann, M., Gajski, D. D., Mano, M. M., and Kime, C. R. *Digital Design and Computer Fundamentals.* Pearson Custom Publ., 2005. ISBN: 1846582822.

[41] Lam, S.-H., and Kok, C.-W. Demosaic : Color Filter Array Interpolation for Digital Cameras. In *Second IEEE Pacific-Rim Conference on Multimedia (IEEE-PCM)* (Beijing, KINA, October 2001), pp. 1084–1089.

[42] Lilja, D. J. *Measureing Computer Performance : A Practitioners guide*, first ed. Cambridge University Press, Cambridge, USA, 2000. ISBN: 0-521-64105-5.

[43] Longbottom, R. *Dhrystone Benchmark Results on PCs*, November 2009. `http://www.roylongbottom.org.uk/dhrystone%20results.htm` Downloaded December 3. 2009.

[44] Loo, S. M., and Planting, C. A. Use of Discrete and Soft Processors in Introductory Microprocessors and Embedded Systems Curriculum. In *ACM SIGBED Review Volum 6 Issue 1 Article No.:5* (New York, USA, January 2009).

[45] Lysecky, R., and Vahid, F. A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning. In *Design Automation and Testin Europe (DATE) Volume01* (Munich, Germany, 2005), pp. 18–23.

[46] Marwedel, P. *Embedded System Design.* Springer, Dordrecht, The Netherlands, 2006. ISBN: 9780387300870.

[47] Mentor Graphics. *ModelSim Tutorial*, May 2008.

[48] Micheli, G. D., and Gupta, R. K. Hardware/Software Co-Design. In *Proceedings of the IEEE, Vol. 85, NO.3* (March 1997).

[49] Nakamura, J. *Image Sensors and Signal Processing for Digital Still Cameras*, first ed. CRC Press, Boca Raton, USA, 2006. ISBN: 0-8493-3545-0.

[50] Opencores.org - a resource for development and publications of free (LGPL) hardware core designs. `http://www.opencores.org`.

[51] Rose, J. Hard vs. Soft: The Central Question of Pre-Fabricated Silicon. In *Proceedings of the 34th International Symposium on Multiple-Valued Logic(ISMVL04)* (Toronto, USA, 2004), ISMVL.

[52] Sakamoto, T., Nakanishi, C., and Hase, T. Software Pixel Interpolation for Digital Still Cameras Suitable for a 32-bit MCU. In *IEEE Transactions on Consumer Electronics, Vol. 44, No.4* (November 1998).

[53] Shoup, R. Superpaint: An Early Frame Buffer Graphics System. In *IEEE Annals of the History of Computing* (April-June 2001).

[54] Smith, K. A., Sheppard, S. D., Johnson, D. W., and Johnson, R. T. Pedagogies of engagement: Classroom-based practices. In *Journal of Engineering Education* (January 2005), pp. 87–101.

[55] Terasic. *TRDB_D5M 5 Mega Pixel Digital Camera Development Kit*, version 1.0 ed., March 2008. `http://www.terasic.com.tw`.

[56] Terasic. *TRDB-D5M Hardware Specification*, version 0.2 ed., June 2009. `http://www.terasic.com.tw`.

[57] Terasic Homepage. `http://www.terasic.com.tw/en/`.

[58] Weiss, A. R. *Dhrystone Benchmark: History, Analysis, "Scores" and Recommendations*, November 2002. `http://www.johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf` Downloaded December 3. 2009.

[59] Wolf, W. H. Hardware-Software Co-Design of Embedded Systems. In *Proceedings of the IEEE, Vol. 82, No.7* (July 1994), pp. 967–989.

[60] Wolf, W. H. A Decade of HW/SW Codesign. In *IEEE Computer, Vol. 36, No.4* (April 2003), pp. 38–43.

[61] Wolf, W. H. *Computers as Components : Principles of Embedded Computing System Design, Chapter 7 Hardware Accelerators.* Elsevier, San Francisco, USA, 2005. ISBN: 0-12-369459-0.

[62] Xilinx Homepage. `http://www.xilinx.com/`.

[63] Yiannacouras, P., Steffan, J. G., and Rose, J. Application-Specific Customization of Soft Processor Microarchitecture. In *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays* (New York, USA, 2006), ACM.

Appendix A

Software Code

```
1  // This is from "raw2rgb.c" and is the demosaic
2  //interpolation algorithm used software
3
4  //This demosaic alg. does just use the neighboring pixels
5  //to calculate the missing colors.
6  void demosaic(unsigned int mDATA_0,unsigned int mDATAd_0,
7  unsigned int mDATA_1,unsigned int mDATAd_1,char xy)
8  {
9
10   //If the coordinate is (0,0) then calculate the
11   //other color data for the pixel.
12   if(xy == 0x0) // x = 0, y = 0
13   {
14     data.R = mDATA_1;
15     data.G = mDATAd_1;
16     data.B = mDATAd_0;
17     if(invert_flag) transmit_RGB(colorInversion_HW(data));
18     else transmit_RGB(data);
19   }
20
21 }
```

Listing A.1: Raw to RGB demosaic software algorithm

```
1  // This file is a part of the colorManipulation - files
2  // including the two functions for trasforming from
3  // RGB to YCbCr and YCbCr to RGB
4
5
6  YCBCR RGBtoYCbCr_HW(RGB data)
7  {
8    YCBCR result;
9    float tmp_result[3];
10   unsigned int RGB[3] = {(data.R),(data.G),(data.B)};
11   int n = 0;
12   int m = 0;
13
14   //Matrix multiplication
15   for(;n<3;n++)
16   {
17     tmp_result[n] = 0;
18     m = 0;
19     for(;m<3;m++)
20     {
21       tmp_result[n] += rbgTOycbcr_factors[n][m] * RGB[m];
22     }
23   }
24
25   result.Y = 16 + tmp_result[0];
26   result.CB = 128 + tmp_result[1];
27   result.CR = 128 + tmp_result[2];
28
29   return result;
30
31 }
32
33 RGB YCbCrtoRGB_HW(YCBCR data)
34 {
35   float tmp_result[3];
36   RGB result;
37   int n = 0;
38   int m = 0;
39   float YCBCR[3] = {data.Y - 16,data.CB - 128,data.CR - 128};
40
41   //Matrix multiplication
42   for(;n<3;n++)
43   {
```

```
44            tmp_result[n] = 0;
45            m = 0;
46            for(;m<3;m++)
47        {
48          tmp_result[n] += ycbcrTOrgb_factors[n][m] * YCBCR[m];
49        }
50      }
51
52      //Rounds up if the value is less than 0
53      if(tmp_result[0]<0) result.R = 0;
54      else result.R = (int) tmp_result[0]+1;
55
56      if(tmp_result[1]<0) result.G = 0;
57      else result.G = (int) tmp_result[1]+1;
58
59      if(tmp_result[2]<0) result.B = 0;
60      else result.B = (int) tmp_result[2]+1;
61
62      return result;
63
64  }
```

Listing A.2: Color Functions

Appendix B

Hardware Code

```
1
2  // This file is a part of the top file of the system '"DE2_D5m.v"
3
4  ////////////////////////////////////////////////////
5  //=================================================
6  // REG/WIRE declarations
7  //=================================================
8
9  //      CCD
10 wire     [11:0]  CCD_DATA;
11 wire             CCD_SDAT;
12 wire             CCD_SCLK;
13 wire             CCD_FLASH;
14 wire             CCD_FVAL;
15 wire             CCD_LVAL;
16 wire             CCD_PIXCLK;
17 wire             CCD_MCLK; //CCD Master Clock
18
19 wire     [15:0]  Read_DATA1;
20 wire     [15:0]  Read_DATA2;
21 wire             VGA_CTRL_CLK;
22 wire     [11:0]  mCCD_DATA;
23 wire             mCCD_DVAL;
24 wire             mCCD_DVAL_d;
25 wire     [15:0]  X_Cont;
26 wire     [15:0]  Y_Cont;
27 wire     [9:0]   X_ADDR;
28 wire     [31:0]  Frame_Cont;
29 wire             DLY_RST_0;
```

```
30  wire              DLY_RST_1;
31  wire              DLY_RST_2;
32  wire              Read;
33  reg       [11:0]  rCCD_DATA;
34  reg               rCCD_LVAL;
35  reg               rCCD_FVAL;
36  wire      [11:0]  sCCD_R;
37  wire      [11:0]  sCCD_G;
38  wire      [11:0]  sCCD_B;
39  wire              sCCD_DVAL;
40
41  wire      [9:0]   VGA_R;  //VGA Red[9:0]
42  wire      [9:0]   VGA_G;  //VGA Green[9:0]
43  wire      [9:0]   VGA_B;  //VGA Blue[9:0]
44  reg               [1:0]   rClk;
45  wire                      sdram_ctrl_clk;
46
47
48  // NIOS wires
49  wire      [5:0]   NIOS_LUT_INDEX;
50  wire      [23:0]  NIOS_LUT_DATA;
51
52  wire              out_nios_start_cam;
53  wire              out_nios_stop_cam;
54  wire              out_nios_reset_camsystem;
55  wire              pll_nios;
56  wire              NiosRST;
57  wire              out_nios_RST;
58  wire      [11:0]  mDATA_0_wire;
59  wire      [11:0]  mDATAd_0_wire;
60  wire      [11:0]  mDATA_1_wire;
61  wire      [11:0]  mDATAd_1_wire;
62
63  wire              PIX_VALCLK_wire;
64  wire      [2:0]   out_nios_HW_function;
65  wire              out_nios_HW_SW_SEL;
66
67  wire      [11:0]  HWsCCD_R;
68  wire      [11:0]  HWsCCD_G;
69  wire      [11:0]  HWsCCD_B;
70  wire              HWsCCD_DVAL;
71
72
```

```
73  wire    [11:0]  SWsCCD_R;
74  wire    [11:0]  SWsCCD_G;
75  wire    [11:0]  SWsCCD_B;
76  wire            SWsCCD_DVAL;
77  wire            SWsPULSE_DVAL;
78  wire            MUX_out_CLK;
79
80  //=============================================
81  // Structural coding
82  //=============================================
83  assign  CCD_DATA[0]     =       GPIO_1[13];
84  assign  CCD_DATA[1]     =       GPIO_1[12];
85  assign  CCD_DATA[2]     =       GPIO_1[11];
86  assign  CCD_DATA[3]     =       GPIO_1[10];
87  assign  CCD_DATA[4]     =       GPIO_1[9];
88  assign  CCD_DATA[5]     =       GPIO_1[8];
89  assign  CCD_DATA[6]     =       GPIO_1[7];
90  assign  CCD_DATA[7]     =       GPIO_1[6];
91  assign  CCD_DATA[8]     =       GPIO_1[5];
92  assign  CCD_DATA[9]     =       GPIO_1[4];
93  assign  CCD_DATA[10]=   GPIO_1[3];
94  assign  CCD_DATA[11]=   GPIO_1[1];
95
96  //CCD_MCLK (master clock on CCD) is 25MHz - clock_50 /2 rClk[0]
97  assign  GPIO_1[16]      =       CCD_MCLK;
98  assign  CCD_FVAL        =       GPIO_1[22];
99  assign  CCD_LVAL        =       GPIO_1[21];
100 assign  CCD_PIXCLK      =       GPIO_1[0];
101 assign  GPIO_1[19]      =       1'b1;  // tRIGGER
102 assign  GPIO_1[17]      =       DLY_RST_1;
103
104 //assign        LEDR            =       SW;
105 assign  LEDG            =       Y_Cont;
106
107 assign  VGA_CTRL_CLK=   rClk[0]; //25MHz
108 assign  VGA_CLK         =       ~rClk[0];
109
110 assign LEDR[0] = out_nios_start_cam;
111 assign LEDR[1] = out_nios_stop_cam;
112
113 //PLL register function rClk[0] gives 25MHz to VGA and CLK-in to camera
114 always@(posedge CLOCK_50)       rClk    <=      rClk+1;
115
```

```
116
117 always@(posedge CCD_PIXCLK)
118 begin
119         rCCD_DATA       <=      CCD_DATA;
120         rCCD_LVAL       <=      CCD_LVAL;
121         rCCD_FVAL       <=      CCD_FVAL;
122 end
123
124
125 VGA_Controller  u1 (//  Host Side
126                     .oRequest(Read),
127                     .iRed(Read_DATA2[9:0]),
128                     .iGreen({Read_DATA1[14:10],Read_DATA2[14:10]}),
129                     .iBlue(Read_DATA1[9:0]),
130
131                     //      VGA Side
132                     .oVGA_R(VGA_R),
133                     .oVGA_G(VGA_G),
134                     .oVGA_B(VGA_B),
135                     .oVGA_H_SYNC(VGA_HS),
136                     .oVGA_V_SYNC(VGA_VS),
137                     .oVGA_SYNC(VGA_SYNC),
138                     .oVGA_BLANK(VGA_BLANK),
139                     //      Control Signal
140                     .iCLK(VGA_CTRL_CLK),
141                     .iRST_N(DLY_RST_2)
142 );
143
144
145 Reset_Delay     u2(
146                     .iCLK(CLOCK_50),
147                     .iRST(KEY[0]),
148                     .iNiosRST(out_nios_RST),
149                     .oRST_0(DLY_RST_0),
150                     .oRST_1(DLY_RST_1),
151                     .oRST_2(DLY_RST_2)
152 );
153
154 CCD_Capture     u3(
155                     .oDATA(mCCD_DATA),
156                     .oDVAL(mCCD_DVAL),
157                     .oX_Cont(X_Cont),
158                     .oY_Cont(Y_Cont),
```

```
159                     .oFrame_Cont(Frame_Cont),
160                     .iDATA(rCCD_DATA),
161                     .iFVAL(rCCD_FVAL),
162                     .iLVAL(rCCD_LVAL),
163                     .iSTART(out_nios_start_cam),
164                     .iEND(out_nios_stop_cam),
165                     .iCLK(CCD_PIXCLK),
166                     .iRST(DLY_RST_2)
167 );
168
169 RAW2RGB             u4(
170                     .iCLK(CCD_PIXCLK),
171                     .iRST(DLY_RST_1),
172                     .iDATA(mCCD_DATA),
173                     .iDVAL(mCCD_DVAL),
174                     .oRed(HWsCCD_R),
175                     .oGreen(HWsCCD_G),
176                     .oBlue(HWsCCD_B),
177                     .oDVAL(HWsCCD_DVAL),
178                     .omDATA_0(mDATA_0_wire),
179                     .omDATAd_0(mDATAd_0_wire),
180                     .omDATA_1(mDATA_1_wire),
181                     .omDATAd_1(mDATAd_1_wire),
182                     .oPIX_VALCLK(PIX_VALCLK_wire),
183                     .iX_Cont(X_Cont),
184                     .iY_Cont(Y_Cont),
185                     .iFUNC(out_nios_HW_function)
186 );
187
188 SEG7_LUT_8          u5(
189                     .oSEG0(HEX0),.oSEG1(HEX1),
190                     .oSEG2(HEX2),.oSEG3(HEX3),
191                     .oSEG4(HEX4),.oSEG5(HEX5),
192                     .oSEG6(HEX6),.oSEG7(HEX7),
193                     .iDIG(Frame_Cont[31:0])
194 );
195
196 sdram_pll           u6(
197                     .inclk0(CLOCK_50),
198                     //co gives out a clock rate of 125MHz
199                     .c0(sdram_ctrl_clk),
200                     //c1 gives out a clock rate of 125MHz with phase shift -3ns
201                     .c1(DRAM_CLK),
```

```
202                 //c2 gives out a clok rate of 150MHz
203                 .c2(pll_nios)
204                                                     );
205 //Uses a register to divide the CCD_MCLK by two, gives 50MHZ/2=25MHz
206 assign CCD_MCLK = rClk[0];
207
208 Sdram_Control_4Port      u7(
209                 //HOST Side
210                 .REF_CLK(CLOCK_50),
211                 .RESET_N(1'b1),
212                 .CLK(sdram_ctrl_clk),
213
214                 //FIFO Write Side 1
215                 .WR1_DATA({1'b0,sCCD_G[11:7],sCCD_B[11:2]}),
216                 .WR1(sCCD_DVAL),
217                 .WR1_ADDR(0),
218                 .WR1_MAX_ADDR(640*480),
219                 .WR1_LENGTH(9'h100),
220                 .WR1_LOAD(!DLY_RST_0),
221                 .WR1_CLK(~MUX_out_CLK),
222
223                 //FIFO Write Side 2
224                 .WR2_DATA({1'b0,sCCD_G[6:2],sCCD_R[11:2]}),
225                 .WR2(sCCD_DVAL),
226                 .WR2_ADDR(22'h100000),
227                 .WR2_MAX_ADDR(22'h100000+640*480),
228                 .WR2_LENGTH(9'h100),
229                 .WR2_LOAD(!DLY_RST_0),
230                 .WR2_CLK(~MUX_out_CLK),
231
232
233                 //FIFO Read Side 1
234                 .RD1_DATA(Read_DATA1),
235                 .RD1(Read),
236                 .RD1_ADDR(0),
237                 .RD1_MAX_ADDR(640*480),
238                 .RD1_LENGTH(9'h100),
239                 .RD1_LOAD(!DLY_RST_0),
240                 .RD1_CLK(~VGA_CTRL_CLK),
241
242                 //FIFO Read Side 2
243                 .RD2_DATA(Read_DATA2),
244                 .RD2(Read),
```

```
245                     .RD2_ADDR(22'h100000),
246                     .RD2_MAX_ADDR(22'h100000+640*480),
247                     .RD2_LENGTH(9'h100),
248                     .RD2_LOAD(!DLY_RST_0),
249                     .RD2_CLK(~VGA_CTRL_CLK),
250
251                     //SDRAM Side
252                     .SA(DRAM_ADDR),
253                     .BA({DRAM_BA_1,DRAM_BA_0}),
254                     .CS_N(DRAM_CS_N),
255                     .CKE(DRAM_CKE),
256                     .RAS_N(DRAM_RAS_N),
257                     .CAS_N(DRAM_CAS_N),
258                     .WE_N(DRAM_WE_N),
259                     .DQ(DRAM_DQ),
260                     .DQM({DRAM_UDQM,DRAM_LDQM})
261 );
262
263
264 assign  UART_TXD = UART_RXD;
265
266 I2C_CCD_Config  u8(
267                     //Host Side
268                     .iCLK(CLOCK_50),
269                     .iRST_N(DLY_RST_2),
270                     .iZOOM_MODE_SW(SW[16]),
271                     .iEXPOSURE_ADJ(KEY[1]),
272                     .iEXPOSURE_DEC_p(SW[0]),
273                     //Nios connection to read lut_index
274                     //and give the corresponding configuration data
275                     .iNIOS_LUT_DATA(NIOS_LUT_DATA),
276                     .oNIOS_LUT_INDEX(NIOS_LUT_INDEX),
277                     //      I2C Side
278                     .I2C_SCLK(GPIO_1[24]),
279                     .I2C_SDAT(GPIO_1[23])
280 );
281
282 assign  LCD_ON          =       1'b1;// LCD Power ON/OFF
283 assign  LCD_BLON        =       1'b1;// LCD Back Light ON/OFF
284
285 niosii  u9      (
286
287 //INPUTS
```

```
288
289 .clk(pll_nios),
290 .reset_n(KEY[0]),
291 //[7..0] 8 bit input, INTERRUPT rising edge
292 .in_port_to_the_SWint(SW[7:0]),
293 //[7..0] 8 bit input, INTERRUPT rising edge
294 .in_port_to_the_SWint2(SW[15:8]),
295
296 //Input from the I2C_CCD_CONFIG module with the index of which of
297 //the configuration-data to transfer. Interrupt -any-edge
298 .in_port_to_the_LUT_INDEX(NIOS_LUT_INDEX),
299
300 //[1..0] X(11bit)[0] and Y(11bit)[0] - DATA
301 .in_port_to_the_CCD_XY({Y_Cont[0],X_Cont[0]}),
302 //mDATA_0[11..0]
303 .in_port_to_the_IN_mDATA_0(mDATA_0_wire),
304 //mDATAd_0[11..0]
305 .in_port_to_the_IN_mDATAd_0(mDATAd_0_wire),
306 //mDATA_1[11..0]
307 .in_port_to_the_IN_mDATA_1(mDATA_1_wire),
308 //mDATAd_1[11..0]
309 .in_port_to_the_IN_mDATAd_1(mDATAd_1_wire),
310 //PIXCLK & DVALID -> 1 bit PIX-VALCLK
311 .in_port_to_the_PIX_VALCLK(PIX_VALCLK_wire),
312
313 //OUTPUTS
314
315 //To reset the HW
316 .out_port_from_the_RESET_OUT(out_nios_RST),
317 //[7..0]
318 .out_port_from_the_LEDs(
319 {LEDR[7:2],out_nios_stop_cam,out_nios_start_cam}),
320 //[23..0]
321 .out_port_from_the_LUT_DATA(NIOS_LUT_DATA),
322 //Red data (12bit)
323 .out_port_from_the_RED(SWsCCD_R),
324 //Green data (12bit)
325 .out_port_from_the_GREEN(SWsCCD_G),
326 //Blue data (12bit)
327 .out_port_from_the_BLUE(SWsCCD_B),
328 //Data valid (1bit)
329 .out_port_from_the_DVAL(SWsCCD_DVAL),
330 //[3..0] Control signals to RAW2RGB
```

```
331  .out_port_from_the_RAW2RGB_CONTROL_OUT(
332  {out_nios_HW_SW_SEL,out_nios_HW_function}),
333
334  //LCD symbol display SIGNALS
335  .LCD_E_from_the_LCD_0(LCD_EN), //1bit out
336  .LCD_RS_from_the_LCD_0(LCD_RS), //1bit out
337  .LCD_RW_from_the_LCD_0(LCD_RW), //1bit out
338  .LCD_data_to_and_from_the_LCD_0(LCD_DATA) //[7..0] inout
339
340                                );
341
342  HW_SW_MUX          u10(//HW inputs
343                         .iHW_R(HWsCCD_R),
344                         .iHW_G(HWsCCD_G),
345                         .iHW_B(HWsCCD_B),
346                         .iHW_V(HWsCCD_DVAL),
347                         .iHW_CLK(CCD_PIXCLK),
348                         //SW inputs
349                         .iSW_R(SWsCCD_R),
350                         .iSW_G(SWsCCD_G),
351                         .iSW_B(SWsCCD_B),
352                         .iSW_V(SWsPULSE_DVAL),
353                         .iSW_CLK(pll_nios),
354                         //Input SEL, SB from RAW2RGB_CONTROL_OUT
355                         // default = '0' -> HW processing
356                         .iSEL(out_nios_HW_SW_SEL),
357                         //MUX outputs
358                         .oR(sCCD_R),
359                         .oG(sCCD_G),
360                         .oB(sCCD_B),
361                         .oV(sCCD_DVAL),
362                         .oCLK(MUX_out_CLK)
363
364  );
365
366  SINGLEPULSEGEN     u11(//Inputs
367                         .iCLK(pll_nios),
368                         .iSIG(SWsCCD_DVAL),
369                         //Outputs
370                         .oPULSE(SWsPULSE_DVAL)
371
372  );
373
```

```
374 endmodule
```

Listing B.1: A part of the Top file "DE2_D5M"

```
1  //This is a part of the RAW2RGB Verilog module made by Terasic
2
3  module RAW2RGB(
4    //Outputs
5    oRed,
6    oGreen,
7    oBlue,
8    oDVAL,
9    omDATA_0,
10   omDATAd_0,
11   omDATA_1,
12   omDATAd_1,
13   oPIX_VALCLK,
14   //Inputs
15   iX_Cont,
16   iY_Cont,
17   iFUNC,
18   iDATA,
19   iDVAL,
20   iCLK,
21   iRST
22 );
23
24 input    [10:0]  iX_Cont;
25 input    [10:0]  iY_Cont;
26 input    [11:0]  iDATA;
27 input            iDVAL;
28 input            iCLK;
29 input            iRST;
30 input    [2:0]   iFUNC;
31 output   [11:0]  oRed;
32 output   [11:0]  oGreen;
33 output   [11:0]  oBlue;
34 output           oDVAL;
35 output   [11:0]  omDATA_0;
36 output   [11:0]  omDATAd_0;
37 output   [11:0]  omDATA_1;
38 output   [11:0]  omDATAd_1;
39 output           oPIX_VALCLK;
40 wire     [11:0]  mDATA_0;
```

```
41  wire     [11:0]  mDATA_1;
42  wire             INV_COLOR;
43  reg      [11:0]  mDATAd_0;
44  reg      [11:0]  mDATAd_1;
45  reg      [11:0]  mCCD_R;
46  reg      [12:0]  mCCD_G;
47  reg      [11:0]  mCCD_B;
48  reg              mDVAL;
49
50  //------------------ Added functionality
51  //If INV_COLOR == '1' then invert the data else normal color mode
52  assign   oRed             =  INV_COLOR ? ~mCCD_R[11:0] : mCCD_R[11:0];
53  //Divides by two because of the addition of the green in the demosaic
54  assign   oGreen           =  INV_COLOR ? ~mCCD_G[12:1] : mCCD_G[12:1];
55  assign   oBlue            =  INV_COLOR ? ~mCCD_B[11:0] : mCCD_B[11:0];
56
57  assign   oDVAL            =  mDVAL;
58  assign   omDATA_0         =  mDATA_0[11:0];
59  assign   omDATAd_0        =  mDATAd_0[11:0];
60  assign   omDATA_1         =  mDATA_1[11:0];
61  assign   omDATAd_1        =  mDATAd_1[11:0];
62  assign   oPIX_VALCLK      =  iDVAL&iCLK;
63  assign   INV_COLOR        =  iFUNC[0:0];
64
65  //------------------
66
67  //The Line_Buffer is a 2tap shift register with a 1280
68  //clock width between the taps
69  //(1280 is the same as the column size from the camera)
70  //Because of this width the two taps presents two lines
71  //simultanious which is used to be able to read 2x2 pixel
72  //blocks for the demosiac algorithm.
73  Line_Buffer      u0(
74    .clken(iDVAL),
75    .clock(iCLK),
76    .shiftin(iDATA),
77    .taps0x(mDATA_1),
78    .taps1x(mDATA_0)
79  );
80
81  always@(posedge iCLK or negedge iRST)
82  begin
83  if(!iRST)
```

```
84  begin
85    mCCD_R          <=  0;
86    mCCD_G          <=  0;
87    mCCD_B          <=  0;
88    mDATAd_0        <=  0;
89    mDATAd_1        <=  0;
90    mDVAL           <=  0;
91  end
92  else
93  begin
94  mDATAd_0          <=  mDATA_0;
95  mDATAd_1          <=  mDATA_1;
96
97  //This just gives the frame buffer valid data 1/4 th of the time
98  mDVAL             <={iY_Cont[0]|iX_Cont[0]}?1'b0:iDVAL;
99
100 if({iY_Cont[0],iX_Cont[0]}==2'b10)//Y=odd and X=even => Red data
101 begin
102   mCCD_R          <=  mDATA_0;
103   mCCD_G          <=  mDATAd_0+mDATA_1;
104   mCCD_B          <=  mDATAd_1;
105 end
106
107 else if({iY_Cont[0],iX_Cont[0]}==2'b11)//Y=odd and X=odd => Green1 data
108 begin
109 mCCD_R            <=  mDATAd_0;
110 mCCD_G            <=  mDATA_0+mDATAd_1;
111 mCCD_B            <=  mDATA_1;
112 end
113
114 //This, when x[0] and y[0] both are equal 00 is the only
115 //one that gives data valid out.
116 //The reason for not using all the four combinations
117 //is to reduce the output resolution
118 //to 640x480 by interpolation 1 of 4 pixels.
119 //This equals a step of two horizontally and a step
120 //of two vertically
121 else if({iY_Cont[0],iX_Cont[0]}==2'b00)//Y=even and X=even => Green2 data
122 begin
123 mCCD_R            <=  mDATA_1;
124 mCCD_G            <=  mDATA_0+mDATAd_1;
125 mCCD_B            <=  mDATAd_0;
126 end
```

```
127
128 else if({iY_Cont[0],iX_Cont[0]}==2'b01)//Y=even and X=odd => Blue data
129 begin
130 mCCD_R     <= mDATAd_1;
131 mCCD_G     <= mDATAd_0+mDATA_1;
132 mCCD_B     <= mDATA_0;
133 end
134 end
135 end
136
137 endmodule
```

Listing B.2: The RAW2RGB hardware video processor