NTNU

Norwegian University of
Science and Technology

Embedded System for Electronic Circuit Education

Kai André Venjum

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: 22. January 2010
Supervisor: Per Gunnar Kjeldsberg, IET

Abstract

Embedded systems are ideal as electronic demonstrators because they provides the designer with wide possibilities for optimization through codesign. In many situations, like school visits at the Norwegian University of Science and Technology (NTNU), "Forskningstorget" and "Elektronikk- & Telekommunikasjonsdagen", it is desirable for the Department of Electronics and Telecommunication to both motivate and recruit new students to a future career in electronics. Thus, a demonstrator with an interesting presentation may give students an insight in what is possible when studying electronics at NTNU, in addition to a good examples of an electronic system. A good demonstrator for the department include one or more electronic topics and presents the relevant theory in different educational levels

This master thesis includes the implementation of an embedded system used for demonstrating basic electronics. The Embedded System for Electronic Circuit Education is a platform for easy implementation of several scenarios with different topics within electronics. The system is designed with respect to a pedagogical view, and is implemented on both the Altera DE2 and the Atmel AVR STK600. In addition the embedded demonstrator includes a monitor for user interface and demonstration materiel. The main modules in the demonstrator is the Cyclone II 2C35 FPGA and the AVR AT90USB1287 microcontroller used to control the system behavior and the user interface. The demonstrator already includes two example scenarios, namely the "Automatic Adjustment of Light" and "How to Count in Binary, Hexadecimal and Decimal" with the topics Electronic Components and Numerical Systems respectively.

With both existing and future scenarios, the embedded demonstrator has the possibility to both motivate and activate students with an interactive interface. In addition, the demonstrator may individualize educational levels to the different target groups with the demonstration material displayed on the monitor. Thus, the Embedded System for Electronic Circuit Education is a well suited demonstrator to recruit and motivate students to a future carrier in electronics.

Preface

This master thesis is the finish line of my five years as a student in Trondheim. These years have without a doubt been the best years of my life and I will always carry the memories with me.

The goal with this project has been to design and implement an embedded demonstrator for Department of Electronics and Telecommunication at NTNU. This demonstrator is intended to help the department to recruit future students to NTNU. The thoughts of motivating younger pupils to a five years master study, hopefully with much fun and future good memories, together with practical use of courses through time, was the main motivation for this project.

The last six months have been both exciting and challenging at the same time. It seemed like the learning process through this project would never stop. Now a new era of my life begins with new challenges. The literature study of both the Altera DE2 and Atmel STK600 have been fulfilling. However, due to time constrains, a dynamic frame buffer could not be implemented.

Several people have contributed to this report, without them, both the report and the product had not been the same. I want to thank my supervisor Per Gunnar Kjeldsberg at Department of Electronics and Telecommunication for advices and guidelines through the whole process. I am also thankful to Cato Marwell Jonassen and Jarle Larsen for their collaboration through the five years and particularly this last six months. I am also thankful to Cato Marwell Jonassen for our collaboration with the "One Pulse Generator" and to Ingulf Helland for his help with the VGA-Controller. Finally, I am particularly grateful for the support from Jarle Larsen and Ingrid Vågan Tøgersen.

Kai André Venjum

NTNU, Trondheim

June 2010

Definitions

ADC : Analog-to-Digital Converter - converts continuous signals to discrete digital numbers

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

DAC : Digital-to-Analog Converter - converts a digital code to an analog signal

DMA : Direct Memory Access - allows certain hardware subsystems within a computer to access system memory for reading and/or writing independently of the central processing unit

FIFO : First In, First Out - This expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served

FLASH : Flash memory - a non-volatile computer storage technology that can be electrically erased and reprogrammed

FPGA : Field-Programmable Gate Array - an integrated circuit designed to be configured by the customer or designer after manufacturing

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

HDL : Hardware Descriptive Language - a language for describing hardware

ISR : Interrupt Service Routine - a callback subroutine in an operating system or device driver whose execution is triggered by the reception of an interrupt

LSB : Least Significant Bit - the bit position in a binary number having the smalest value

MSB : Most Significant Bit - the bit position in a binary number having the greatest value

PLL : Phase-locked loop - generate a frequency that is a multiple of the input frequency

PWM : Pulse-Width Modulation - uses a rectangular pulse wave whose pulse width is modulated resulting in the variation of the average value of the waveform

RISC : Reduced Instruction-Set Computing - a CPU design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction

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

SPI : Serial Peripheral Interface - is a synchronous serial data link standard that operates in full duplex mode

SRAM : Static Random Access Memory - a type of volatile memory which do not need to be refreshed, but it still need power to contain its data

USB : Universal Serial Bus - a specification to establish communication between devices and a host controller

VGA : Video Graphics Array - may refere to the analog computer display standard, the 15 pin d-sub connector or the 640x480 resolution

Contents

Chapter 1

Introduction

In many situations, like "Elektronikk- & Telekommunikasjonsdagen" and school visits at NTNU, it is important for Department of Electronics and Telecommunication to get peoples' attention, and to promote themselves to potential future students. In these situations, a demonstrator may be used to present the courses provided by the department, what electronic is and where it can be used. A demonstration may consist of a professor talking about the mentioned topics, but in this project it is desired to show concrete examples of practical use of electronics.

The main goal with a demonstration is to get people interested in the presented topics. To achieve this, the demonstrator should both be interactive and have familiar topics from the students everyday life. In addition, an ideal demonstrator should also provide a level of education in its demonstration.

This report implements the specifications of an embedded demonstrator described in [33]. The demonstrator specifies a platform for easy implementation of different scenarios for electronic topics. Two scenarios are already included, and provides education in both numerical systems and automatic adjustment of light. The platform and the included scenarios are specified with the theory influenced by a pedagogical view.

1.1 Preliminary Work

The motivation and the purpose of the mentioned report was to make a demonstrator to recruit new students to Department of Electronics and Telecommunication at NTNU. The implemented demonstrator should be both interactive and include electronic educational aspects. The best solution to achieve the requirements was considered to be the design of a platform for easy implementation of several scenarios with different educational levels, called an Embedded System for Electronic Circuit Education. This system provides the designer with the opportunity to implement several scenarios with the use of

different methods implemented in the Altera DE2 and an interactive component board. The interactive component board is used as an interface between the user and the digital domain, and contains both an analog-to-digital converter (ADC) and a digital-to-analog converter (DAC), together with a selection of analog components indented for user interaction. In addition to these two modules, the Altera DE2 is connected to a monitor used to present scenario information and relevant theory that may be adjusted to the educational level, as shown in Figure 1.1.

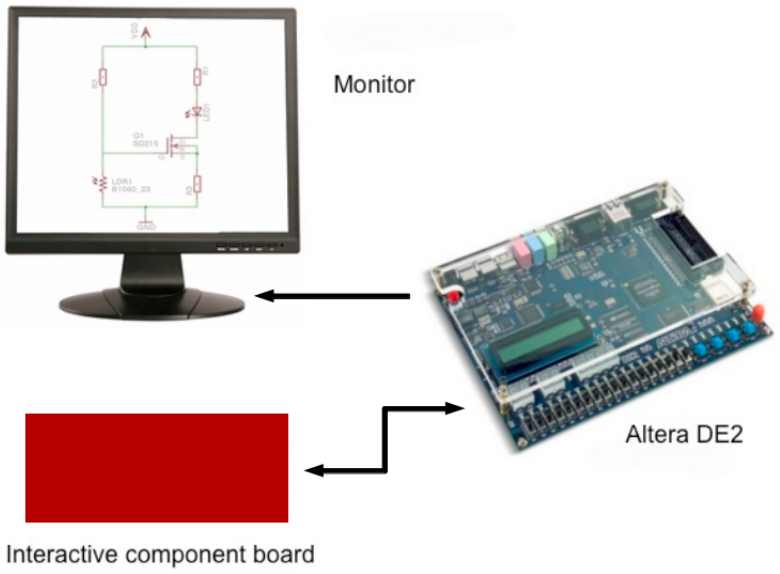

Figure 1.1: Preliminary System Overview

The idea for this system is to let the user choose between the scenarios that is most interesting for him or her. An example of a scenario is Automatic Adjustment of Light, where the interactive component board is used together with Altera DE2 to demonstrate the effects of LDRs and LEDs. In addition, a virtual circuit on the monitor displays how the currents and the voltages changes in the circuit during operation, thus providing the user with a basic education in electronic circuit theory.

1.2 Embedded systems

According to John Catsoulis [14], computer systems fall into two categories. The first is the desktop computer. When someone says computer, this is the machine that usually come to their mind. The second type of computer is the embedded computer, a computer that is integrated into another system for the purpose of control and/or monitoring. The term embedded means being part of a larger unit and providing a dedicated service to that unit [24].

Both desktop computers and embedded computers consist of a processor, memory and often several forms of input and output [14]. Embedded hardware is often much simpler than a desktop hardware, but it can also be more complex. An embedded system can be implemented on a single chip with just a few support components, or it can consist of hundreds of processors working in parallel. The main difference between them is their intend in use. Desktop computers can run a variety of application programs, with system resources assigned by an operating system. In contrast, the embedded computer is normally dedicated to a specific task. In many cases, an embedded computer is used to replace application-specific electronics.

The embedded system typically has one application and only one, which is permanently running. The embedded computer may or may not have an operating system, and it rarely provides the user with the ability to arbitrarily install new software. The software is normally contained in the system's nonvolatile memory.

The advantage of using an embedded microprocessor over dedicated electronics is that the functionality of the system is determined by the software, not the hardware. This makes the embedded system easier to produce, and much easier to develop, than a complicated circuit [14].

1.3 Codesign

Hardware/software co-design means meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design [24].

The process of performing hardware/software tradeoffs requires a direct interaction between hardware and software design [21]. Hardware/software tradeoffs refer to decisions regarding the allocation of functions into hardware and software that attempt to satisfy a set of objectives. Thus hardware/software tradeoffs affect the amount and mixture of software and hardware employed, and lead to other kinds of tradeoffs, such as performance versus cost.

The introduction of field-programmable gate array (FPGA) technologies blurred the distinction between hardware and software. With field-programmable technology it is possible to configure the gate-level interconnection of hardware circuits after manufacturing. This flexibility opens new applications of digital circuits, and new hardware/software co-design problems arise. For example, one FPGA circuit may be configured on-the-fly to implement a specific software function with better performance than executing the corresponding code on a microprocessor [24].

1.4 Characteristics for a good demonstrator

Some of the principles for good teaching is motivation, activation and individualization [17]. When teaching a group or a student, it is important that the teaching motivates and falls in interest for the target group. If the topics presented are known through their everyday life, it is easier to be motivated and to learn from the presentation. In order to achieve this, it can be a good idea to include the target group in activities where the group is included in the learning process.

People are different from each other, and they learn by different methods. Thus individualization is important to adjust the education to the individual needs and qualifications [16]. This is why schools have classes based on age, and why teaching is different in kindergarden and high school. Differentiation is the name of this theory, and age is one of the oldest methods to differentiate in the school system. In theory, the most perfect differentiation of a group is individualization, but in practice this is not possible with large groups[17].

When designing an embedded demonstrator it is important to have topics like differentiation, activation and motivation, as discussed above, in mind. The goal with this demonstrator is to recruite new students from higher secondary school. However, it is also desirable to motivate other students from lower levels to consider electronics as a future career. The age of the target group is thus important for the design, and it should be possible to adjust the difficulty according to the person or group who attends the presentation. Whether someone presents the demonstrator or the demonstrator is capable of introducing itself, it is important to remember that students have different qualifications related to their background. Due to this, a demonstrator should include the possibility for different levels of understanding.

To maintain the students interest in the demonstration, it is a good idea to allow the presentation to be interactive. When the user is interactive with the system, it is easier to keep him or her motivated. Another method for motivating pupils or students, is to relate the topics of the demonstrator to familiar devices in their everyday life. In addition, since the demonstrator will be used to recruit students to the department, it should also be relevant for courses that the department provides. In this way it can strengthen the future students idea of what electronics are and where it can be used.

1.5 Main Contributions

The main contributions in this report are:

- A presentation of the necessary theory to design both the platforms and scenarios
- The implementation of a VGA-Controller in VHDL
- The realization of a platform for implementation of scenarios

- The implementation of two different scenarios
- A implementation guide to future scenarios

1.6 Structure of Report

This report is divided into the following chapters: Chapter 2 presents the design tools used to implement the embedded demonstrator. In Chapter 3 and 4 the platforms used in the implementation process, namely the Altera DE2 and the Atmel STK600, and their related theory, are described in detail. Further, in Chapter 5, the communication interface between the two development platforms are discussed. Chapter 6 and 7 describes the theory needed to understand the implemented scenarios, while Chapter 8 and 9 presents the system overview and an in-depth discussion of the system implementation, together with a description on how to implement future scenarios. In Chapter 10, a guide to how the example scenarios should be demonstrated is presented. The final chapter, Chapter 11, presents the main conclusions of the system implementation.

Chapter 2

Design Tools

This chapter describes the design tools used for the design and test of the Embedded System for Electronic Circuit Education. When designing an embedded system, either an ASIC or a FPGA are needed for hardware circuits. Designing one ASIC is very expansive, thus a FPGA is a reasonable choice. There are in general two options for FPGAs and their design tools. Xilinx and Altera are the market leaders in FPGAs, and together they control over 80 percent of the market [1]. However, there are several options for microcontrollers, for example Atmel, Texas Instruments and ARM, and the choice of tools and hardware for this system was based on their availability at the department, and the design tools already familiar.

2.1 Quartus II

There are several EDA (Electronic Design Automation) tools available for circuit synthesis, implementation and simulation using VHDL or Verilog. The Altera Quartus II is one of them[28]. This design software includes a multi-platform design environment that can adapt to the user specific design needs. The Quartus II software includes solutions for the different phases of FPGA design, as seen in Figure 2.1.

The Quartus II software allows the designer to use the a graphical user- and command-line interface for each phase of the design flow. It is possible to use one of these interfaces for the entire flow, or use different options at different phases [6].

SOPC Builder

The SOPC (System-On-a-Programmable-Chip) Builder is a system development tool for creating systems based on processors, peripherals, memories and communication interfaces like USB and SPI [25]. USB and SPI are described in Chapter 5. SOPC

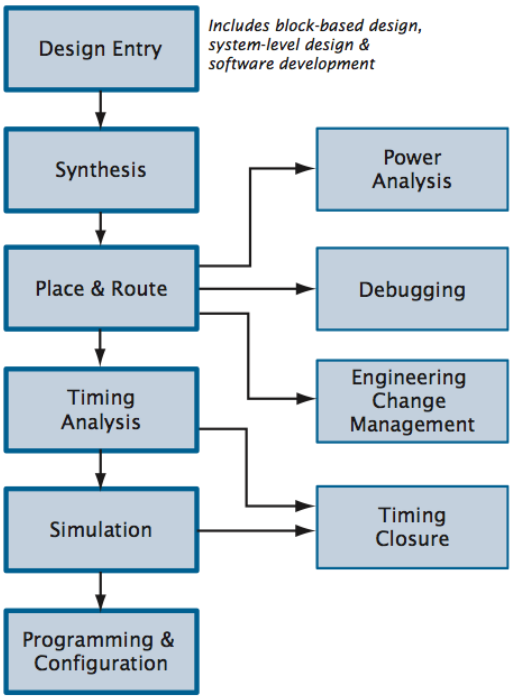

Figure 2.1: Quartus II Design Flow [6]

Builder enables the designer to define and generate a complete SOPC, and in a system design is included as part of the Quartus II software [9].

The SOPC Builder automates the task of integrating hardware components [9]. When using traditional design methods, the designer must manually write HDL modules to wire together the pieces of the system. With SOPC Builder, the designer only needs to specify the system components in a GUI, and SOPC Builder generates the interconnect logic automatically. SOPC Builder generates HDL files that defines all components of the system, and a top-level HDL file that connects all the components together. SOPC Builder generates either Verilog HDL or VHDL equally [9].

An another approach, is to use the MegaWizard plug-in manager. The MegaWizard can create most of same modules as SOPC builder. However, the MegaWizard can run as a stand-alone utility. Thus a SOPC system is not needed. An example of use, is when a hardware system only needs a PLL [9].

2.2 Nios II Embedded Design Suite

Writing software for the Nios II processor is similar to the software development process for any other microcontroller family [8]. The Nios II EDS provides a consistent software

development environment that works for all Nios II processor systems. It also includes many proprietary and open-source tools such as the GNU C/C++ tool chain for creating Nios II programs. In addition, the Nios II EDS has a build-in Hardware Abstraction Layer (HAL), which is a lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware [7]. The HAL application program interface, or API, is integrated with the ANSI C standard library.

With a PC, an Altera FPGA and a JTAG download cable, it is possible to write programs for, and communicate with, any Nios II processor system. The Nios II processor's JTAG debug module provides a single, consistent method to communicate with the processor using a JTAG download cable. Accessing the processor is the same, regardless of whether a device only implements a Nios II processor system, or whether the Nios II processor is embedded deeply in a complex multiprocessor system. Furthermore, the Nios II EDS includes a flash programmer utility that allows to program flash memory chips on a target board [8].

2.3 ModelSim

ModelSim is a verification and simulation tool for VHDL, Verilog, SystemVerilog, and mixed-language designs [23]. The ModelSim includes a graphical user interface which consists of various windows that provides access to parts of the design and numerous debugging tools.

2.4 AVR Studio 4

The AVR Studio 4 is an Integrated Development Environment, IDE, for debugging AVR software [27]. The AVR Studio allows chip simulation, debugging and in-circuit emulation for the AVR family of microcontrollers. In addition, AVR Studio 4 provides a complete set of features, as well as target configuration and management, and full programming support for stand-alone programmers.

Chapter 3

Altera DE2 and Related Theory

In this chapter, different modules for the Altera Development and Education board are presented, in addition to the background theory for the VGA-control and the frame buffer.

3.1 Features of the Altera DE2

The Altera DE2 is a development board for test and design of systems. The development board has many features the designer may implement to improve the system functionality. Some of them are memory extensions like SRAM, SDRAM, flash memory and an SD-card socket [3]. For communication, the DE2 board has one USB host and one slave controller. It also provides RS-232, an IrDA transceiver, ethernet and the possibility for two expansion cards. The card also includes modules for audio and video implementation. For audio it is implemented a 24-bit CD quality audio codec. A VGA-module and a TV-decoder are included for video applications. The most important feature on the DE2 board is the Altera Cyclone II FPGA device that requires the Quartus II and Nios II Embedded Design suite for programming. The board has different switches and LEDs, plus a PS/2 connector for keyboard or mouse. In addition, it has a 2x16 digit LCD display and eight 7-segment displays. Figure 3.1 shows the Altera DE2 card and how the modules are placed.

When configuring the DE2 board there is an on board USB blaster that supports programming and user API control. These modules allow the user to implement a wide range of designed circuits, from simple circuits to various multimedia projects [3].

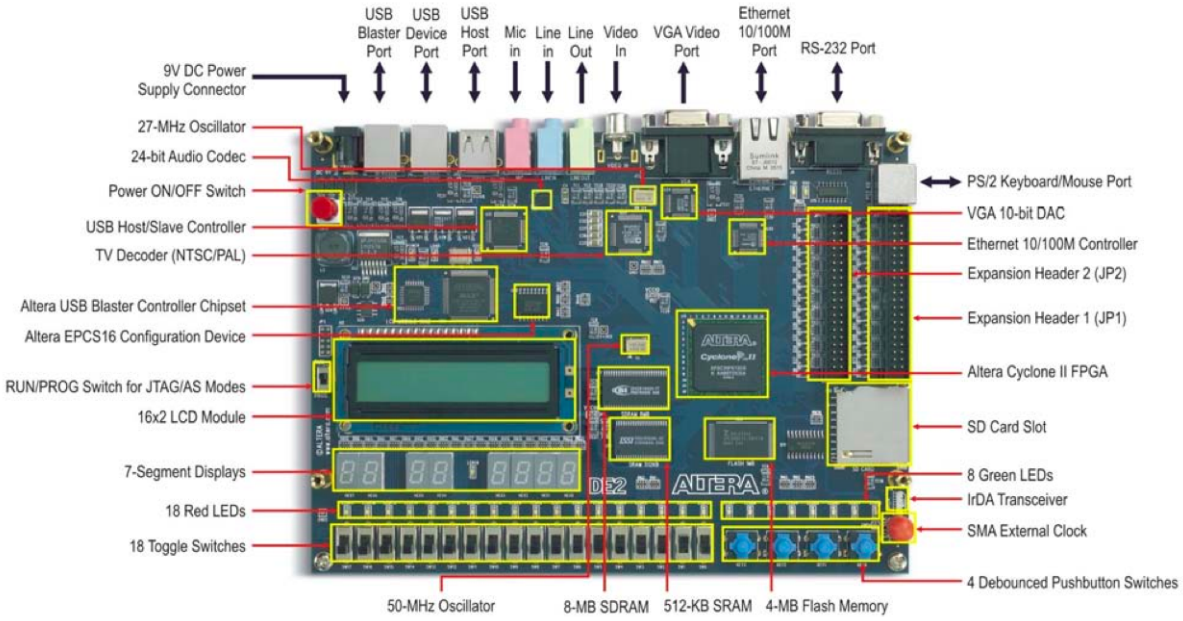

Figure 3.1: Altera DE2 Development and Education Board [3]

3.2 Altera Cyclone II 2C35

Cyclone II devices contain a two-dimensional row- and column-based architecture to implement custom logic [2]. Column and row interconnects of varying speed provide signal interconnects between logic array blocks, (LABs), embedded memory blocks, and embedded multipliers. The logic array consists of LABs, with 16 logic elements, (LEs) in each LAB. An LE is the smallest unit of logic in the Cyclone II architecture that provides efficient implementation of user logic functions. LABs are grouped into rows and columns across the device. The Cyclone II 2C35 FPGA includes 33,216 LEs [3].

The FPGA also contains 475 user I/O pins, a total memory of 105 M4K memory RAM blocks with 4608 bits in each block, 35 embedded multipliers and 4 PLLs. The M4K memory RAM blocks and the embedded multipliers are arranged in columns across the device. Cyclone II PLLs provide general-purpose clocking with clock synthesis and phase shifting as well as external outputs for high-speed differential I/O support [3].

3.3 Nios II

The Nios II processor core is a soft-core central processing unit, CPU, that can be programed along with other hardware components that comprise the Nios II system onto an Altera FPGA [7].

A Nios II processor system is equivalent to a microcontroller that includes a processor and a combination of peripherals and memory on a single chip [7]. A Nios II processor system consists of a Nios II processor core, a set of on-chip peripherals, on-chip memory, and interfaces to off-chip memory, all implemented on a single Altera device.

There are three different types of the Nios II core; economic, standard and fast. The Nios II/f fast core is designed for high execution performance[7]. Thus, the performance is gained at the expense of core size. The Nios II/s standard core is designed for small core sizes. Since the Nios II/s core uses less logic than the Nios II/f core, the execution performance drops by roughly 40%. The last one is the Nios II/e economy core. This core is designed to achieve the smallest possible core size. This core is roughly half the size of the Nios II/s core, but the execution performance is considerably lower [7].

With the Altera Nios II embedded processor, the system designer can accelerate time-critical software algorithms by adding custom instructions to the Nios II processor instruction set. When using custom instructions, it is possible to reduce a complex sequence of standard instructions to a single instruction implemented in hardware [4].

Interrupts

Like most microprocessors, the NIOS II supports interrupts [7]. Interrupts are a technique of diverting the processor from the execution of the current program so that it may deal with some event that has occurred [14]. Such an event may be an error from a peripheral, or simply that an I/O device has finished the last task it was given and is now ready for another.

Interrupts free the processor from having to continuously check the I/O devices to determine whether they require service [14]. Instead, the processor may continue with other tasks. The I/O devices will notify it when they require attention by asserting one of the processor's interrupt inputs.

When an interrupt occurs, the usual procedure is for the processor to save its state by pushing its registers and program counter onto the stack [14]. The processor then loads an interrupt-vector into the program counter. The interrupt-vector is the address at which an interrupt-service-routine (ISR) lies. Thus, loading the vector into the program counter causes the processor to begin execution of the ISR, performing whatever service the interrupting device requires. The last instruction of the ISR is always a return from interrupt instruction. This causes the processor to reload its saved state from the stack and resume its original program.

3.4 7-Segment Display

The 7-segment display is a package of seven LEDs [34]. The purpose of the 7-segment displays are to display information in form of numbers and some alphabetic characters. In addition, most 7-segment displays include a decimal point. Each segment needs one pin for cathode and one for anode. This means a total of sixteen pins for one single 7-segment display. To reduce the number of pins, common cathode or anode are usually applied. This reduces the number of pins to nine connections, one for each segment and one for power or ground.

The Altera DE2 Board has eight 7-segment displays [3]. These displays are arranged into two pairs and a group of four, with the intent of displaying numbers of various sizes. As indicated in the schematic in Figure 3.2, the seven segments are connected to pins on the Cyclone II FPGA. It is also worth mentioning that the dot in each display is unconnected and cannot be used. Each segment in a display is identified by an index from 0 to 6, with the positions given in Figure 3.3. Applying a low logic level to a segment causes it to light up, and applying a high logic level turns it off [3].

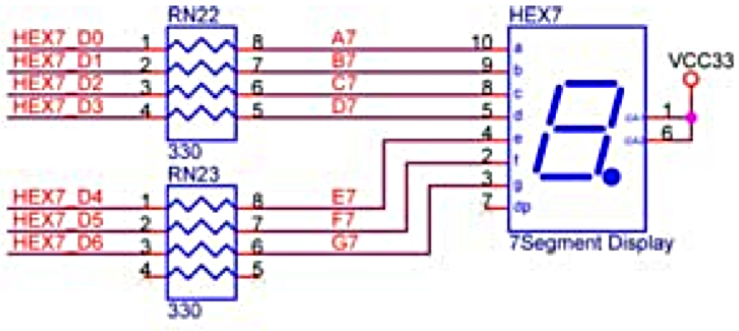

Figure 3.2: Schematic diagram of the 7-Segment Displays [3]

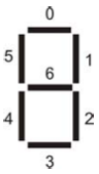

Figure 3.3: Position and index of each segment in a 7-Segment Display [3]

3.5 Monitor Control

VGA, or Video Graphics Array, is an analog video display port that is capable of displaying an unlimited number of colors by varying the level of red, green and blue per

pixel on screen [31][19]. The VGA video signal contains five active signals. The horizontal sync and vertical sync are used for synchronization of the video, and three analog signals with range from 0.7V to 1.0V peak-to-peak are used to control the color. The color signals are often collectively referred to as the RGB signals.

3.5.1 Video Display Technology

The major component inside early VGA computer monitors was the color CRT or Cathode Ray Tube shown in Figure 3.4 [19][10]. The electron beam must be scanned over the viewing screen in a sequence of horizontal lines to generate an image. The deflection yoke uses magnetic or electrostatic fields to deflect the electron beam to the appropriate position on the face of the CRT. The RGB color information in the video signal is used to control the strength of the electron beam. Light is generated when the beam is turned on by a video signal and it strikes a color phosphor dot or line on the face of the CRT. Figure 3.5 shows that the face of a color CRT contains a series of rows with three different phosphors. One type of phosphor is used for each of the primary colors of red, green and blue.

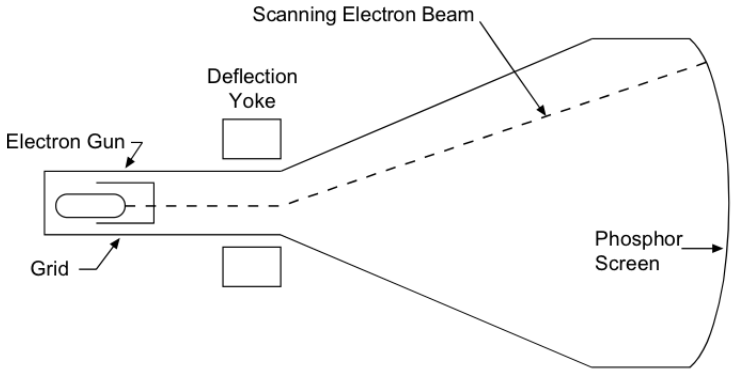

Figure 3.4: Color CRT [19]

In standard VGA format, as seen in Figure 3.6, the screen contains 640 by 480 picture elements or pixels [19]. The video signal must redraw the entire screen 60 times per second to provide for motion in the image and to reduce flicker. This period is called the refresh rate. The human eye can detect flicker at refresh rates less than 30 to 60Hz.

To reduce flicker from interference from fluorescent lighting sources, refresh rates higher than 60 Hz at around 70Hz are sometimes used in PC monitors [19][10]. The color of each pixel is determined by the value of the RGB signals when the signal scans across each pixel. In 640 by 480-pixel mode, with a 60Hz refresh rate, this is approximately 40 ns per pixel. A 25MHz clock has a period of 40 ns. A slightly higher clock rate will produce a higher refresh rate.

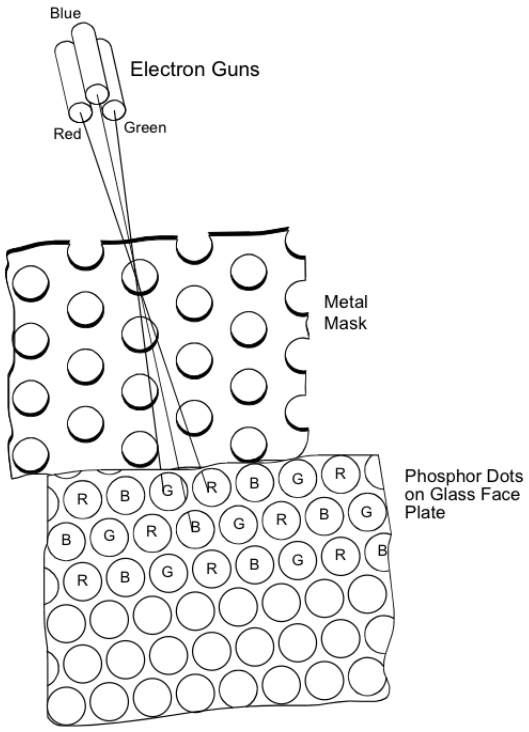

Figure 3.5: Phosphor Dots on Face of Display [19]

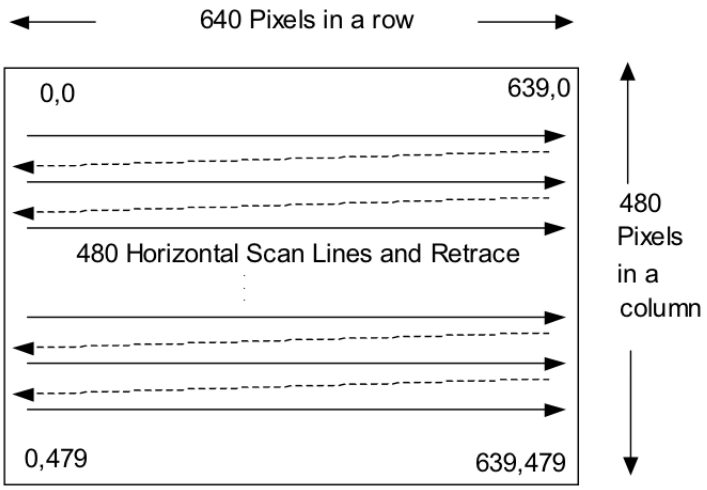

Figure 3.6: VGA Image - 640 by 480 Pixel Layout [19]

3.5.2 Video Refresh

The screen refresh process seen in Figure 3.6 begins in the top left corner and "paints" 1 pixel at a time from left to right [19]. At the end of the first row, the row increments and the column address is reset to the first column. Each row is "painted" until all pixels have been displayed. Once the entire screen has been "painted", the refresh process begins again.

The following process describes how the video signal "paints" or refreshes each image [19]. The vertical sync signal, as shown in Figure 3.7, tells the monitor to start displaying a new image or frame, and the monitor starts in the upper left corner with pixel 0,0. The horizontal sync signal, as shown in Figure 3.8, tells the monitor to refresh another row of 640 pixels.

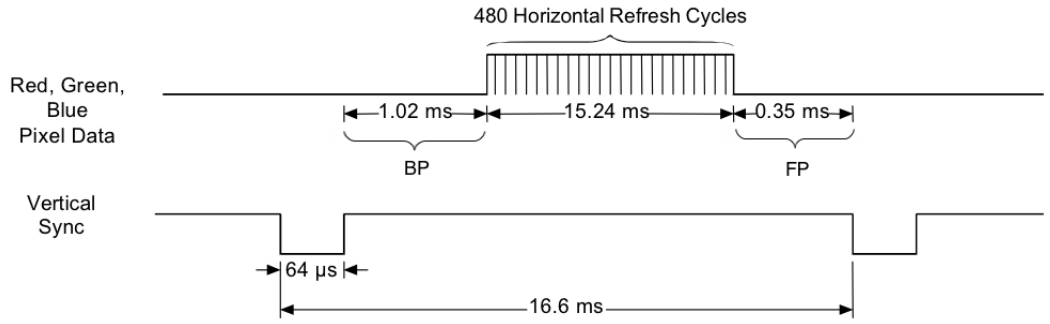

Figure 3.7: Vertical Sync Signal Timing for 640 by 480 at 60Hz [19]

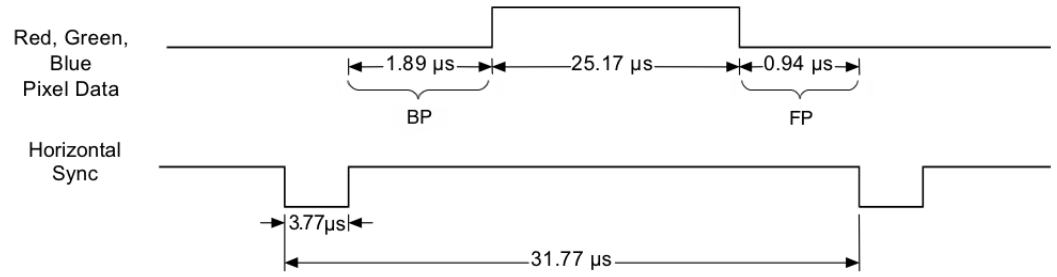

Figure 3.8: Horizontal Sync Signal Timing for 640 by 480 at 60Hz [19]

After 480 rows of pixels are refreshed with 480 horizontal sync signals, a vertical sync signal resets the monitor to the upper left corner and the process continues [19]. The time when pixel data is not being displayed is called Back Porch and Front Porch. During this time, the beam is returning to the left column to start another horizontal scan and the RGB signals should all be set to the color black. This means all bits set to zero.

Many VGA monitors will shut down if the two sync signals are not the correct values [19]. Most PC monitors have an LED that is green when it detects valid sync signals and yellow when it does not lock in with the sync signals. Modern monitors will sync up to an almost continuous range of refresh rates up to their design maximum. In a PC graphics card, a dedicated video memory location is used to store the color value of every pixel in the display. This memory is read out as the beam scans across the screen to produce the RGB signals. There is not enough memory storage inside the current generation of FPGA chips, so for this approach, the FPGA must use external memory [19].

3.5.3 ADV7123 VGA-DAC

The DE2 board includes a 16-pin D-SUB connector for VGA output [3]. The VGA synchronization signals are provided directly from the Cyclone II FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC is used to produce the analog data signals red, green, and blue. Figure 3.9 shows the associated schematic for the VGA-DAC.

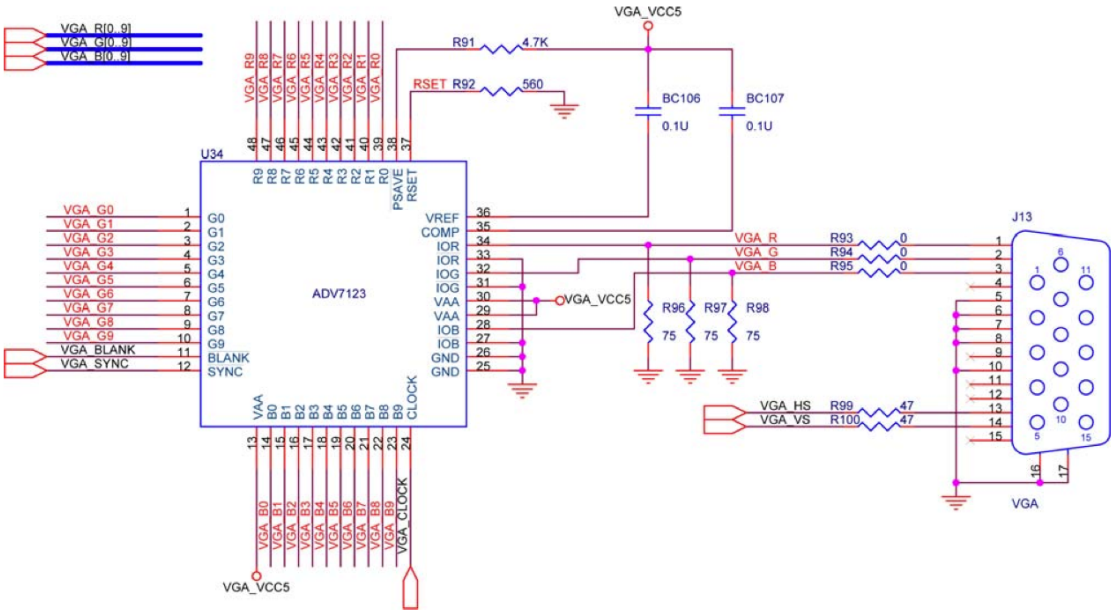

Figure 3.9: VGA circuit schematic [3]

The VGA-DAC supports multiple screen resolutions and refresh rates. The tables in Figure 3.10 and Figure 3.11, different screen resolutions, refresh rates and their horizontal or vertical timing specifications are shown. Table 3.1 explains the meaning of the letters a, b, c and d in these two figures.

Character	*Explanation*
a	Horizontal or Vertical sync
b	Back porch
c	Display interval
d	Front porch

Table 3.1: Explanation for letters in Figure 3.10 and Figure 3.11

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)

Figure 3.10: VGA horizontal timing specification [3]

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

Figure 3.11: VGA vertical timing specification [3]

3.6 Memory

When the FPGA design uses external memory, it is common to use flash memory for permanently storage [14]. If the system needs a faster program memory, SRAM or SDRAM are commonly used.

3.6.1 IS42S16400 SDRAM

Synchronous Dynamic Random Access Memory (SDRAM) can be used as temporary memory for a system [14]. SDRAM is a volatile memory, meaning that the SDRAM lose its contents when the system loses power. The DRAM technologies uses arrays of what are essentially capacitors to hold individual bits of data. The capacitor arrays will hold their charge only for a short period before it begins to diminish. Thus, The SDRAM need continuous refreshing, every few milliseconds or so.

The SDRAM chip on the Altera DE2 board has the capacity of 64 Mbits, or 8 Mbytes. It is organized as 1M x 16 bits x 4 banks [18]. The signals needed to communicate with this chip are shown in Figure 3.12. All of the signals, except the clock, can be provided by an SDRAM controller that may be generated through the SOPC builder. The clock signal has to meet the clock-skew requirements of the SDRAM, and can be provided by a PLL.

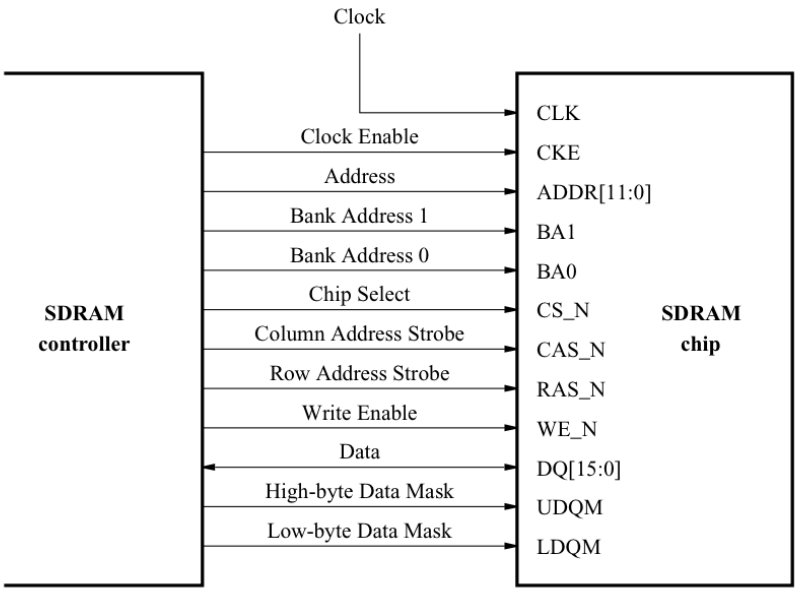

Figure 3.12: The SDRAM Signals [5]

3.6.2 S29AL032D Flash Memory

Flash is a nonvolatile memory, meaning that it do not require any power to retain its content [14]. The primary purpose for flash is to hold system code and other data the system needs to be present at power-up. The reason why this technology is not used as program memory, is because it is generally slower than RAM.

The Altera DE2 board contains a flash memory that may store up to 4 Mbytes of data, and is organized as 4M x 8 bits [32]. The signals needed for communicating with the flash memory chip can be generated by the SOPC builder. A figure of this communication interface is shown in Figure 3.13.

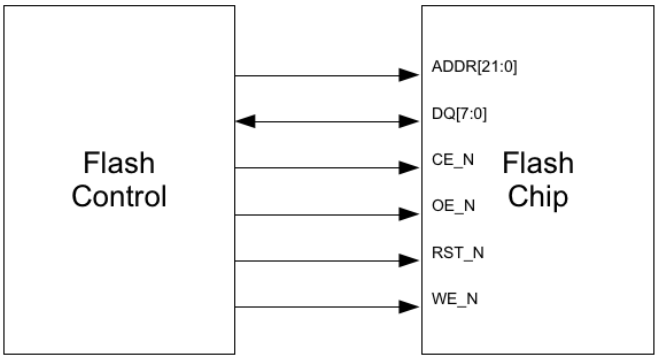

Figure 3.13: The Flash Memory Signals [32]

HAL, as described in Section 2.2, provides a simple interface for reading and writing to the flash memory[7]. With these methods it is possible to read and write bytes anywhere in the flash memory. In addition, Altera provides a read-only zip file system also for use with the HAL. The read-only zip file system provides access to a simple file system stored in flash memory. The flash drivers take advantage of the HAL generic device driver framework for file subsystems. Therefore, it is possible to access the zip file subsystem using the ANSI C standard library I/O functions, such as fopen() and fread().

3.6.3 Frame Buffer

Almost all graphic systems are raster based [10]. A picture is produced as an array of picture elements, or pixels, within the graphic system. Collectively, the pixels are stored in a part of memory called the frame buffer. The frame buffer can be viewed as the core element of a graphic system. The resolution of the frame buffer, that is the number of pixels in the frame buffer, determines the detail that it is possible to see in the image. The depth, or precision, of the frame buffer, defined as the number of bits that are used for each pixel, determines properties such as how many colors can be represented on a given system. For example, a 1-bit deep frame buffer allows only two colors, whereas an

8-bit deep frame buffer allows 2^8, or 256, colors. In full-color systems, there are at least 24 bits per pixel. Such systems can display sufficient colors to represent most images realistically. They are also called RGB-color systems, because individual groups of bits in each pixel are assigned to each of the tree primary colors; red, green and blue, used in most displays [10].

The frame buffer is usually implemented with special types of memory chips that enables fast redisplay of the contents of the frame buffer [10]. Pixel memory must always be in read mode whenever RGB data is displayed [19]. To avoid flicker and memory access conflicts on single port memory, designs should update pixel RAM and other signals that produces the RGB output, during the time the RGB data is not being displayed. When the scan of each horizontal line is complete, there are typically over 100 clock cycles before the next RGB value is needed, as seen in Figure 3.14. Additional clocks are available when a vertical sync signal resets the monitor to the first display line. The exact number of clocks available depends on the video resolution and refresh rate. In most cases, calculations that change the video image should be performed during this off-screen period of time to avoid memory conflicts with the readout of video RAM or other registers which are used to produce the RGB video pixel color signals [19].

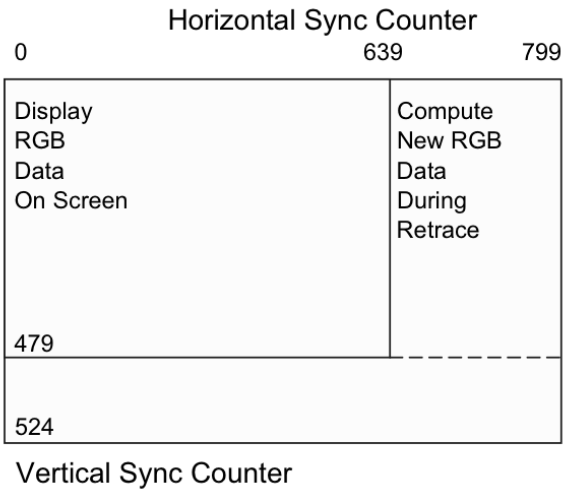

Figure 3.14: Display and Compute clock cycles available in a 640 by 480 Video Frame [19]

Chapter 4

Atmel AVR STK600 and Domain Conversion

In this chapter the Atmel AVR STK600 and the microcontroller AT90USB1287 are presented, in addition to theory of domain conversion for analog and digital signals.

4.1 Features of the STK600

The Atmel AVR STK600 is a complete starter kit and development system for the AVR and AVR32 flash microprocessors from ATMEL Corporation [11]. It is designed to give designers a quick start to develop code on the AVR, combined with advanced features for using the starter kit to prototype and test new designs. The STK600 is programmed using USB communication to PC, in addition, it supports communication interfaces for data exchange, such as CAN, RS232 and USB-client. The developments board also includes buttons and LEDs as user interface. The STK600 provides the opportunity to use most of the AVR microprocessors through different routing cards and socket boards. Figure 4.1 shows a picture of the different modules and how they are placed according to each other [11].

4.2 AVR AT90USB1278 Microcontroller

The Atmel AVR AT90USB1287 is an 8-bit microprocessor based on RISC architecture [12]. The microcontroller provides features like ten PWM channels, one Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART), an 8-channels 10-bit ADC, master and slave SPI, and USB. The microcontroller does also support enabling of interrupt on communication interfaces and I/O pins. Interrupt is described in Section 3.3. Furthermore, both USB and SPI will be presented in Chapter 5

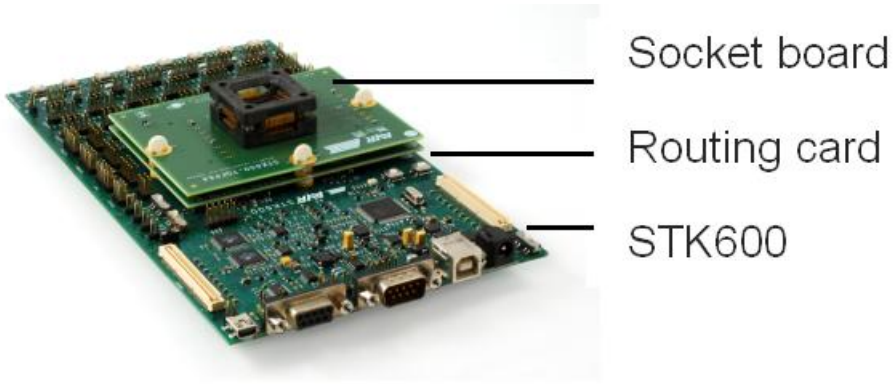

Figure 4.1: STK600 socket and routing system [11]

4.2.1 Analog to Digital Conversion

Most signals of practical interest, such as speech, radar signals, sonar signals and various communications signals such as audio and video signals, are analog [29]. To process analog signals by digital means, it is first necessary to convert them into digital form, that is, to convert them to a sequence of numbers having finite precision. This procedure is called analog-to-digital-, or A/D-conversion, and the corresponding devices are called ADCs. The A/D conversion can be described in a three-step process. Figure 4.2 illustrates this process [29].

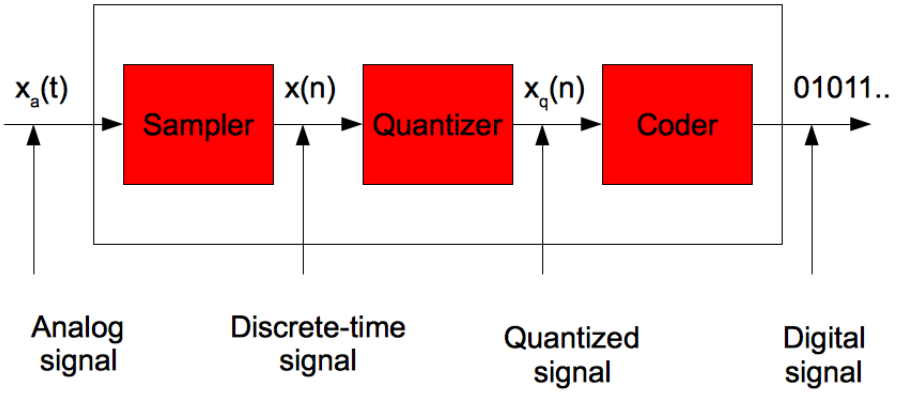

Figure 4.2: Basic parts of an analog-to-digital converter [29]

1. *Sampling:* This is the conversion of a continuous time signal into a discrete time signal obtained by taking samples of the continuous time signal at discrete time instants. Thus, if $x_a(t)$ is the input to the sampler, the output is $x_a(nT) \equiv x(n)$, Where T is called the sampling interval.

2. *Quantization:* This is the conversion of a discrete time continuous valued signal

into a discrete time, discrete-valued, digital signal. The value of each signal sample is represented by a value selected from a finite set of possible values. The difference between the unquantized sample *x(n)* and the quantized output $x_q(n)$ is called the quantization error.

3. *Coding:* In the coding process, each discrete value $x_q(n)$ is represented by a bit binary sequence.

Although we model the ADCs as a sampler followed by a quantizer and coder, in practice the A/D conversion is performed by a single device that takes $x_a(t)$ and produces a binary-coded number [29]. The operations of sampling and quantization can be performed in either order, but in practice, sampling is always performed before quantization.

4.2.2 Digital to Analog Conversion

In many cases of practical interest, for example speech processing, it is desirable to convert the processed digital signals into analog form [29]. The process of converting a digital signal into the analog domain is known as digital to analog conversion, D/A Conversion. All digital to analog converters (DACs), "connects the dots" in a digital signal by performing some kind of interpolation, whose accuracy depends on the quality of the D/A conversion process. Figure 4.3 illustrates a simple form of D/A conversion, called a zero-order hold or a staircase approximation [29]. The original signal is the input signal to ADC, and the staircase approximation is the output of the same signal after both an AD- and DA-convertsion. Before the staircase approximated signal is used, it is sent through a filter to smooth up the edges.

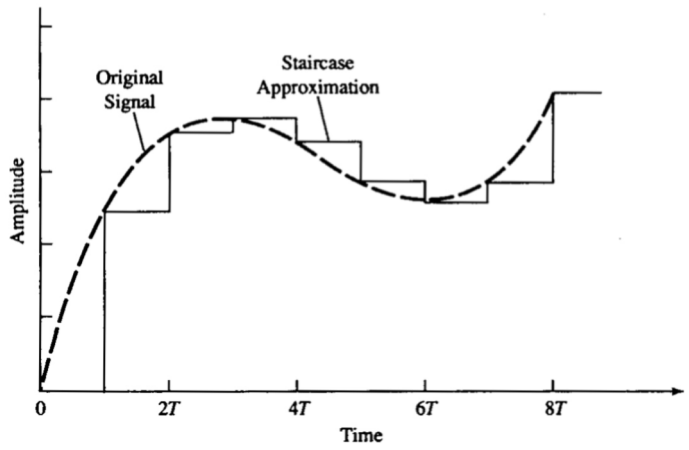

Figure 4.3: Zero-order hold D/A conversion [29]

4.2.3 Pulse Width Modulation

Using a DAC may seem the obvious way to generate an analog output voltage from a digital system, but there is another way that uses noting more than a digital I/O line configured as an output [14]. This technique is known as Pulse Width Modulation (PWM).

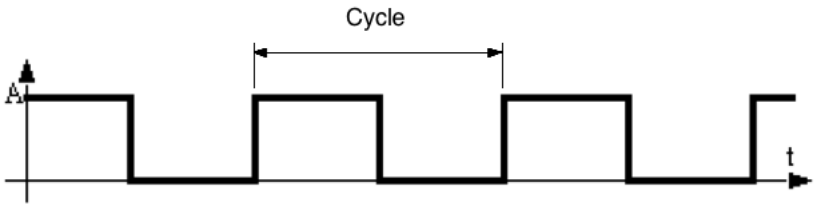

Figure 4.4: 50% duty cycle [14]

Consider the average square wave shown in Figure 4.4. The width of the logic high is equal to the width of the logic low, so this wave is said to have a 50% duty cycle. In other words, it is high for exactely half the cycle. If the amplitude A of this square wave is 5V, the average voltage over the cycle is 2,5V [14].

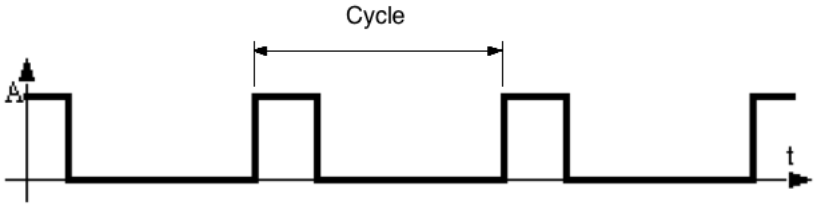

Figure 4.5: 25% duty cycle [14]

Now consider the square wave in Figure 4.5. This wave has a 25% duty cycle, which means that the average voltage over the cycle is 1,25V. An averaging filter, like a low-pass filter, on the PWM output will convert the pulses to an analog voltage, proportional to the duty cycle of the PWM signal. By varying the duty cycle it is possible to vary the analog voltage [14].

Chapter 5

Communication Interface

A communication interface is included in both the Altera DE2 and the Atmel AVR STK600. Thus, the theory for this implementation is presented in this chapter.

5.1 Serial Peripheral Interface

SPI, or Serial Peripheral Interface, is a synchronous protocol in which all transmissions are referenced to a common clock generated by the master processor [14]. SPI is used for data storage in memory, in peripherals like ADCs and DACs, and even between processors. The receiving peripheral, or slave, uses the clock to synchronize incoming bits of the serial bit stream, thus the data rate is decided by the master clock. Many chips may be connected to the same SPI interface of a master. A master selects a slave to receive by asserting the slave's chip select input. A peripheral that is not selected will not take part in a SPI transfer.

Signal	**Description**
MOSI	Master Out Slave In
MISO	Master In Slave Out
SCLK or **SCK**	Serial Clock
$\overline{\textbf{CS}}$ or $\overline{\textbf{SS}}$	Chip Select or Slave Select

Table 5.1: SPI signals [14]

In Table 5.1, the four main SPI signals are shown [14]. MOSI is generated by the master and is received by the slave. MISO is produced by the slave, but its generation is controlled by the master. The chip select to the peripheral is normally generated by simply using a spare pin on the master. Figure 5.1 shows a microprocessor interfaced to a peripheral using SPI.

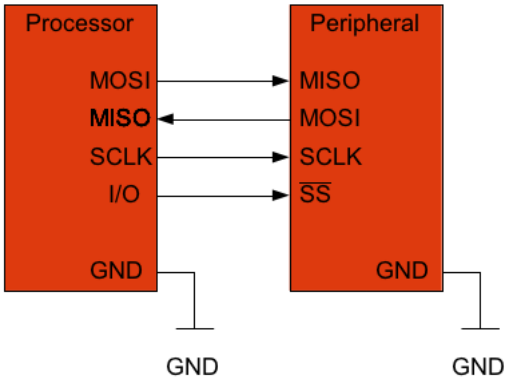

Figure 5.1: Basic SPI interface [14]

Both master and slave contains a serial shift register [14]. The master starts a transfer of a byte by writing it to its SPI shift register. As the register transmits the byte to the slave on the MOSI signal line, the slave transfers the contents of its shift register back to the master on the MISO signal line. In this way, the contents of the two shift registers are exchanged. Both a write and a read operation are performed simultaneously. Figure 5.2 illustrates this process.

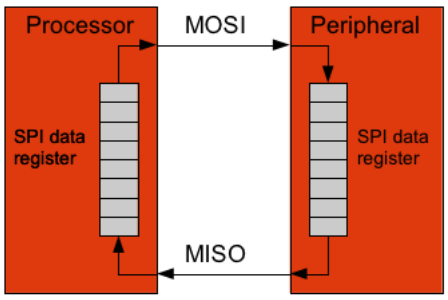

Figure 5.2: SPI transmission [14]

If only a write operation is desired, the master just ignores the byte it receives [14]. On the other hand, if the master wishes to read from the slave, it must transfer a dummy byte to initiate a slave transmission.

5.2 Universal Serial Bus

Universal serial bus, or USB, is a high speed bus that allows up to 127 peripherals to be connected to a host, at a data rate of 480Mbps [14]. USB allows peripherals and computers to interconnect in a standard way with a standard protocol. In addition,

USB can provide power to the peripherals connected to the host through the same cable as they use for communication. Since peripherals are able to self-identify to a host computer, the USB is popular among devices like printers, cameras, mice and keyboards. However, this is also why the developing with USB is more complex than with for example SPI. USB add an extra layer of complexity to the software, since the system must interact with the host in the appropriate way.

Signal	*Purpose*
V_{BUS}	USB device power(+5V)
D+	Differential data line
D-	Differential data line
GND	Power and signal ground

Table 5.2: USB Signals [14]

The USB standard uses a shielded, four-wire cable to interconnect devices on the network [14]. As shown in Table 5.2, the data transmission is accomplished over a differential twisted pair D+ and D-. The other two wires are V_{BUS}, which carries power to USB devices, and GND.

Chapter 6

Electronic Components

In this chapter the basic electronic components; resistor, transistor, LED and LDR are presented. In addition the Ohm's law and voltage division is described.

6.1 Resistor

Resistance is the capacity of materials to impede the flow of current or, more specifically, the flow of electric charge [26]. The circuit element used to model this behavior is the resistor, and it is measured in ohm (Ω).

6.1.1 Ohm's law

For purposes of circuit analysis, it is important to reference the current in the resistor to a terminal voltage [26]. This can be done in two ways, either in the direction of the voltage drop across the resistor or in the direction of the voltage rise across the resistor. If the first method is chosen the relationship between the voltage and current is shown in Equation (6.1), and the second is shown in Equation (6.2).

$$v = i \cdot R, \tag{6.1}$$

$$v = -i \cdot R, \tag{6.2}$$

In these equations v represents voltage measured in volts, i represents current measured in amperes and R represents resistance measured in Ω. These equations are called Ohm's law [26].

6.1.2 Voltage divider

When voltage is divided between series resistors, as shown in Figure 6.1, the voltage across each resistor can be found according to the Equation (6.3) and (6.4) [26].

$$v_1 = \frac{R_1}{R_1 + R_2} \cdot v_s \tag{6.3}$$

$$v_2 = \frac{R_2}{R_1 + R_2} \cdot v_s \tag{6.4}$$

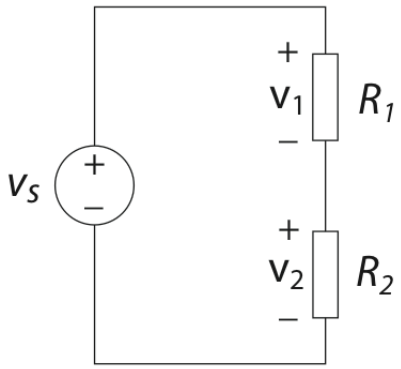

Figure 6.1: Voltage divider [26]

6.2 Transistor

The MOSFET transistor has three signal terminals: Gate (G), drain (D) and source (S), plus the bulk terminal (B), to which the gate, drain and source voltages are referenced [30]. In Figure 6.2 the three signal terminals are shown.

The simplest view of MOSFET logic operation treats the transistor as a switch. The gate terminal is analogous to the light switch on the wall. When the gate has a sufficient high voltage, the transistor closes like a switch, and the drain and source terminals are electrically connected. Just as a light switch requires a certain force level to activate it, the transistor needs a certain voltage level to connect the drain and source terminals. This voltage is called the transistor threshold voltage V_t, and is a fixed voltage.

Transistors act as switches with two conducting states, on and off, depending on the control (Gate) terminal voltage. An ideal transistor has a zero ohm resistance between the drain and source when it is in the on-state, and an infinite resistance between these terminals in the off-state [30].

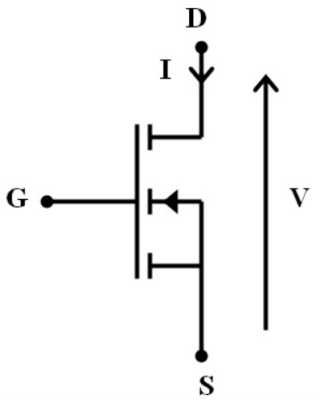

Figure 6.2: MOSFET transistor with signal terminals [30]

6.3 Light Dependent Resistor

A light dependent resistor (LDR) consists of a disc of semiconductor material with two electrodes on its surface [13].

When dark or in dim light, the material of the disc contains relatively few free electrons, thus there are few electrons to carry electric charge. This means that it is a poor conductor of electric current, thus its resistance is high. In contrast, with more light, a larger number of electrons escape from the atoms of the semiconductor. Thus there are more electrons to carry electric charge and it becomes a good conductor, which means its resistance is low. The more light, the more electrons, the lower resistance [13].

6.4 Light Emitting Diode

Light emitting diodes, usually known as LEDs, provideslight when a current flows through them [13]. A typical LED is in a domed plastic package with a rim. There are two terminal wires at the bottom, and usually the cathode wire is shorter than the anode wire.

An LED needs about 20 mA to be lit to its full brightness, though as little as 5 mA still produces a clearly visible glow. The forward voltage drop of an LED averages about 1.5V, so a 2 V supply will light most types to their maximum brightness. When lit by a higher voltage, the LED may be burned out if the forward voltage across it exceeds 2V. It is thus essential to wire a current limiting resistor in series with it[13].

Chapter 7

Numerical Systems

A numbering system is a mechanism used for representing numeric values. In today's society, people most often use the decimal numbering system while most computers uses binary representation. In this chapter, the general positional number system is explained. Further, the hexadecimal and binary number systems are introduced together with an introduction to the conversion between them.

7.1 Positional Number Systems

To understand the binary number system used in computer design, it is important to recognize that the number system used in our every day life is a positional number system [22]. In such a system, any number is represented by a string of digits in which the position of each digit has an associated weight. The value of a given number, then, is equivalent to the weighted sum of all its digits. An example of this is shown in Equation (7.1).

$$1234 = 1 \cdot 1000 + 2 \cdot 100 + 3 \cdot 10 + 4 \cdot 1 \tag{7.1}$$

In this example, each weight is a power of 10 that is equal to 10^i, where i corresponds to the digit's position counting from the right [22]. In general, any decimal number D of the form $d_3d_2d_1d_0$ has the value $D = d_3 \times 10^3 + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$. Here, 10 is called the radix of the number system. In a general positional, the radix may be any integer r, and a digit in position i then has the weight r^i. Thus, a general system can be described with Equation (7.2) where m is the total of digits. Equation (7.3) shows that the value of this number is the sum of products of each digit multiplied by the corresponding power of the radix.

$$d_{m-1}d_{m-2} \cdots d_1d_0 \tag{7.2}$$

$$D = \sum_{0}^{m-1} d_i \cdot r^i \tag{7.3}$$

In positional number systems the leftmost digit is called the most-significant digit (MSD) and the rightmost is the least-significant digit (LSD).

7.2 The Binary Number System

Since digital systems use binary digits, binary radix is used to represent any given number in a digital system [22][20]. Equation (7.4) shows the general form of such a binary number, and Equation (7.5) represent the equivalent to its value.

$$b_{m-1}b_{m-2}\cdots b_1 b_0 \tag{7.4}$$

$$B = \sum_{0}^{m-1} b_i \cdot 2^i \tag{7.5}$$

When working with other non-decimal numbers, it is common to use a subscript to indicate the radix of each number, although the radix is often clear from the context [22][15]. Equation (7.6) and Equation (7.7) shows some examples of binary numbers and their decimal equivalents.

$$10101_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 21_{10} \tag{7.6}$$

$$110101_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 53_{10} \tag{7.7}$$

7.3 The Hexadecimal Number System

The hexadecimal number system uses radix 16, and is often used to provide convenient shorthand representations for binary numbers, reducing the need for long strings of binary digits [22][15]. Since the hexadecimal number system needs to express sixteen different values, it uses the digits 0 through 9 and supplements with the letters A through F.

The rules for positional number systems also applies for the hexadecimal number system. Equation (7.8) shows an example of hexadecimal numbers and their decimal equivalent.

$$12EF_{16} = 1 \cdot 16^3 + 2 \cdot 16^2 + 14 \cdot 16^1 + 15 \cdot 16^0 = 4847_{10} \tag{7.8}$$

In Table 7.1, the hexadecimal numbers from 0 through 12 with there decimal and binary equivalents are shown [22].

Decimal	*Binary*	*Hexadecimal*
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12

Table 7.1: Different representation of numbers 0 through 18

7.4 Conversion Between Number Systems

Different number systems for different occasions are useful. But before a computer presents a decimal number or a designer programs low-level code, they need to know how to convert between the different number systems.

7.4.1 Conversion From Binary

It is very easy to convert a binary number to a hexadecimal form [22][15]. Starting at the LSB and working left with separating bits in groups of 4 and replace each group with the corresponding hexadecimal digit in Table 7.1. An example of this is shown in Equation (7.9). In this example, zeroes are added to the left to make the total number of bits a multiple of 4.

$$1010011100_2 = 0010\,1001\,1100 = 29C_{16} \tag{7.9}$$

Conversion from binary to decimal was introduced in Equation (7.6) and Equation (7.7), and these equations are derived from Equation (7.5) with $m = 4$ and $m = 5$ respectively [22][20]. As seen in the equations each digit is multiplied with the corresponding power of the radix.

7.4.2 Conversion From Hexdecimal

The conversion between hexadecimal and binary is as easy as the conversion from binary to hexadecimal. Each hexadecimal digit are replaced by the corresponding 4 bits in Table 7.1 [22]. Equation (7.10) shows an example of this conversion.

$$7E5D_{16} = 0111\,1110\,0101\,1101 = 111111001011101_2 \tag{7.10}$$

The progress to obtain the decimal value from a hexadecimal number, is the same as for the binary number [22][15]. The Equation (7.3) can modified to Equation (7.11), where r is the hexadecimal radix 16, and m indicates the number of digits. The example in Equation (7.8) is derived from Equation (7.11).

$$D = \sum_{0}^{m-1} h_i \cdot 16^i \tag{7.11}$$

7.4.3 Conversion From Decimal

Equation (7.3) can be rewritten as Equation (7.12). This can be used when converting a decimal number D to a number in radix r [22][20].

$$D = ((\cdots((d_{m-1})r + d_{m-2})r + \cdots)r + d_1)r + d_0 \tag{7.12}$$

If Equation (7.12) is divided by r, the parenthesized part of Equation (7.12) represents the quotient in Equation (7.13) while the rest is the remainder shown in Equation (7.14) [22].

$$Q = (\cdots((d_{m-1})r + d_{m-2})r + \cdots)r + d_1 \tag{7.13}$$

$$R = d_0 \tag{7.14}$$

In other words, d_0 is obtained as a remainder of the division of D by r [22]. Furthermore, since the quotient Q in Equation (7.13) has the same form as the original number, it is known that successive division by r yields successive digit of D from right to left until all

the digits of D have been derived. Equation (7.15) and Equation (7.16) show an example how conversion between decimal and binary, and decimal and hexadecimal respectively are done.

$$\begin{aligned}
179 \div 2 &= 89 && remainder\ 1\ (LSB) \\
89 \div 2 &= 44 && remainder\ 1 \\
44 \div 2 &= 22 && remainder\ 0 \\
22 \div 2 &= 11 && remainder\ 0 \\
11 \div 2 &= 5 && remainder\ 1 \\
5 \div 2 &= 2 && remainder\ 1 \\
2 \div 2 &= 1 && remainder\ 0 \\
1 \div 2 &= 0 && remainder\ 1\ (MSB) \\
Thus,\ 179_{10} &= 10110011_2
\end{aligned} \tag{7.15}$$

$$\begin{aligned}
3417 \div 16 &= 213 && remainder\ 9\ (LSB) \\
213 \div 16 &= 13 && remainder\ 5 \\
13 \div 16 &= 0 && remainder\ 13\ (MSB) \\
Thus,\ 3417_{10} &= D59_{16}
\end{aligned} \tag{7.16}$$

Chapter 8

System Modules and Scenarios

This chapter presents an overview of the system and the two example scenarios. The supporting theory of the development boards, namely the Altera DE2 and the Atmel AVR STK600 may be found in Chapter 3 and Chapter 4 respectively. Furthermore, the scenario theory can be found in Chapter 6 and Chapter 7.

8.1 Overview of System

The purpose for this embedded demonstrator is to teach basic circuit theory and demonstrate how a selection of simple circuits work. The parameters; area, power and performance have no restriction in this system except the limitations of resources on the Altera Development and Education Board and the microprocessor on the Atmel AVR STK600. The demonstrator includes different interactive scenarios that the user may choose between through a user interface. Each scenario has a topic that it tries to convey through the system interface.

As seen in Figure 8.1, this system consists of three modules; a monitor, an Altera DE2 and an Atmel STK600. The purpose for the monitor is to show information about the current scenario, and at the same time display images of virtual circuits. In these scenarios the voltages and currents in the virtual circuit are changing depending on how the user operates the system. The monitor is connected to a VGA-DAC onboard the Altera DE2.

As mentioned in Chapter 2, both the hardware and software were chosen based on their availability at the department, and the design tools already familiar. However, a dedicated circuit board was thought of as an alternative to the Atmel STK600. The implementation of this dedicated board would have to be produced from scratch, thus the implementation of the Atmel STK600 was both easier and faster.

The Altera DE2 is the master communication device that controls both the monitor

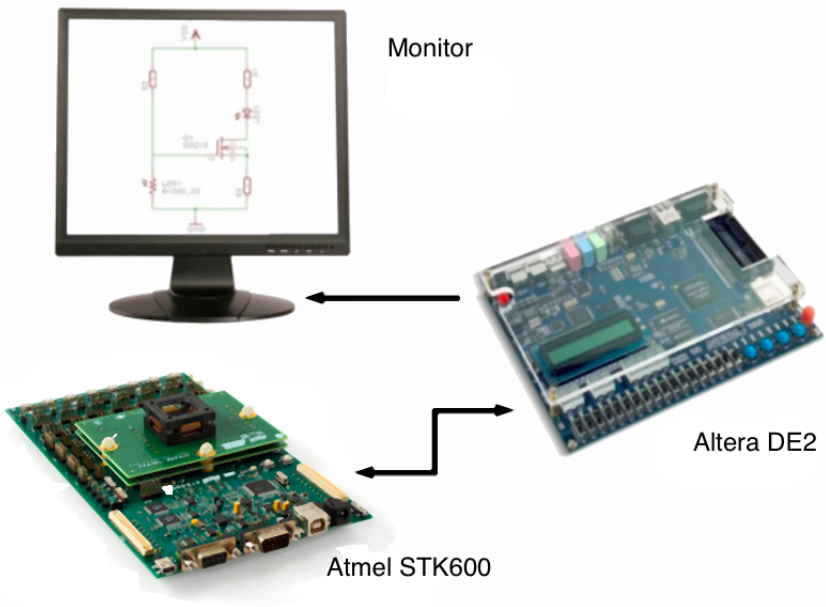

Figure 8.1: System overview

and the STK600. In addition, it includes a user interface and a memory-controller that controls a frame buffer.

The purpose for the last module, the Atmel STK 600, is reading and writing to analog electrical components using ADC and PWM. These electrical components are used in different scenarios, and are chosen from the Altera Development and Education Board. This module also includes a user interface which is controlled from the microprocessor mounted on the STK 600.

8.2 Overview of Scenarios

The system is designed as a platform for easy implementation of scenarios. Through the use of methods in both hardware and software, a developer can control both the interface and the behavior of the system. This section describes two, already implemented, scenarios that uses different modules in the system.

8.2.1 Automatic Adjustment of Light

One of the scenarios implemented is the automatic adjustment of light. The topics for this scenario are Ohm's law and voltage dividing presented in Chapter 6, in addition to the basic functionality of a transistor, a LDR and a LED.

In Figure 8.2, the virtual circuit of this scenario is shown. This circuit along with supporting theory are displayed on the system monitor. Depending on the amount of light surrounding the LDR, the brightness of the LED will vary. If the light intensity on the LDR increases, the brightness of the LED will decrease. On the other hand, if the light intensity on the LDR decreases, the brightness of the LED will increase. The analog circuits do not exist in the system, it is the soft processor, the Nios II, that simulates both the voltages and currents in the virtual circuits. The LDR and LED are controlled by the ADC and PWM respectively, implemented on Atmel's STK600.

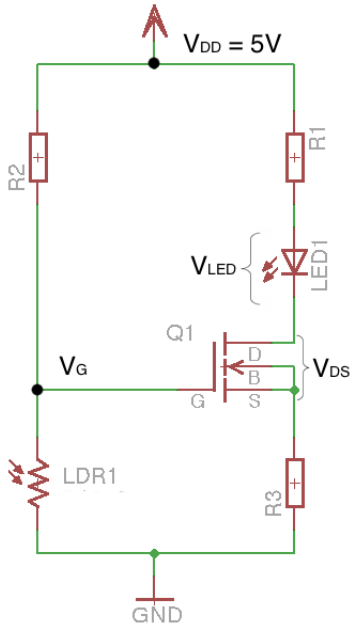

Figure 8.2: Automatic adjustment of light

In a real analog circuit, the reduction of resistance caused by an increase in brightness, and the voltage dividing between R2 and LDR, would decrease the voltage between ground and gate, V_G, on the transistor Q1. Thus, the voltage between gate and source, V_{GS}, decreases. This causes the voltage between drain and source, V_{DS}, to increase, thus the voltage over LED will decrease and the light will fade out. On the other hand, if the light intensity on the LDR increases, the V_G, and V_{GS}, will increase. This causes the V_{DS} to decrease, and the voltage over the LED will increase and thus its light intensity, will increase.

This scenario is considered to be a good demonstrator due to its ability to demonstrate how a relatively simple electrical circuit provides a functionality present in many well known electronic devices. In mobile phones, a circuit like this may be used to turn the LCD-display of the device off when the user holds the phone to his or her ear. Thus, the Automatic Adjustment of Light scenario provides a good demonstration of topics related to the user's everyday life.

8.2.2 How to Count in Binary, Hexadecimal and Decimal

The topic for this scenario is numerical systems. It demonstrates and presents the differences between the most ordinary numerical system used in everyday life, the decimal system, and the two numerical systems; binary and hexadecimal used in the digital world. The supported theory for this scenario may be found in Chapter 7

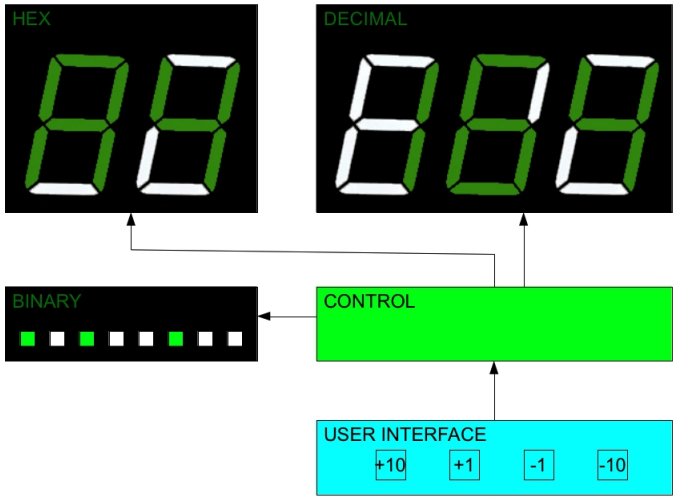

Figure 8.3: How to Count in Binary, Hexadecimal and Decimal

In Figure 8.3 a block diagram of the scenario is shown. The user interface for this module has four buttons, +10, +1, -1 and -10, where each of the buttons interact with the three display-modules; HEX, DECIMAL and BINARY. When the user presses a button, the control unit adds or subtracts one or ten in the decimal system from the three displays, depending on which of the buttons the user presses. By default, and when the system resets, the value on the displays are set to zero. The monitor in this scenario, shows the different methods to convert between the number systems.

This scenario provides the user with the ability to play with different number systems presented on the DE2 board. The demonstration could include some simple tasks to be solved with pen and paper, where the monitor in the system, together with a set of 7-segment displays, would provide the user with support on the theory needed to complete the calculations. In this way, the pupils or students who uses the demonstrator may learn the difference between the different number systems, in addition to the conversion operations between them.

Chapter 9

System Implementation and Discussion

In this chapter, the implementation of the block diagram shown in Figure 9.1 is discussed. The block diagram shows the Altera DE2, STK600 and their modules used for the realization of this embedded demontrator.

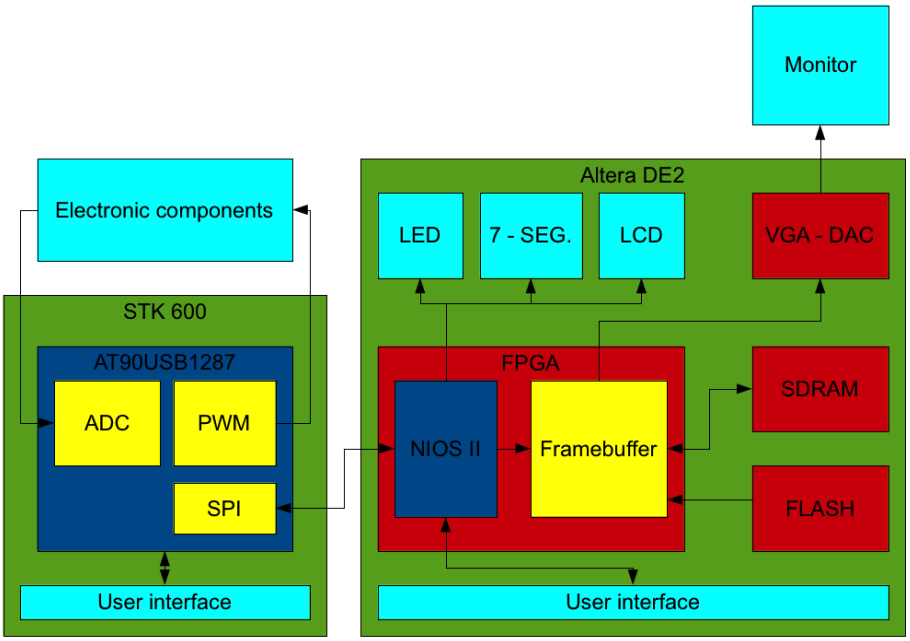

Figure 9.1: Block diagram of system

9.1 VGA-Controller and Frame Buffer

The system is designed to display scenario information on a screen through the VGA-DAC on Altera DE2. This information can be both pictures or theory about the current scenario. The RGB pixel data the VGA-DAC needs to display the current pixel is stored in a frame buffer that uses the SDRAM as a storage bank. The supporting theory can be found in Section 3.5 and Section 3.6.

9.1.1 VGA-Controller

The purpose for the VGA-controller in Figure 9.2 is to control the behavior of the VGA-DAC. The signals of the VGA-controller is presented in Table 9.1. The outputs in this table are the inputs to the VGA-DAC, except the sync signals *VGA-HS* and *VGA-VS*. These are connected directly to the VGA-connector together with the clock signal *VGA-CLK*. Both the VGA-DAC and the VGA-connector are shown in Figure 3.9 on page 18.

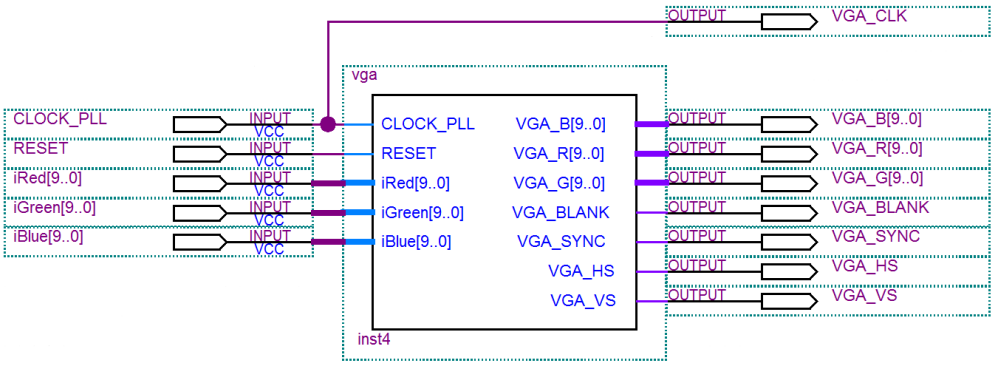

Figure 9.2: VGA-Controller Block

As explained in Section 3.5.2, the monitor needs signals that controls when the "painting" of a new row or a new image should begin. The VGA-controller solves this by including two counters for controlling the horizontal and the vertical timing. The vertical and horizontal controllers implement the behavior of Figure 3.7 and Figure 3.8 respectively. In the VGA controller initialization code, reference values for these counters are calculated from the timing parameters found in the tables in Figure 3.10 and Figure 3.11 on page 19. If not reset to zero by the *Reset* signal, the counters count to there reference values and then start counting from zero again. The choice of parameters from these tables decide the resolution and the refresh rate displayed on the connected monitor. The chosen resolution for the system is 640×480 with a refresh rate at $60MHz$. The reason for this choice is further explained in Section 9.1.2.

In Figure 9.3 a waveform simulation of an entire frame is shown. The figure shows that

Pin	Purpose
CLOCK-PLL	VGA pixel clock
RESET	Active low reset for VGA-controller
iRed	10-bits vector representing red color from frame buffer
iGreen	10-bits vector representing green color from frame buffer
iBlue	10-bits vector representing blue color from frame buffer
VGA-R	10-bits vector representing red color to VGA-DAC
VGA-G	10-bits vector representing green color to VGA-DAC
VGA-B	10-bits vector representing blue color to VGA-DAC
VGA-HS	Horizontal sync
VGA-VS	Vertical sync
VGA-BLANK	Not used. Active low. Used for setting screen color to black
VGA-SYNC	Not used. Active low. Too reduce power consumption

Table 9.1: In and Outputs for the VGA-Controller

the vertical sync signal, *VGA-VS*, has a short low period in both the beginning and at the end of the waveform. This waveform is a simulation of the behavior of Figure 3.7 on page 17. As shown in the waveform, horizontal lines are painted with RGB pixel data between these short low periods.

The waveform simulation shown in Figure 9.4 displays the simulation of one horizontal sync, or the painting of one horizontal line. This simulation shows the VGA-controllers implementation of the behavior in Figure 3.8. As seen in both figures, *vertical sync* and *VGA-HS* have the same behavior, and the RGB pixel data is zero in both ends of the horizontal line. In both the simulations shown in Figure 3.7 and Figure 3.8, it is not possible to see one single clock cycle. This is the reason why it looks like the RGB pixel data is set low at the same time as the sync signals *VGA-VS* and *VGA-HS* are set low.

When the electron beam is out of the displayed picture, the pixel data must be set to the color black as described in Section 3.5.2. This can be achieved by using the *VGA-BLANK* signal. When this signal is set to low, the VGA-DAC do not care about the RGB pixel data, it sets the output voltage to 0V peak-to-peak. Another solution is shown in Figure 9.4, and is used in this VGA-controller. The *video-on* signal in the red square determines whether the pixel data is displayed or not. As shown in the Figure 9.4, whenever *video-on* is high, the RGB pixel data is transmitted to the VGA-DAC. On the other hand, if it is low, the RGB pixel data is low too. The *video-on* signal is only high if both *video-on-h* and *video-on-v* are high. The latter two signals are controlled by the horizontal and vertical counters respectively. The last solution was chosen because it supports VGA-DACs without a blank function.

The clock frequency is found using the table shown in Figure 3.10, and the calculation is shown in Equation (9.1).

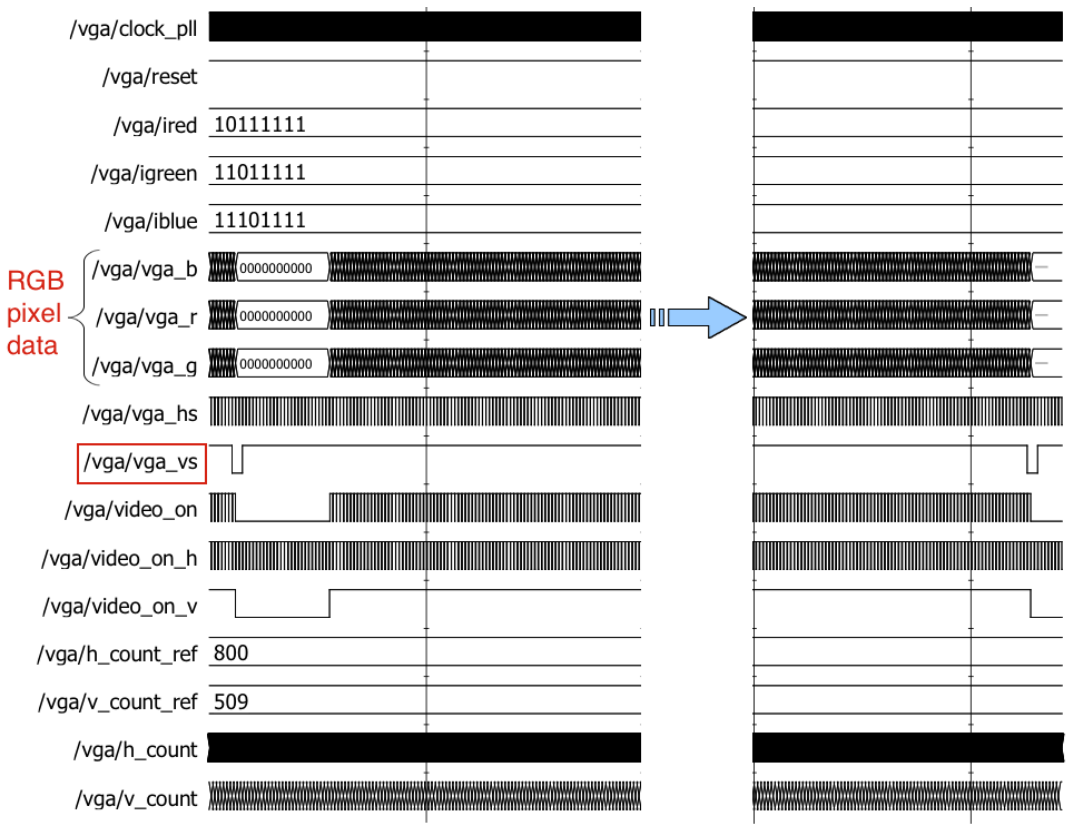

Figure 9.3: ModelSim Simulation of Vertical Sync Signal

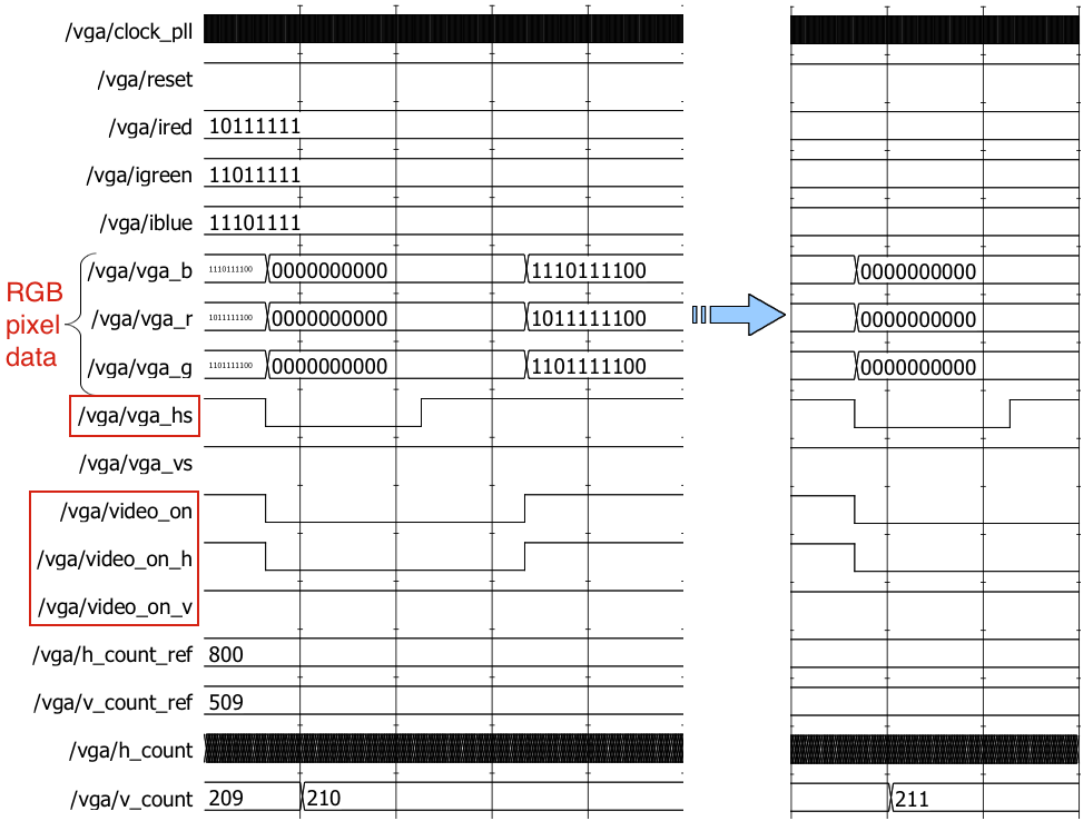

Figure 9.4: ModelSim Simulation of Horizontal Sync Signal

$$Pixel\ Clock = \frac{Horizontal\ resolution}{Display\ interval} = \frac{640}{25.4} = 25.196850\ MHz \tag{9.1}$$

The frequency is generated from a PLL designed with the MegaWizard. An alternative to MegaWizard is to generate the frequency with a PLL in the SOPC builder, but this was not considered because there was no Nios processor in the system at this time. The VHDL code for the VGA-controller can be found in Appendix A, and the supported theory about both SOPC and MegaWizard are described in Section 2.1. Finally, the Nios II is presented in Section 3.3.

9.1.2 Frame Buffer

The purpose for the frame buffer is to store the current frame shown on the monitor and to deliver the right RGB data for each pixel at the right time to the VGA-DAC. As mentioned in Section 3.2, the Cyclone II includes 105 $M4K \cdot 4608$ bits = 483840 bits of total memory. If one frame is 640×480 pixels, and the color is set to black and white, equal to a 1-bit deep frame buffer as explained in Section 3.6.3, the frame will occupy 307200 bits, or 63,5% of available memory in the FPGA. The additional frame buffer controllers and the logic for the rest of the system will thus have less than 40% of program memory.

To show more colors, the depth of the frame buffers must be increased, an operation that will cost over 100% of the FPGA memory usage. The solution to this problem is the SDRAM mounted on Altera DE2, described in Section 3.6.1. The SDRAM chip has a capacity of 8 MBytes. This is over a 100 times larger than the total memory of the FPGA. But SDRAM is a volatile memory, meaning that it loses its data when the system is turned off. This can be solved by another chip on the Altera DE2. As mentioned in Section 3.6.2, the flash chip may contain 4 MBytes and hold the data with or without power. On the other hand, this memory-technology is slower than the SDRAM technology, and is generally not used as frame buffer memory.

Either of the two scenarios, Automatic Adjustment of Light or How to Count in Binary, Hexadecimal and Decimal, described in Section 8.2, are dependent on a screen with a wide color spectrum, since none of them includes colorful scenario-pictures. However, this demonstrator is designed for adding several scenarios, thus a worthy color depth is appropriate. In Figure 9.5 and Figure 9.6 illustrates the visible specter with 16 and 24-bits per pixel respectively. As shown in these figures, the different spectrums look almost the same, except for that Figure 9.5 can represent $2^{16} = 65536$ different colors and Figure 9.6 can represent $2^{24} = 16777216$ different colors. Thus, some of the visible color spectrum are not represented in Figure 9.5. This is why Figure 9.5 have larger blocks of each color than Figure 9.6.

As mentioned, both the FPGA, the SDRAM and the flash have limitations in capacity. It is thus important to have these limitations in mind when choosing resolution and color

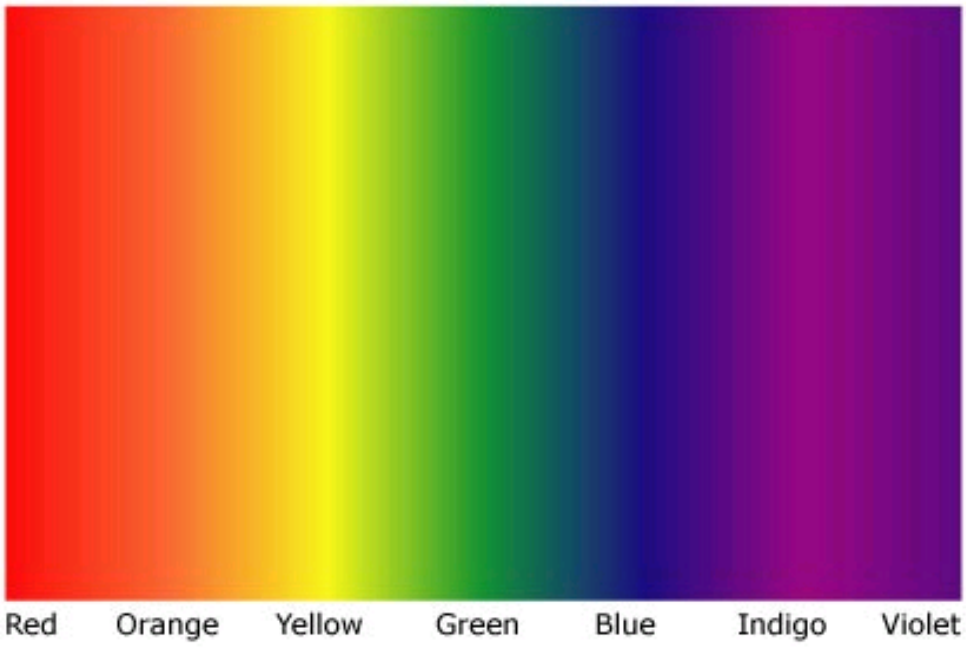

Figure 9.5: Visible Spectrum in 16-bit color depth

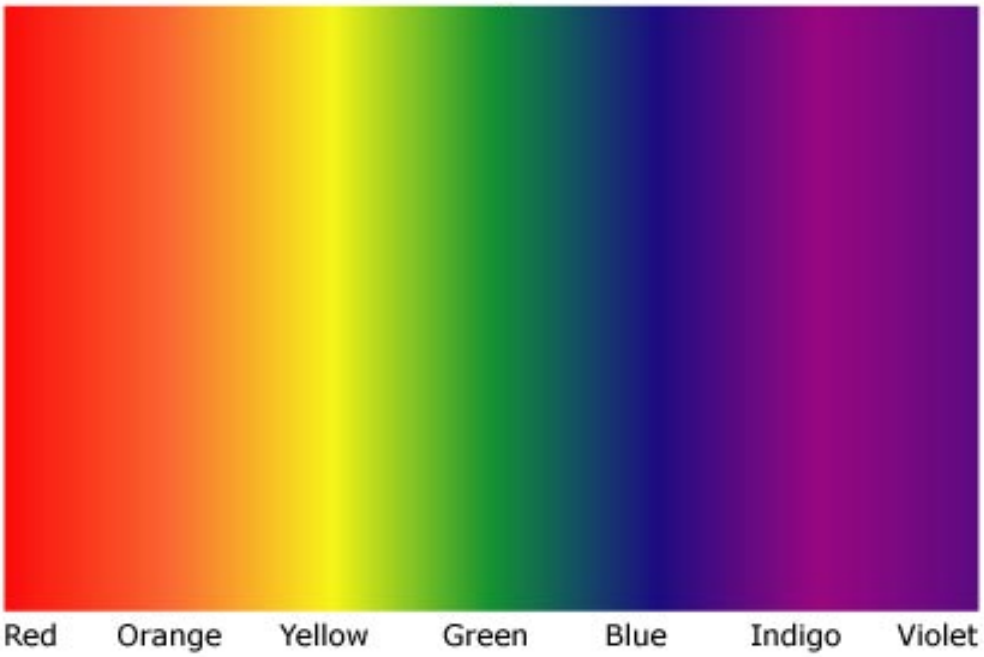

Figure 9.6: Visible Spectrum in 24-bit color depth

depth of a frame buffer. Table 9.2 presents different resolutions and depths of the frame buffers, expressed as bits per pixel (bpp), together with their respective memory usage and the total of frames in flash and SDRAM.

Resolution · *bpp*	**Mbits**	**MBytes**	**Frames in Flash**	**Frames in SDRAM**
$640 \times 480 \cdot 16$	4.9152	0.6144	6	13
$800 \times 600 \cdot 16$	7.68	0.96	4	8
$1024 \times 768 \cdot 16$	12.582912	1.572864	2	5
$640 \times 480 \cdot 24$	7.3728	0.9216	4	8
$800 \times 600 \cdot 24$	11.52	1.44	2	5
$1024 \times 768 \cdot 24$	18.874368	2.359296	1	3

Table 9.2: Resolution and Color Depth versus Memory Usage

With respect to the limitation of capacity and the desire of a true color demonstrator, the 640×480 resolution with 24 bpp was chosen. Although the same resolution with 16 bpp results in two more frames in flash, as seen in Table 9.2, it will be more preferable to include scenarios with a wider color spectrum. In addition, Section 3.6.3 describes that a 24-bits RGB-color system uses 8-bits to represent each color, thus with this solution it is easier to read the RGB data from flash since the flash is organized as blocks of bytes as described in Section 3.6.2. If more flash storage is needed, The Altera DE2 includes an SD card socket as shown in Figure 3.1, and a further implementation of this module will increase the number of scenarios possible.

9.1.3 SDRAM

There are many different options on how to implement SDRAM as a part of a frame buffer. For example, the SDRAM control can either be in software or hardware, or the system can use Direct Memory Access (DMA) to transfer RGB pixel data from memory to VGA-controller, meaning that the processor have noting to do with the transfer of bytes. SDRAM and frame buffers are described in Section 3.6.1 and Section 3.6.3 respectively.

The SDRAM controller has strict timing requirements, thus writing a new controller in VHDL seemed to introduce several challenges. Due to this, the first idea was to use SOPC builder for this implementation. The SOPC module for the SDRAM was easy to implement on the soft-processor Nios II, the same was the use of the SDRAM through HAL as described in Section 2.2. But unfortunately, no documentation was found on the frame buffer functionality of the SOPC builder. Thus another approach was tried out, this time with DMA and a First-in-First-out (FIFO) buffer. The purpose of the FIFO buffer is to synchronize the data flow between the Nios II, future described in Section 9.4.2, and the VGA-controller. A FIFO buffer is simply a buffer where the first data in, is the first data out. In this attempt, the system did not present a static picture

on screen, the screen image was distorted. Due to this, the performance of the Nios II was tested with a module in SOPC builder called performance counter to check the number of clock cycles the NIOS II used for one data transfer from the SDRAM to the FIFO buffer. The number was measured to 183 clock cycles for one transfer. Thus, even the fastest Nios II processor, the Nios II/f as described in Section 3.3, with a clock rate of 4 times the VGA pixel clock, did not make the data transfer in time. A solution may be to use the Nios II custom instructions to speed up the data transfer, but due to time in this project, and an alternative method in hardware, this was not implemented.

Section 3.6.3 describes that frame buffers are usually realized in hardware. This information, together with a example design from Terasic Technologies were found after an extended literature study of the frame buffer. The design found was used to display video from a camera to a monitor in real time, and the whole design was made in hardware. Thus, the end of the camera system data flow provided a frame buffer and a VGA-controller. Both of these modules were studied and implemented in The Embedded System for Electronic Circuit Education, although a VGA-controller was already designed, the timing and the communication between the VGA-controller and the frame buffer from Terasic was already implemented and ready to use.

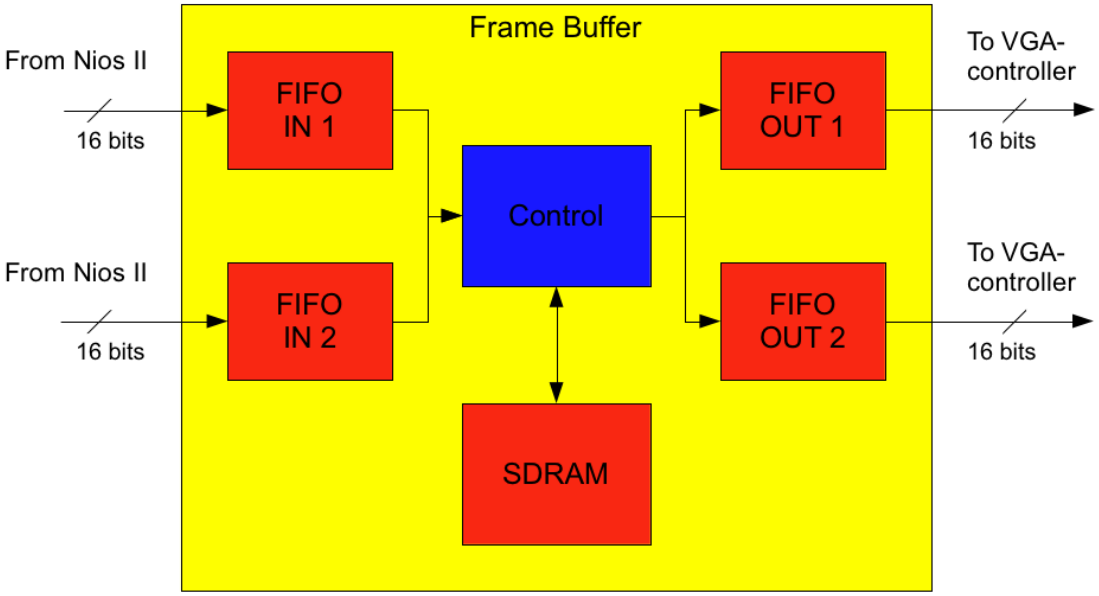

Figure 9.7: Simple Block Diagram of Frame Buffer from Terasic

A basic block diagram of the frame buffer from Terasic is shown in Figure 9.7. The reason for the use of two FIFO buffers in, and two FIFO buffers out, is because the SDRAM have 4M addresses with each 16-bits available as described in Section 3.6.1. The frame buffer have 32-bits in- and output, on the other hand, the Nios II sends 24-bits, and the VGA-controller only receive 30-bits. Thus, an adjustment of the data widths must be

performed. Table 9.3 present this adjustment.

FIFO	Bit 15	Bit 14 to 10	Bit 9 to 0
IN 1	0	Green[7–3]+00	Blue[7–0]+00
IN 2	0	Green[2–0]+00	Red[7–0]+00
OUT 1	Not in use	iGreen[9–5]	iBlue[9–0]
OUT 2	Not in use	iGreen[4–0]	iRed[9–0]

Table 9.3: Connections for the FIFO buffers

The designed VGA-controller and the VGA-controller from Terasic uses the same color signals as in Table 9.1, thus signal *iRed*, *iGreen* and *iBlue* from Table 9.3 are the inputs to the VGA-controller from Terasic as well. The signals *Red*, *green* and *blue* are the primary colors from the Nios II processor. To achieve the best possible color spectrum, Each of the 8-bits for *Red*, *Green* and *Blue* from Nios II, are placed in the MSB for both the FIFO IN 1 and FIFO IN 2, the remaining bits are set to zero as shown in Table 9.3.

After the implementation of the Terasic frame buffer, an image was displayed through the frame buffer to the monitor. However this image was deformed in the same way as for the DMA-version of the frame buffer. This made a theory that the Nios II sets a request signal high in several cycles longer than it should. The *request-to-write* signal is set high one cycle for the frame buffer to read one RGB pixel data from the Nios II. As mentioned, the Nios II failed to set the request signal low after one cycle. This conflict was solved by a module named the One Pulse Generator (OPG). The behavior of the OPG is as the name explains. When a trigger signal is set high, it generates one pulse with a given frequency from a clock signal, regardless of the trigger signals length. The OPG cannot generate a new pulse before the trigger signal has been set low. The module was achieved by a fellow designer with the same problem, and the code can be found in Appendix B.

As seen in Figure 9.7, the Nios II do not have direct access to the SDRAM. It can only be controlled by the frame buffer. This causes a static image on the screen because it is not possible for the Nios II to change parts of a picture without changing the contents of the frame buffer. However, it may be possible to rewrite the frame buffer to support writing directly from Nios II, but due to time constrains, this was not implemented in this project.

9.1.4 Flash

The SOPC builder, as described in Section 2.1, includes modules to implement different types of flash memory. The only options are the size and organization of the flash. Altera DE2 provides 4MB of flash memory as described in Section 3.6.2. The Nios II EDS, described in Section 2.2, includes a Flash Programmer for writing directly to a flash memory included in a Nios II processor. The Read-only zip file system is enabled

in the NIOS II processor, thus an uncompressed ZIP file with scenario pictures may be written to flash as a file system from the Nios II EDS.

With the zip file system, HAL supports easy access to flash through the commands fopen(), for opening a scenario file , and fread(), for reading the file. When the fread() command is executed, the Nios II sends a pointer of a free memory space the command uses for storing the flash content read. The problem is that fread() do not support offset reading in a file. This is a big drawback because without direct access to the SDRAM, as mentioned is Section 9.1.3, the Nios II must read and send one pixel at the time. However, the simple methods alt_open_flash() and alt_read_flash support offset reading, but with this methods the start address of each picture must be found manually. In addition, because of the zip file stored on flash, there is an offset before the scenario pixels are represented. The problem is solved by letting the Nios II begin at address(picture start+offset) when transferring pixels to frame buffer.

To conclude, the implemented version of flash works fine, but with a future frame buffer with direct access, the use of the Read-only zip file system would ease the implementation process of future scenario pictures.

9.2 Electronic Components and Domain Conversion

The electronic components used in this demonstrator are either mounted on the Altera DE2 and Atmel STK600, or connected through I/O pins on the Atmel STK600. To manipulate the virtual circuits in the demonstrator, an ADC must convert the analog voltage from the physical components into bits and a DAC must convert bits back to an analog voltage. The Altera DE2 provides both an ADC and a DAC in one chip, but this chip is designed for audio manipulation. This means that the input stage of the ADC and the output stage of the DAC includes a filter to eliminate direct current (DC). Due to this, the embedded system for electronic circuit education includes a development board from Atmel as well. The theory for this section can be found in this Chapter 4 and Chapter 6.

9.2.1 ADC

The microprocessor AT90USB1287 mounted on Atmel STK600 includes an 8-channel 10-bits ADC as mentioned in Section 4.2.1. Thus the ADC can be used in eight different scenarios or with eight different components. In Figure 9.8, an example on how to measure voltage across a LDR is shown. As seen in the figure, it is a voltage division between the resistance *R2* and the *LDR* that allows the ADC to read the voltage over the LDR. If an another scenario is added, the same procedure can be used for most passive components.

Passive components vary in value, thus to avoid wrong calculations of a voltage value, the ADC reads the voltage value four times and divide it by four. This reduces the chance of errors.

The ADC translates a voltage value to 10-bits. However, this architecture is 8-bit. The ADC solves this by splitting the 10-bits in two data registers, ADCH for the MSBs and ADCL for the LSBs. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the control signal ADLAR. Since the scenario "Automatic Adjustment of Light" operates with DC, the system do not need the high resolution with 10-bit. Thus the ADLAR bit is set, and only the ADCH register is in use. However, this can be changed if another scenario needs a higher resolution with the extra two bits.

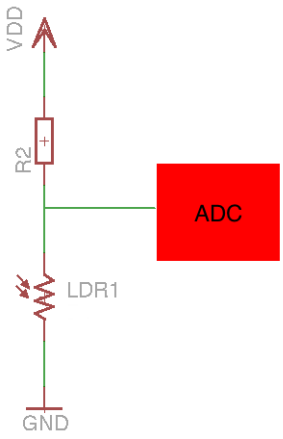

Figure 9.8: Circuit for reading voltage level

9.2.2 DAC and PWM

The AVR microcontroller from Atmel do not include a DAC. On the other side, it has four fixed 8-bits PWM channels and six with programmable resolution from 2 to 16 bits. As described in Section 4.2.3, PWM can be used as a DAC. With the microcontroller AT90USB1287, the duty cycle is varied by writing different values to the register which decides how long the PWM pulse is high. If the PWM is configured with an 8-bits resolution, and the value of the register is 64_{10}, which is 25% of 256_{10} (2^8), the duty cycle will also be 25% of the total pulse.

The PWM signal may function as a source to an electrical components, as long as the output current never exceeds 40 mA. This is the max avaliable DC current per I/O of the microcontroller. If higher currents are required by a component, Figure 9.9 shows an

alternative. The PWM controls a component through a transistor. With this alternative, the scenario can use higher source voltage if needed.

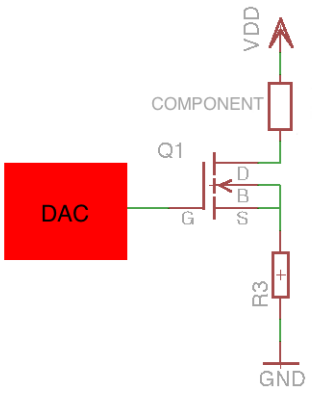

Figure 9.9: Circuit for writing voltage level

9.3 User Interface

The user interfaces is the most important modules in the demonstrator. In fact, if the user interface is not attractive enough, the system may not be used. The embedded system for electronic circuit education has the opportunity to use different technologies to both let the user interact with system, and also see information in different ways. The different technologies and where they are used is presented in this section.

9.3.1 Buttons, Switches and LEDs

Both the Altera DE2 and the Atmel AVR STK600 includes pushbuttons and LEDs. Furthermore, the Altera DE2 includes 18 switches, as seen in Figure 9.10. The eight switches in the right red square are pre-programed to switch between scenarios. However, if the system implements more than eight scenarios, the other switches may be chosen to scenario switching as well. In this version of the demonstrator, the other switches may be used for manipulation of the different scenarios. The switches are implemented with SOPC builder and connected directly to the Nios II through a positive-edge-interrupt-vector, which means that, the interrupts are only triggered with a positive pulse edge. The theory for interrupts may be found in Section 3.3.

As mentioned, both of the system demonstration boards have different LEDs. The LEDs above the scenario switches in Figure 9.10, indicates the current scenario. However, the

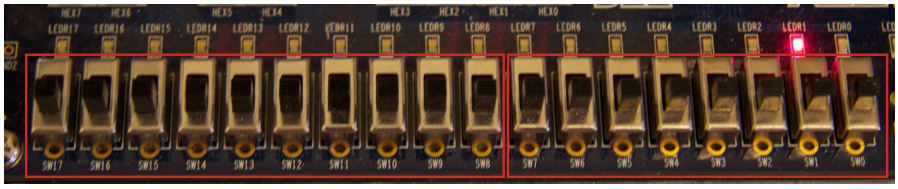

Figure 9.10: The 18 Switches on Altera DE2

other LEDs in this figure and Figure 9.11, may be used in different scenarios. The push-buttons in Figure 9.11 are active low, which means that they are perfect for reset circuits. Thus the *KEY0* is the Nios II system reset. The other three push-buttons of the Altera DE2 are not in use, but may easily be implemented in a scenario.

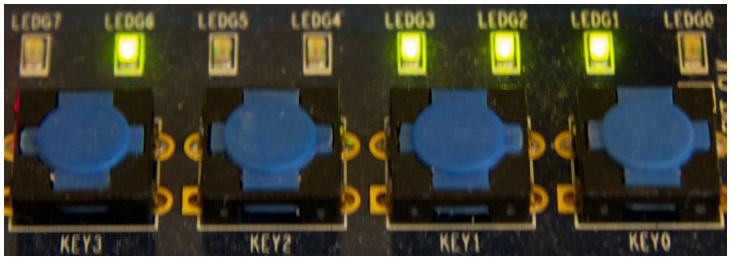

Figure 9.11: The 4 Pushbuttons on Altera DE2

Both the LEDs and the push-buttons on the STK600 shown in Figure 9.12 are controlled by the AVR microcontroller and used in different scenarios. If one of the buttons are pressed, an ISR activates a method which stores the current button pressed in a register. The Nios II reads this register through SPI as described in Section 5.1.

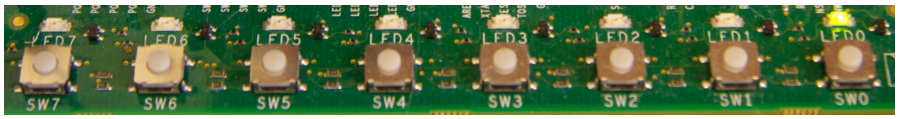

Figure 9.12: The 8 Pushbuttons on Atmel AVR STK600

9.3.2 7-Segment

As mentioned in Section 3.4, and as shown in Figure 9.13, the Altera DE2 includes eight 7-segment displays. The first choice of design tools when implementing these displays was the SOPC builder. However, the SOPC builder contains no module for the 7-segment display. Thus a control module was implemented in VHDL.

As seen in Figure 9.14, and from Table 9.4, the control module is a decoder from the Nios II to the 7-segment displays. The Nios II operates these displays with the following commands. First, set the address of the display to signal *ADDR*, and the desired

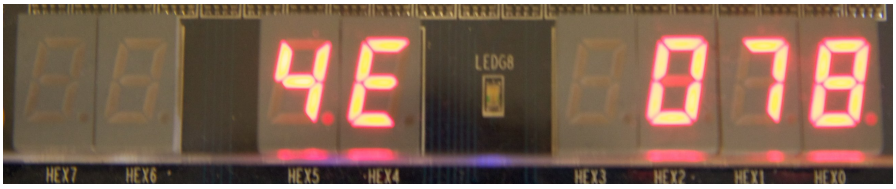

Figure 9.13: The eight 7-segment displays on Altera DE2

hexadecimal number to signal *DATA*. Secondly set the signal *SET*. The VHDL code for this module can be found in Appendix 3.8.

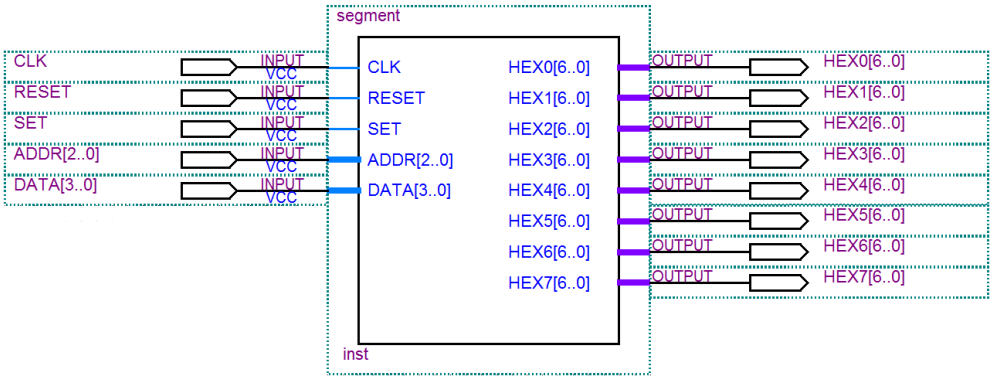

Figure 9.14: 7-Segment Decoder

Pin	Purpose
CLK	Nios II clock
RESET	Active low reset
SET	Setts DATA on ADDR
ADDR	3-bits address vector to choose between the 7-Segments
DATA	4-bits data vector for representing the number 0–F_{16}
HEX0 – HEX7	Eight outputs to the HEX0 to HEX7

Table 9.4: In and Outputs for the 7-Segment Decoder

9.3.3 Monitor and LCD

To present pictures and other scenario effects, the Embedded System for Electronic Circuit Education includes both an 16×2 LCD display, as mentioned in Section 3.1, and a monitor. These devices may be used in the different scenarios, but the monitor has its limitations, it can not display a dynamic image, as discussed in Section 9.1.2. The LCD module is generated with the SOPC builder, mentioned in Section 2.1, and contains two rows as seen in Figure 9.15. These two rows are set separately in the Nios II code.

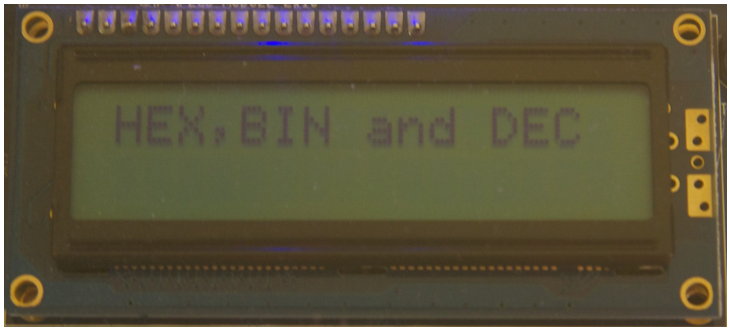

Figure 9.15: The Liquid Crystal Display on Atmel DE2

By the use of different ZIP files, as described in Section 9.1.4, the monitor may show different theory on screen depending on the educational level of the target group.

9.4 Communication and Processors

This section discusses the two communication interfaces USB and SPI, since these are both suitable and possible protocols for communication between the two processors in the system. In addition, the purpose of both the Nios II and the AVR AT90USB1287 is described. The supporting theory for the communication interfaces can be found in Chapter 5, while the theory for Nios II and the AVR can be found in Section 3.3 and Section 4.2 respectively.

9.4.1 Choice of Communication Interface

The idea was to implement USB as the communication interface between the Nios II and the AVR. USB is a well known communication protocol, and most people know that the purpose of a USB cable is to transfer data between modules. Both the AVR AT90USB1287 and the STK 600 includes support for a USB interface as mentioned in Chapter 4. In addition, the SOPC builder, described in Section 2.1, includes a module to realize the USB interface for the Nios II. However, when the SOPC module for the USB interface was included, and the generation of the new system was started, an error occurred. The failure was caused by a license error, and the reason why this occurred was unknown. Furthermore, the failure was studied with no luck of finding a solution.

After realizing that there was no way to implement the USB interface through SOPC builder, due to time constrains in this project, another approach for communication between the two development boards had to be found. As a result, the SPI interface was introduced. Similarly to USB, the SPI is supported by both the Nios II through SOPC builder, and the AT90USB1287 through STK600. Thus the USB benefits; speed, easy wiring between the cards and the demonstration effect, is not there. However,

the Scenarios added, and very likely the future scenarios, do not need the speed that USB offers, and SPI is considered to be fast enough. In addition, the SPI is easier to implement and to use in designs. The Altera DE2 was set up as the SPI master and STK600 as slave with SPI-interrupt. As mentioned, the implementation of SPI on Altera DE2 was done by the SOPC builder. In the SOPC-SPI-wizard, Altera DE2 was chosen as master. Due to this, the AVR AT90USB1287 was used as slave. This was achieved by setting the control bits *SPE* and *SPIE* in register *SPCR* to enable the SPI and the SPI-interrupt.

Table 9.5 shows the connection between the Altera DE2 and STK600. As seen from the table, one of the expansion headers (GPIO-0) on the Altera DE2 was used. The connection was made through a cable to the correct pins on the STK600, while the STK600 power and ground was delivered through a USB cable from the Altera DE2 USB-host.

Signals	**Altera DE2 Master**	**STK600 Slave**
MOSI	GPIO-0[2]	MOSI
MISO	GPIO-0[3]	MISO
SCLK	GPIO-0[1]	SCLK
$\overline{\textbf{SS}}$	GPIO-0[0]	PB0

Table 9.5: SPI Connections between the Altera DE2 and the STK600 with AT90USB1287

9.4.2 Tasks for the NIOS II

The Nios II was generated in SOPC-builder together with the different peripherals discussed in this chapter. As mentioned in Section 3.3, Nios II is a soft-processor implemented in an FPGA. In this demonstrator, it is implemented in a Cyclone II on a Altera DE2. In Section 9.4.1, the Nios II was declared as the master SPI device. However, the Nios II processor controls most of the user interface and the memory handling. For this reason the processor was chosen as the main control-unit.

The Nios II is supposed to control a dynamic frame buffer, but due to time constraints, as discussed in Section 9.1.3, this was not done. However, if a future dynamic frame buffer is implemented, the Nios II has to implement an algorithm to generate new RGB data for an area of pixels to changes screen contents. Thus to support the capacity of a main control-unit, and the control of a frame buffer, the NIOS II/f with a 100 MHz clock rate is implemented.

The microcontroller chose between scenarios with the interrupt-based switches presented in Section 9.3.1. When one of the switches is activated, the Nios II program counter is set to the ISR and a switch-interrupt-vector is generated, which the Nios II reads and uses to activate the desired scenario. The scenario program code is looped until a new

interrupt from the scenario switches occur. Furthermore, in each loop Nios II sends an SPI command with the information of the current scenario to AVR microcontroller.

9.4.3 Purpose for the Atmel AVR AT90USB1287

As mentioned in Section 9.2, the main purpose for the AVR is to control both the ADC and the PWM. In addition, the AVR respond to interrupts from both the user interface described in Section 3.2, and the Nios II through SPI. The supporting theory of Nios II and SPI can be found in Section 9.4.2 and Section 9.4.1 respectively. As mentioned in Section 9.4.1, the demonstrator intended to use USB as the communication interface, and this was featured by the AVR AT90USB1287, this was the main reason for the choice of microcontroller.

Each time the AVR receives an SPI-interrupt from the Nios II, the SPI-ISR is loaded into the AVR program counter. In the ISR code, the AVR sets the SPI message from the Nios II to the current scenario. Thus, the microcontroller executes the commands given in the current scenario and reports to the Nios II.

In future scenarios the AVR may be used for a lot more than it already is. With the current scenarios implemented in this demonstrator, the code uses 0.8% of the total capacity of the microcontroller. In addition, the microcontroller includes several other features than SPI, PWM and DAC.

9.5 Implementation Guide for Future Scenarios

In this section a recommended procedure for the implementation of future scenarios are described in an enumerated list. This procedure uses codesign methodologies, as described in Section 1.3, where both the current and future hardware modules are controlled from software.

1. *Overview:* First, it is important to get an overview of the desired scenario modules. Which modules do the demonstration need to present the theory desired.
2. *Modules:* Do the Altera DE2 or Atmel STK600 already include the implementation of the desired modules? If not, do they provide the modules? If the Embedded System for Electronic Circuit Education do not provide the modules, the solution may be to use a expansion header on the Altera DE2 or a port on the Atmel STK600, in addition to a external circuit.
3. *Implementation of extra modules:* If modules are already implemented by the system, go to point 4. The desired scenario-modules on Altera DE2 may be implemented by the SOPC builder, if not, a HDL driver must be made. However, Atmel provides a well documented manual for the AT90USB1287 [12]. This manual both describes and presents example code for the different features of AT90USB1287.

4. *Scenario:* Select a free scenario switch on the DE2 board that the scenario will be triggered on, as described in Section 9.3.1. Make a new state in the Nios II main file for the current scenario. If the Atmel STK600 board is needed, make a new state in the AVR main file as well, and use the SPI interface to communicate between these two processors.

5. *Methods:* Implement the methods needed for calculations and the control of the hardware modules in both processors.

6. *User interface:* Use the user interface to both present the scenario in best way possible, in addition to motivation of students. The screen may be used for demonstration materiel and can be customized by using different ZIP file for different target groups as mentioned in Section 9.3.3.

7. *Final adjustments:* In this point, the scenario is almost finished. Implement the methods made in point 5 into the states made in point 4. Finally, finish the demonstration materiel.

Chapter 10

Demonstration

In Section 1.4, several topics related to the design and implementation of a good demonstrator were mentioned. These topics are also important to have in mind when the system is demonstrated. In this chapter a guideline to how the different scenarios can be presented.

The different demonstrations may be presented oral with a demonstrator which introduce the topics of the scenarios. In addition, the system also has the opportunity to present itself through the monitor. As mentioned in Section 9.3.3, the demonstration materiel shown on the system monitor, may be individualized to the different target groups with the use of several zip files.

In Table 10.1 and Table 10.2 an example of the motivation and demonstration for the scenarios "Automatic Adjustment of Light" and "How to Count in Binary, Hexadecimal and Decimal" are presented.

Target groups	**Motivation**	**Demonstration**
Primary school pupils	Possible to change the light-intensity with their hands.	Give a short presentation of the system and mention that their mobile phones have some of the same functionality. In addition, let the pupils play with the scenario and answer any questions they may have.
Secondary school students	Familiar behavior used in everyday life.	Give a demonstration of the system and briefly and present Ohm's law, voltage division and the different components. Let the students play with the scenario and answer any questions they may have.
High school students	The students may already have an understanding of basic electronics from physics, and a motivation may be to see the practical use of it. In addition, the functionality of the system design may be a motivation.	In addition to the presentations of the scenario above, the demonstrator should explain shortly of how the system works.

Table 10.1: Presentation of "Automatic Adjustments of Light"

Target groups	**Motivation**	**Demonstration**
Primary school pupils	Possible to count with system	Give a short presentation of the system and mention which number system human and computer uses. In addition, let the pupils play with the scenario and answer any questions they may have.
Secondary school students	Learn the computer way to count and become aware of different numbers systems	Give a demonstration of the system and briefly describe the methods used to convert between them.
High school students	The students have already an understanding of different number systems from math, and this may motivate to see how the conversion between them works. In addition, the functionality of the system design may be a motivation.	In addition to the presentations of the scenario above, the demonstrator should explain shortly of how the system works.

Table 10.2: Presentation of "How to Count in Binary, Hexadecimal and Decimal"

Chapter 11

Conclusions

Embedded systems are well suited as demonstrators since the combination of hardware and software gives both flexibility in the design process and wide possibilities for optimization. With reconfigurable hardware like an FPGA, the development time gets shorter, and the systems functionality can be changed without replacing existing hardware.

In this master thesis, an embedded system for easy implementation of several scenarios with electronic educational aspects was made. In addition, two example scenarios were implemented,namely the Automatic Adjustment of Light and How to Count in Binary, Hexadecimal and Decimal. The realization of the embedded demonstrator was implemented on the Altera DE2 and the Atmel AVR STK600, with the use of the Cyclone II FPGA and the AVR AT90USB1287 microcontroller respectively as control units.

In the modern world, electronics are everywhere. However, most users operate these electrical systems without any understanding of how the devices actually work. The Embedded System for Electronic Circuit Education includes scenarios to demonstrate electronic circuits in our daily life, in addition to a basic education in these. With an interactive user interface and the possibility to choose between different scenarios, the user is able to learn the basic electronics that the system presents. This makes the embedded system well suited as a demonstrator, providing both inspiration and motivation for future students to choose a career in electronics.

Further development may improve the demonstration effects and the implementation procedure of the system. Firstly, the frame buffer should be improved to a dynamical frame buffer. With this functionality, the level of theory displayed on monitor may be adjusted through the user interface. Furthermore, the implementation of several more scenarios with different electronic educational aspects may contribute to a better demonstrator.

Bibliography

[1] Altera and Xilinx report: The battle continues. `http://seekingalpha.com/article/85478-altera-and-xilinx-report-the-battle-continues`.

[2] Altera Corporation. *Cyclone II Device Handbook, Version 2.0.* San Jose, California, USA, 2005.

[3] Altera Corporation. *DE2 Development and Education Board User Manual, Version 1.4.* San Jose, California, USA, 2006.

[4] Altera Corporation. *Nios II Custom Instruction User Guide, Version 1.5.* San Jose, California, USA, 2008.

[5] Altera Corporation. *Using the SDRAM Memory on Altera's DE2 Board with VHDL Design.* San Jose, California, USA, 2008.

[6] Altera Corporation. *Introduction to the Quartus II Software, Version 9.0.* San Jose, California, USA, 2009.

[7] Altera Corporation. *Nios II Processor Reference Handbook, Version 9.0.* San Jose, California, USA, 2009.

[8] Altera Corporation. *Nios II Software Developer's Handbook, Version 9.0.* San Jose, California, USA, 2009.

[9] Altera Corporation. *Quartus II Handbook, Volume 4:SOPC Builder, Version 9.0.* San Jose, California, USA, 2009.

[10] Angel, E. *Interactive Computer Graphics, A Top-Down Approach Using OpenGL*, fifth ed. Pearson Education International, Boston, Massachusetts, USA, 2009.

[11] Atmel. *AVR1900: Getting started with ATxmega128A1 on STK600.* San Jose, California, USA, 2008.

[12] Atmel. *AT90USB1287, 8-bit AVR Microcontroller with 64/128K Bytes of ISP Flash and USB Controller.* San Jose, USA, 2009.

[13] Bishop, O. *Electronics - A first Course*, second ed. Elsevier Ltd, Burlington, USA, 2006.

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

[15] Hyde, R. *Write Great Code : Understanding the Machine.* No Starch Press Inc., San Francisco, USA, 2004.

[16] Imsen, G. *Elevenes verden - Innføring i pedagogisk psykologi*, fourth ed. Universitetsforlaget, Oslo, Norway, 2005.

[17] Imsen, G. *Lærerens verden - Innføring i generell didaktikk*, third ed. Universitetsforlaget, Oslo, Norway, 2006.

[18] Integrated Circuit Solution Inc. *IS42S16400 SDRAM Memory, Rev.D.* Hsin-Chu, Taiwan, 2007.

[19] James O. Hamblen, T. S. H., and Furman, M. D. *Rapid Prototyping of Digital Systems - SOPC Edition.* Springer US, USA, 2008.

[20] Koren, I. *Computer Arithmetic Algorithms*, second ed. A K Peters, Natick, Massachusetts, USA, 2002.

[21] Kumar, S. *The codesign of embedded systems - A Unified Hardware/Software Representation.* Kluwer Academic Publishers, Massachusetts, USA, 1996.

[22] Mano, M. M., and Kime, C. R. *Logic and Computer Design Fundamentals*, third ed. Prentice Hall, New Jersey, USA, 2003.

[23] Mentor Graphics Corporation. *ModelSim User's Manual, Version 6.3g.* Oregon, USA, 2008.

[24] Michelin, G. D., and Gupta, R. K. Hardware/software co-design. In *Proceedings of the IEEE, VOL. 85, NO. 3* (1997), IEEE.

[25] Navabi, Z. *Embedded Core Design with FPGAs.* McGraw-Hill, USA, 2007.

[26] Nilsson, J. W., and Riedel, S. A. *Electric circuits*, seventh ed. Pearson Education International, New Jersey, USA, 2005.

[27] Pardue University, Jeffrey J. Richardson. *AVR Simulation with the ATMEL AVR Studio 4.* West Lafayette, Indiana, U.S, 2003.

[28] Pedroni, V. A. *Circuit Design with VHDL.* Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2004.

[29] Proakis, J. G., and Manolakis, D. G. *Digital Signal Processing*, fourth ed. Pearson Education, Inc, New Jersey, USA, 2007.

[30] Segura, J., and Hawkins, C. F. *CMOS Electronics - How it works, How it fails.* Institute of Electrical and Electronics Engineers, Inc, USA, 2004.

[31] Soper, M. E. *Absolute Beginer's Guide to A+ Certification.* Que Publishing, USA, 2004.

[32] Spansion Inc. *S29AL032D Flash Memory, Rev.A5.* Sunnyvale, USA, 2005.

[33] Venjum, K. A. Embedded System for Electronic Circuit Education. January 2010.

[34] Wilmshurst, T. *Designing Embedded Systems with PIC Microcontrollers, Principles and Aplications.* Elsevier Ltd., Oxford, UK, 2007.

Appendix A

VHDL-code: VGA-Controller

```
1  -- This module is for controlling a VGA-DAC.
2  -- The resolution is set to 640 x 480 @ 60Hz
3  -- The module requires a clock signal of approximately 25.197MHz
4
5  -- Designed by: Kai Andre Venjum, 03.03.2010
6
7  LIBRARY ieee;
8  USE ieee.std_logic_1164.ALL;
9  USE ieee.numeric_std.ALL;
10 USE IEEE.STD_LOGIC_ARITH.ALL;
11 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13 ENTITY vga IS
14 PORT ( CLOCK_PLL : IN std_logic; --clock from pll
15   RESET     : IN std_logic;
16   iRed      : IN std_logic_vector(9 downto 0);
17   iGreen    : IN std_logic_vector(9 downto 0);
18   iBlue     : IN std_logic_vector(9 downto 0);
19   VGA_B     : OUT std_logic_vector(9 downto 0);
20   VGA_R     : OUT std_logic_vector(9 downto 0);
21   VGA_G     : OUT std_logic_vector(9 downto 0);
22   VGA_BLANK : OUT std_logic;    --active low
23   VGA_SYNC  : OUT std_logic;    --active low
24   VGA_HS    : OUT std_logic;
25   VGA_VS    : OUT std_logic);
26 END vga;
27
28
29 ARCHITECTURE vg OF vga IS
30
31  SIGNAL zero                              :std_logic_VECTOR(9 DOWNTO 0);
32
33  SIGNAL h_sync, v_sync                    :STD_LOGIC;
34  SIGNAL video_on, video_on_h, video_on_v  :STD_LOGIC;
35
36
```

```
37  -- Numbers are calculated with 640*480*60Hz
38  -- referance Altera DE2 user manual
39  -- 1/25Mhz = 40ns Numbers in clockcycles
40  CONSTANT FP_h       : integer := 25;  --Front porch (horizontal)
41  CONSTANT DI_h       : integer := 640; --Display interval (horizontal)
42  CONSTANT BP_h       : integer := 55;  --Back porch (horizontal)
43  CONSTANT HSYNC_TIME : integer := 80;  --H_Sync pulse lenght
44
45  -- numbers in horizontal lines
46  CONSTANT FP_v        : integer := 10;  --Front porch (vertical)
47  CONSTANT DI_v        : integer := 480; --Display interval (vertical)
48  CONSTANT BP_v        : integer := 33;  --Back porch (vertical)
49  CONSTANT VSYNC_LINES : integer := 2;   --V_Sync pulse lenght
50
51
52  -- Do not change
53  CONSTANT h_sync_active  : integer := FP_h + DI_h + BP_h;
54  CONSTANT v_sync_active  : integer := FP_v + DI_v + BP_v;
55
56
57  SIGNAL h_count_ref : integer;
58  SIGNAL v_count_ref  : integer;
59
60  SIGNAL h_count : integer := 0;
61  SIGNAL v_count : integer := 0;
62
63
64  BEGIN
65  -- set ref value for on h_count and v_count
66  h_count_ref <= FP_h + DI_h + BP_h + HSYNC_TIME;
67  v_count_ref <= FP_v + DI_v + BP_v + VSYNC_LINES;
68
69  VGA_SYNC <= '1';
70  VGA_BLANK <= '1'; -- when blanc set all bits to zero
71
72
73  video_on <= video_on_v AND video_on_h;
74
75
76  zero <= "0000000000";
77
78
79  PROCESS
80   BEGIN
81   WAIT UNTIL ((clock_PLL'EVENT) AND ( clock_PLL = '1' ));
82
83   IF RESET = '0' THEN
84    h_count <= 0;
85    v_count <= 0;
86    video_on_v <= '0';
87    video_on_h <= '0';
88
89   ELSE
```

```
90
91      -- Generate Horizontal and Vertical Timing Signals
92      -- H_count counts pixels (640 + extra time for sync signals)
93
94      -- Horizontal Count
95
96      IF ( h_count = h_count_ref ) THEN
97       h_count <= 0;
98      ELSE
99       h_count <= h_count + 1;
100
101    END IF;
102
103     --Generate Horizontal Sync Signal using H_count
104      IF (( h_count <= h_count_ref) AND (h_count >= h_sync_active )) THEN
105      h_sync <= '0';
106     ELSE
107      h_sync <= '1';
108     END IF;
109
110     --V_count counts rows of pixels (480 + extra time for sync signals)
111
112     -- Vertical counter
113
114     -- 20 is added to h_sync_active to insure that h_sync is finish
115     IF ((v_count >= v_count_ref ) AND ( h_count >= (h_sync_active + 20))) THEN
116      v_count <= 0;
117     ELSIF ( h_count = (h_sync_active + 20) ) THEN
118      v_count <= v_count + 1;
119     END IF;
120
121     -- Generate Vertical Sync Signal using V_count
122     IF ( v_count <= v_count_ref) AND (v_count >= v_sync_active) THEN
123      v_sync <= '0';
124     ELSE
125      v_sync <= '1';
126     END IF;
127
128     -- Generate Video on Screen Signals for Pixel Data
129     IF ( (h_count <= h_sync_active) AND (h_count >= BP_h)) THEN
130      video_on_h <= '1';
131     ELSE
132      video_on_h <= '0';
133     END IF;
134
135     IF ((v_count <= v_sync_active) AND (v_count >= BP_v)) THEN
136      video_on_v <= '1';
137     ELSE
138      video_on_v <= '0';
139     END IF;
140
141     -- Put all video signals through DFFs to eliminate
142     -- any delays that can cause a blurry image
```

```
143     -- Turn off RGB outputs when outside video display area
144     IF (video_on = '1') THEN
145      VGA_R <= iRed;
146      VGA_G <= iGreen;
147      VGA_B <= iBlue;
148     ELSE
149      VGA_R <= zero;
150      VGA_G <= zero;
151      VGA_B <= zero;
152     END IF;
153
154     VGA_HS  <= h_sync;
155     VGA_VS  <= v_sync;
156
157    END IF;
158   END PROCESS;
159  end vg;
```

Listing A.1: VGA-Controller

Appendix B

Verilog-code: One Pulse Generator

```
1  //This module generates a single pulse on a rising edge input.
2  //The puls pulsehas a length of one clock period based on input clock.
3
4  //Designed by: Cato Marwell Jonassen, 12.05.2010
5
6
7  module  SINGLEPULSEGEN(//Inputs
8                                                  iCLK,
9                                                  iSIG,
10                                                 //Outputs
11                                                 oPULSE
12
13 );
14
15
16 //Input ports
17 input   iSIG;
18 input   iCLK;
19
20 //Output ports
21 output  oPULSE;
22
23 //Registers
24 reg             oPULSE;
25 reg     temp;
26
27
28 always @(posedge iSIG or posedge oPULSE)
29 begin
30         if(oPULSE)
31                 temp <= 0;
32         else
33                 temp <= 1;
```

```
34  end
35
36
37  always @(posedge iCLK)
38  begin
39          oPULSE <= temp;
40  end
41
42  endmodule
```

Listing B.1: One Pulse Generator

Appendix C

VHDL-code: Seven-Segment Decoder

```
1  -- This module is for controlling 8 seven-segment displays.
2  -- To set a hex-number on a display:
3  -- Set ADDR and DATA enable with SET
4
5  -- Designed by: Kai Andre Venjum, 30.05.2010
6
7  LIBRARY ieee;
8  USE ieee.std_logic_1164.ALL;
9  USE ieee.numeric_std.ALL;
10 USE IEEE.STD_LOGIC_ARITH.ALL;
11 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13 ENTITY segment IS
14 PORT (  CLK             : IN std_logic;
15                 RESET    : IN std_logic;
16                 SET             : IN std_logic;
17                 ADDR     : IN std_logic_vector(2 downto 0);
18                 DATA     : IN std_logic_vector(3 downto 0);
19                 HEX0     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
20                 HEX1     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
21                 HEX2     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
22                 HEX3     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
23                 HEX4     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
24                 HEX5     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
25                 HEX6     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1');
26                 HEX7     : OUT std_logic_vector(6 downto 0) := (OTHERS => '1')
27                 );
28 END segment;
29
30
31 ARCHITECTURE x OF segment IS
32
33         SIGNAL hex      :std_logic_VECTOR(6 DOWNTO 0);
```

```
34
35  BEGIN
36
37  PROCESS
38          BEGIN
39          WAIT UNTIL ((CLK'EVENT) AND ( CLK = '1' ));
40                  IF RESET = '0' THEN
41                          HEX0    <=      "1111111";
42                          HEX1    <=      "1111111";
43                          HEX2    <=      "1111111";
44                          HEX3    <=      "1111111";
45                          HEX4    <=      "1111111";
46                          HEX5    <=      "1111111";
47                          HEX6    <=      "1111111";
48                          HEX7    <=      "1111111";
49                  ELSE
50                          IF (DATA = "0000") THEN
51                                  hex <= "1000000";--0
52                          ELSIF (DATA = "0001") THEN
53                                  hex <= "1111001";--1
54                          ELSIF (DATA = "0010") THEN
55                                  hex <= "0100100";--2
56                          ELSIF (DATA = "0011") THEN
57                                  hex <= "0110000";--3
58                          ELSIF (DATA = "0100") THEN
59                                  hex <= "0011001";--4
60                          ELSIF (DATA = "0101") THEN
61                                  hex <= "0010010";--5
62                          ELSIF (DATA = "0110") THEN
63                                  hex <= "0000010";--6
64                          ELSIF (DATA = "0111") THEN
65                                  hex <= "1111000";--7
66                          ELSIF (DATA = "1000") THEN
67                                  hex <= "0000000";--8
68                          ELSIF (DATA = "1001") THEN
69                                  hex <= "0011000";--9
70                          ELSIF (DATA = "1010") THEN
71                                  hex <= "0001000";--A
72                          ELSIF (DATA = "1011") THEN
73                                  hex <= "0000011";--b
74                          ELSIF (DATA = "1100") THEN
75                                  hex <= "1000110";--C
76                          ELSIF (DATA = "1101") THEN
77                                  hex <= "0100001";--d
78                          ELSIF (DATA = "1110") THEN
79                                  hex <= "0000110";--E
80                          ELSIF (DATA = "1111") THEN
81                                  hex <= "0001110";--F
82                          END IF;
83
84                          IF (SET='1') THEN
85                                  IF (ADDR = "000") THEN
86                                          HEX0 <= hex;
```

```
87                                      ELSIF (ADDR = "001") THEN
88                                              HEX1 <= hex;
89                                      ELSIF (ADDR = "010") THEN
90                                              HEX2 <= hex;
91                                      ELSIF (ADDR = "011") THEN
92                                              HEX3 <= hex;
93                                      ELSIF (ADDR = "100") THEN
94                                              HEX4 <= hex;
95                                      ELSIF (ADDR = "101") THEN
96                                              HEX5 <= hex;
97                                      ELSIF (ADDR = "110") THEN
98                                              HEX6 <= hex;
99                                      ELSIF (ADDR = "111") THEN
100                                             HEX7 <= hex;
101                                     END IF;
102                             END IF;
103                     END IF;
104             END PROCESS;
105     end x;
```

Listing C.1: Seven-Segment Decoder