HW/SW Co-design of embedded systems within Alteras design environment

The Department of
Computer Science and Electronics
2006-02-14

Examiner: Lennart Lindh
Supervisor: Lennart Lindh
Author: Zoltan Nagy
Abstract

Hardware/Software (HW/SW) design has become a strategic technology for modern electronic systems. System designers are now turning to hardware/software (HW/SW) codesign approaches that offer real time capabilities while maintaining flexibility to support increasing complex systems and time-to-market.

This thesis report presents a hardware/software design, implementation, and testing of a basic embedded system which includes more or less complex components like a CPU, on-chip RAM, external RAM a UART and a Real Time Operating System kernel (RTOS) core, which is implemented as a hardware component. All the above mentioned, includes simulation along with error tracking, timing verification, synthesis, software performance analysis and hardware/software debugging. The mentioned design is intended to be used in course projects at Mälardalen University. As the VHDL files that describe the RTOS IP are under strict license, any unauthorized access can be severely damaging for the copywriter, thus the method of converting a VHDL based intellectual property core or component into a netlist based component is also illustrated and discussed. The report furthermore presents a VGA Controller design which was developed in this thesis work. Evaluation of two development boards is included in the report as well. In addition, a significant part of the work serves as a tool for educational purpose at Mälardalens University. For this purpose, to assist teaching, descriptive tool-guides were also created. This thesis report can be useful as an introduction to Alteras development environment, which consists of reconfigurable technologies and supporting design tools, and how these tools can be used in HW/SW design of embedded systems.
Acknowledgements

I would like to dedicate my acknowledgement and appreciation to all, who in some way or another, was involved and supported me during this thesis. First and foremost, I would like to express my innermost gratitude to my supervisor/examiner Prof. Lennart Lindh for providing me with the opportunity to undertake this appealing project. Furthermore, I would like to thank him for all the encouragement he has provided me during the time of my work.

Next, I would like to show my thankfulness to Susanna Nordström, who despite a demanding PhD thesis, made time available to assist me on my project. Susanna, I wish you good luck for the remainder of your work, and all the best in the future.

Last but, certainly not least, a big thanks to all my family and girlfriend, for their love and that they stood by and supported me and most of all had patience and understanding, during the completion of my thesis work.
List of Acronyms

ANSI  American National Standards Institute
ASIC  Application Specific Integrated Circuit
CLB   Configurable Logic Block
CPU   Central Processing Unit
DAC   Digital to Analog Converter
DDR   Double Data Rate
DSP   Digital Signal Processor
FIFO  First In First Out
FPGA  Field Programmable Gate Array
FSM   Finite State Machine
GUI   Graphical User Interface
HW    Hardware
IC    Integrated Circuit
IEEE  Institute of Electrical and Electronics Engineers
I/O   Input Output
IOB   Input/Output Block
IP    Intellectual Property
IRQ   Interrupt Request Line
JTAG  Joint Test Access Group
LE    Logical Element
LED   Light Emitting Diode
LUT   Look-Up Table
PC    Personal Computer
PIO   Parallel Input Output
RAM   Random-Access Memory
RISC  Reduced Instruction Set Computer
ROM   Read Only Memory
SRAM  Synchronous Random-Access Memory
RGB   Red Green Blue
RTOS  Real-Time Operating System
SW    Software
SOPC  System on a Programmable Chip
UART  Universal Asynchronous Receiver Transmitter
VGA   Video Graphics Array
VHDL  VHSIC Hardware Description Language
VHSIC Very High Speed Integrated Circuit
VQM   Verilog Quartus Mapping File
# TABLE OF CONTENTS

Chapter 1 .................................................................................................................. 7
Introduction .................................................................................................................. 7
  1.1. Purpose .................................................................................................................. 7
  1.2. Accomplishments associated with this thesis work ........................................... 7
  1.3. Outline of the thesis report .................................................................................. 8

Chapter 2 .................................................................................................................. 9
Technology & Background .......................................................................................... 9
  2.1. Field Programmable Gate Array .......................................................................... 9
  2.2. VHDL ..................................................................................................................... 11
  2.3. Choosing the right development board ................................................................ 12
      2.3.1 The UP3-1C12 education kit .......................................................................... 12
  2.4. Development environment .................................................................................. 12
  2.5. Alteras HW/SW design flow ............................................................................... 13

Chapter 3 ................................................................................................................ 14
Design, implementation & verification of a basic embedded system .......................... 14
  3.1. Development of a simple HW/SW platform ....................................................... 14
      3.1.1 System generation in SOPC Builder .................................................................. 16
      3.1.2 Compilation, pin assignment and configuration of the FPGA ......................... 16
      3.1.3 Software development for the system in NiosII IDE .................................... 17
      3.1.4 Adding debugging support to the platform ..................................................... 19
      3.1.5 Software development with debugger for the new design ......................... 20
  3.2. Time Analyze method ....................................................................................... 22
      3.2.1. Adding the timer ........................................................................................... 22
      3.2.2. Time measurement techniques ..................................................................... 23
          3.2.2.1. Calculating the theoretical value for an operation ................................ 23
          3.2.2.2. Time measurement with the timer ..................................................... 24
      3.4. Controlling LEDs with a pushbutton .................................................................. 25
          3.4.1. Adding the PIOs to the system .................................................................. 26
          3.4.2. Software development for the system ....................................................... 26
  3.5. Migrating Software to Hardware ........................................................................ 27
      3.5.1. Designing the hardware ............................................................................... 27
      3.5.2. Software development for the system ......................................................... 28
      3.5.3. Software vs. Hardware ................................................................................ 28
  3.6. Integration of a RTOS kernel in the design ......................................................... 29
      3.6.1. Integrating the Sierra hardware component ............................................... 29
      3.6.2. Using the RTOS kernel Sierra in software development ......................... 31
  3.7. Adding an external RAM to the system .............................................................. 32
      3.7.1. Connecting the external SRAM to the system ............................................. 32
      3.7.2. Testing the external SRAM ........................................................................ 33
  3.8. Creating a netlist based component ................................................................... 34
      3.8.1. Converting the VHDL files into a netlist file .............................................. 34
      3.8.2. Creating the wrap VHDL file & creating the netlist based component ... 35
Chapter 1

Introduction

1.1. Purpose
The thesis main objective is to get familiar with the Hardware/Software (HW/SW) codesign environment, which includes reconfigurable technologies and supporting design tools. The thesis also gives an understanding on how intellectual property (IP) blocks or components are developed and integrated into embedded systems. In addition the project gives great insight into construction, integration and convergence of HW/SW. Hence, the project provides thorough knowledge and understanding of:

- System level designs, with building blocks or intellectual property (IP)
- Hardware design of IP cores using the hardware description language VHDL.
- Simulation along with error tracking
- Timing verification
- Synthesis
- Software performance analysis
- Insight in real time systems
- Hardware and software debugging.

In addition, a significant part of the work will serve as a tool for educational purpose at Mälardalen University. For this purpose, to assist teaching, descriptive tool-guides were created, see Appendix 5 and 6. The project included the different processes of project management such as; selection of topic of the project, planning, selection of research methods, evaluation, analysis and finally implementation of findings. This meant that lot of knowledge and experience was accumulated while completing the project.

1.2. Accomplishments associated with this thesis work
The thesis work conducted, involved a lot of pre-studying, researching, and analyzing preexisting technologies. Subsequently, followed several design tasks. Below is a short, bullet point outline, of the entire thesis work.

- Prior to the undertaking of the thesis work, a feasibility study have been conducted of Alteras design tools, with the purpose of getting familiar with Alteras HW/SW design flow.
- Laboratory projects in course HW/SW design of embedded systems, which only existed in Xilinx design environment, was converted, modified and remade to Alteras design environment.
- Project descriptions (tasks) in Xilinx design environment was rewritten into technology independent descriptions, in able to be used within both Xilinx and Alteras design environment.
- Several Quartus II guides were written, i.e.: QuartusII licencing guide.
• In the earlier part of the project, a preexisting development board was utilized. However, because of insufficient memory resources of the FPGA that was located on the development board, a new more potent board had to be applied. In selecting a development board, several development boards was carefully studied, evaluated and compared. Finally, the one believed to be the most optimal (in reference to performance, cost), was selected.

• A VHDL IP component was converted into a net list base component. The VHDL files for this specific component are under strict license, any unauthorized access can be severely damaging for the copywriter, and thus all files were converted into a netlist.

• Design implementation and testing of a VGA Controller IP component was also conducted.

1.3. Outline of the thesis report
The thesis report begins with a very brief introduction to FPGAs and the hardware description language VHDL. In chapter 3 a design of a simple Hardware/Software (HW/SW) platform is presented, consisting of a CPU, on-chip RAM and an UART. Later in the design flow more/less complex components are integrated in the system, like external RAM and Real Time Operating System kernel (RTOS) core which is implemented as a hardware component. At the end of chapter 3 the method of converting a VHDL based component into a netlist based component is also illustrated and discussed. In chapter 4 design, implementation, and testing of a VGA Controller is presented. At last chapter 5 wraps up the thesis, briefly restating the conclusions and suggesting some possible directions for future research.

The thesis report is intended to be read straight through but the reader needs to have minor knowledge in digital hardware design, computer architecture and the programming language C, a reader familiar with FPGA technology and experiences in VHDL, may want to skip chapters 2.1.and 2.2.
2.1. Field Programmable Gate Array

A field programmable gate array (FPGA) is a general-purpose integrated circuit that is "programmed" by the designer rather than the device manufacturer. Unlike an application-specific integrated circuit (ASIC), which can perform a similar function in an electronic system, generally an FPGA can be reprogrammed, even after it has been deployed into a system. A FPGA is programmed by downloading a configuration program called a bitstream into the memory. These memories vary in different technologies. The three most common technologies are SRAM-based technology, flash-based technology and anti-fuse technology.

The SRAM-based and flash-based technology are similar in the way that they are both reprogrammable while the anti-fuse technology is one-time programmable. The SRAM-based devices lose their configuration data if VCC fails or is reapplied. Therefore, when using SRAM based technology a flash chip is needed outside the FPGA circuit to store the configuration data. The content of the flash chip is loaded to the FPGA power-on, which means that some extra start-up time is needed for SRAM-based FPGAs. Flash-based FPGAs do not need an external flash chip, as SRAM-based FPGAs do. Instead programmable flash cells are integrated in the chip. The advantage of this solution, apart from that no external memory chip is needed, is that the circuit is live directly at start-up because of the non-volatile property of the flash cells. The disadvantage is that the speed of the circuits is lower in flash-based FPGAs compared to SRAM-based FPGAs and also offers lower gate capacity. The anti-fuse technology uses a dielectric material in the switches. Normally this material has high impedance but when a high enough voltage is applied over the switch, the material is altered to have low impedance which means that the switch closes. This process can not be undone but on the other hand no extra circuit or start-up time is needed as with RAM-based FPGAs. Due to the specialized processing steps, these anti-fuse based devices cannot migrate to the newest and most advanced CMOS processes and they do not offer multi-100,000 or million gate capability, compared to SRAM based FPGAs. Today the most successful and fastest-growing programmable device families are the SRAM-based FPGAs. SRAM-based FPGAs are the fastest-growing segment of the semiconductor industry, sharing technology with microprocessors. They are two manufactures that dominate the FPGA market today: these are Xilinx corp and Altera corp.
FPGAs provide the user with a two-dimensional array of configurable resources Xilinx calls these resources CLBs (configurable logic blocks), Altera calls them LEs (Logic Elements). These configurable resources in turn contain one or more so-called lookup tables (LUTs). These LUTs are so-called function generators that can implement any logic function of four variables. A LUT is a RAM with a four-bit address input and a one-bit data output.

Figure 2.1a. FPGA architecture principle

Figure 2.1b. Logical Element (LE)
Figure below illustrates how a particular logic function is implemented in a LUT.

<table>
<thead>
<tr>
<th>4-bit address</th>
<th>Look Up Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcba</td>
<td>out</td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 2.1c.** Logic function implemented in a LUT

Today FPGAs contain from 3,000 up to 200,000 LEs or CLBs. New special blocks has also been developed for the FPGAs these are dedicated DSP blocks, multipliers, dual port memories, tri-state buffers and digital clock managers.

2.2. VHDL

VHDL is the acronym for Very High Speed Integrated Circuit Hardware Description Language. VHDL is a language for describing the structural, physical and behavioral characteristics of digital systems. In addition, is it being used for documentation, verification, and synthesis of large digital designs. This is actually one of the key features of VHDL, since the same VHDL source code can in many cases achieve all three of these methods. The language supports hierarchies, reusable components and error management. The hierarchies are explained by structural VHDL, procedures and functions. The structural VHDL is similar to a block design. Structural VHDL design helps to focus on each block individually, which gives less complexity to handle at once and gives the ability to verify each design block separately. The components can be constructed technology independent. These components can be stored in libraries to be reused in many designs. The component can be designed generic, which means that parameters can be set before synthesis to adopt the design for the user’s needs.
2.3. Choosing the right development board.

It was previously mentioned that the preexisting FPGA on the development board, lacked sufficient memory recourse and thus a new development board had to be chosen. When evaluating and selecting, amongst several development boards from several manufacturers, it was discovered that the manufacturer of the current board in use (UP3-1C6) supplied a board called UP3-1C12. The two UP3 development-boards are almost identical; both are suitable for educational purposes relatively well. However, the Altera Cyclone EP1C12Q240 FPGA device, which is located on the UP3-1C12 board, is almost three times larger than the one that can be found on the UP3-1C6 board. Therefore the UP3-1C12 board is more suitable for larger SOC(System on chip) designs. Even if the UP3-1C12 board is slightly more expensive than the UP3-1C6 board, the higher density off the EP1C12Q240 makes it worth the investment. A major contributing factor, in selecting the board, was its similarities with the previous one. UP3-1C12 features can be found in Appendix 1.

2.3.1 The UP3-1C12 education kit

This board provides a low cost-of-hardware platform for learning purposes as well as an advanced low-cost, off-the-shelf module for use in sophisticated product. Based on the Altera Cyclone EP1C12Q240C8 FPGA which provides around 12,000 Logic Elements. It allows hardware design engineer to design, prototype and test IP cores or any hardware design using HDLs like Verilog or VHDL. The board provides industry standard interconnections, Memory Subsystem, Multiple clocks for system design, JTAG Configuration, expansion headers for greater flexibility, capacity and additional user interface features, see Appendix 1 for a more detailed description for the features of the board. Further, the UP3 education kit can be used for DSP applications by interfacing directly to a DSP processor or implementing DSP functions inside the FPGA. In short, it is a dual-purpose kit, which can be used for prototyping and developing VLSI designs as well as designing and developing microprocessor based embedded system designs.

2.4. Development environment

Major part of the design in this thesis will be implemented in a Field Programmable Gate Array (FPGA). The FPGA device is a Cyclone EP1C12Q240C8 provided by Altera corp. (See table 1.1. below for the Cyclone FPGAs features,).

<table>
<thead>
<tr>
<th>Logic Elements (LEs)</th>
<th>12060</th>
</tr>
</thead>
<tbody>
<tr>
<td>M4K RAM Blocks (4 Kbits + Parity)</td>
<td>52</td>
</tr>
<tr>
<td>Total RAM Bits/KBytes</td>
<td>239616/29.25</td>
</tr>
<tr>
<td>PLLS</td>
<td>2</td>
</tr>
<tr>
<td>Maximum User I/Os</td>
<td>173</td>
</tr>
</tbody>
</table>

Table 2.4. The Cyclone EP1C12Q240C8 FPGA Overview

The Sierra RTOS kernel that is used in the design flow is provided by RealFast AB. The off-chip SRAM which is also used in the design flow is located on the UP3 development board which is provided by SLS corp. The following development software and tools will be used throughout the design flow:
- **Quartus II 5.0 Web Edition** for hardware design, synthesis, place & route, pin-mapping and design verification.
  - **SOPC Builder** which is an integrated tool in the Quartus II software. It is a system-development tool for creating system based on-processors, peripherals and memories.
  - **Component Editor** which is a feature in SOPC Builder. Helps to integrate user-defined components. With the Component Editor developers can import HDL files and specify the interfaces to the component.
  - **Quartus II Programmer** used for programming the FPGA device.
- **Nios II IDE 5.0** for software design and debugging
  - **GNU Compiler** which is a full-featured American National Standards Institute (ANSI) C compiler. Included in the Nios II IDE
  - **GNUPro Debugger** for software debugging. Included in the Nios II IDE
- **Modelsim** a simulation environment, which enables to verify a particular VHDL source code.

2.5. Alteras HW/SW design flow
Generally the design flow starts in the Quartus II software by making a new project, subsequently, from the Quartus II the SOPC Builder tool can be accessed. In SOPC Builder the system is put together with ready intellectual property (IP) cores and/or user made IP cores. After the system is generated by the SOPC Builder tool, once again, the Quartus II software is used to add additional blocks to the system, if needed. Before the system is synthesized pin assignment is made (this is also done in the Quartus II software). After synthesis and fitter operation, the Quarus II Programmer tool is used to configure the FPGA device. For software development the Nios II IDE is employed. When software debugging the GNUPro tools are used, these tools are integrated in the Nios II IDE. If real-time system-level debugging is required, a tool named SignalTap II logic analyzer can be utilized, this tool can be accessed from the Quartus II software.

![Figure 2.5. Alteras design flow](image-url)
Chapter 3

Design, implementation & verification of a basic embedded system

This chapter discusses how HW/SW embedded systems are built of reusable building blocks or intellectual property (IP) within Alteras design environment. In addition software development, performance measurements and debugging is described and illustrated. The chapter starts by describing how a simple basic embedded system, consisting of a CPU, On-chip RAM and a UART, is developed. In order to utilize the systems correct function a simple application is written and ran. In Section 3.2. debugging support is added to the platform and its functions are explained. Section 3.3. describes how a high resolution timer is integrated into the system and how it is configured, with the purpose of measuring and analyzing software performance. There is also a description on how the timer is used in applications, to measure the duration of several executions. In section 3.4. the system is extended by adding PIOs (Parallel Input Output), which in this case are used to control four LEDs with a push-button. Section 3.5. shows how the performance in the system can be enhanced, by migrating software functions to hardware. In section 3.6. the platform is further extended by integrating a RTOS kernel. The section illustrates and describes how the RTOS kernel is integrated into the system, both in HW and SW, and how it can be used in application development. In section 3.7. an external SRAM is integrated in the system. The section describes how the interface to the of-chip SRAM is connected to the Avalon data bus. Subsequently a application that verifies the external SRAM is developed. At last section 3.8 describes how a VHDL based component is converted into a netlist based component.

3.1. Development of a simple HW/SW platform

The fundamental components that build up a HW/SW system are a CPU (Central Processing Unit) which processes the information in a system, On-chip RAM (Random Access Memory) to store the instructions for the CPU and a JATG UART (Joint Test Access Group Universal Asynchronous Receiver Transmitter) for communication with the host computer. These components communicate with each other through the system bus, see figure below.
The system is generated with the help of SOPC Builder tool. This tool makes easy to specify the system components and their connections and generate a complete system-on-programmable-chip (SOPC) in much less time than using traditional, manual integration methods. This is done in a graphical user interface (GUI), (see fig.2.1. on the next page). The connection is done by SOPC builder automatically with the help of the Avalon switch fabric. The Avalon switch fabric is Altera’s parameterized interface bus, Altera calls it the data bus, it provides a set of pre-defined signal types with which a user can connect one or more intellectual property (IP) blocks. These IP blocks can be selected from a list in SOPC Builder component library, downloaded from the internet, or they can be user-made components.
3.1.1 System generation in SOPC Builder

The project starts by instancing the CPU, Altera’s Nios II CPU is used. The Nios II processor family consists of three cores—fast (Nios II/f), economy (Nios II/e), and standard (Nios II/s) cores—each optimized for a specific price and performance range. All three cores feature a general-purpose RISC CPU architecture and share a common 32-bit instruction set architecture. The Nios II/e is used in this project due to the fact that it is optimized for minimum logic usage, it uses 600-700 logical elements (LEs) and two M4ks. Next step is to instance the on-chip RAM and the JATG UART. In the on-chip RAM settings, 32 bits memory width and total size of 20kByte is set, the on-chip RAM is built up of M4k memory blocks. The JATG UART Read/Write FIFO is set to be constructed of registers instead of M4ks, this option is set to save M4ks, another method to save M4k memory blocks is to reduce the Read/Write FIFO.

The system is ready to be generated at this stage.

3.1.2 Compilation, pin assignment and configuration of the FPGA

SOPC Builder automatically creates a symbol of the system top design when the system generation is completed successfully. SOPC Builder generates these files after system generation: A SOPC Builder project file (.ptf) for the SOPC Builder system and a Block Symbol File (.bsf) of the system top design. After adding input symbols to the clock input and the reset input in the Quartus II Block Editor tool (see figure 2.2 below) the design is ready for analysis and synthesis.

Before the FPGA can be configured (programmed) pin assignment must be made, which means that the FPGA pins must be assigned to the input/outputs of a design. In this case the
17

sys clk and the sys rst n inputs (see figure2.2. above), pin assignment is made in the Assignment Editor within the Quaru II software. Also a full compilation of the design has to be completed. Full compilation includes:

- **Analysis and Synthesis**: performs logic synthesis to minimize the design logic and performs technology mapping to implement the design logic using device resources such as logic elements.
- **Fitter**: places and routes the logic of a design into a device.
- **Assembler**: converts the Fitter’s device, logic, and pin assignments.
- **Timing Analyzer**: analyzes and validates the timing performance of all the logic in a design.

Configuration of the FPGA is done with the Quaru II programmer tool, with the help of a ByteBlaster II cable. The configuration mode is JATAG.

FPGA Resource usage after compilation:

- Total logic elements (LEs): 2,668 / 12,060 (22 %)
- M4Ks: 36 / 52 (69 %)
- Total memory bits: 140,288 / 239,616 (58 %)

3.1.3 Software development for the system in NiosII IDE

By this stage the HW structure of the system is complete. To utilize it and verify whether it is working correctly, software has to be created. The programming language that is used is ANSI C. ANSI C (Standard C) is one standardized version of the C programming language.

Before the code (software) can be generated and executed, a project has to be built in Nios II IDE ("user application project") which in turn needs a system library project ("Hardware Abstraction Layer (HAL) system library project"). The system library is created by Nios II IDE automatically after the user application project is created.

---

**Figure 3.1.3a. HAL**

- **Software Application Based on HAL**
  - **User Application Project**
    - **Also know as**: Your program, or user project
    - **Described by**: .c, .h, .s files
    - **Created by**: User
  - **HAL System Library Project**
    - **Also know as**: HAL, or system library project
    - **Described by**: Nios II IDE project settings
    - **Created by**: Nios II IDE
  - **SOPC Builder System**
    - **Also know as**: Nios II processor system, or the hardware
    - **Described by**: .puf file
    - **Created by**: SOPC Builder
After the project is created the following settings are set in the system library properties:

- **Small C library** - When checked, the system library uses a reduced implementation of the Newlib ANSI C standard library. Notably, the printf() family of routines (printf(), fprintf(), sprintf(), etc.) does not support floating-point values when this option is checked. The reduced library is optimized for smaller memory footprint, although the implementation may be less time-efficient.

- **Reduced device drivers** - When checked, the compiler will include the reduced version of device drivers for all devices that provide small drivers. This reduces memory footprint at the expense of functionality. In this example, the JTAG UART will switch to a smaller, polled-operation driver (by default it is interrupt-driven), which executes slower but has a smaller code footprint.

The next step is to add a .c file to the created project, this .c file contains a very simple application that outputs the following messages on the console, with one second time interval:

```
** Main Starts **

Korning nr 1

Korning nr 2

Figure 3.1.3b. Output on the console
```

The simple application consists of a main function and a function called my_sleep (See appendix for the C code). The main function consist of an eternal “while loop” which prints the message (see Figure 2.4. above). After the printing of the lines is complete, it calls the function my_sleep, which is used for delay. The my_sleep function consists of a “for loop” that loops 350000 times, this takes approximately one second. The function takes one argument, which is used for the delay. For example if the argument is one then it loops 350000 times, if the argument is two, then it loops 700000 times, which takes approximately two second. So the argument sets the number of seconds of the delay time. In this case it is set to one, for one second delay. The project has to be built before the application can be executed. By clicking on **Project > Build Project** Nios II IDE builds the project. The application is executed by clicking on **Run > Run As > Nios II Hardware**. The software size is 7204 Bytes.
3.1.4. Adding debugging support to the platform

In this part of the design one of the debugging modules that are available for the Nios II processor family is utilized. Debugging support, helps to find and fix software problems quickly, it enables a developer to locate and remove program errors and create more stable and reliable programs. The debugging module enables:

- Starting and stopping execution
- Setting breakpoints and watch points
- Analyzing registers and memory
- Collecting real-time execution trace data

There are four debugging modules available for the Nios II processor family, Level 1, Level 2, Level 3 and Level 4. The Level 4 debugging module is the most advanced of these four, see Figure 2.6 below for more details about these debugging modules. For the Nios II/e core which is used only Level 1 debugging module is available. If more advanced debugging is required then also one of the more advanced Nios II cores has to be employed. In this case the Level 1 debugging modules is sufficient so therefore it is selected.
The debug module connects to the JTAG circuitry built into the FPGA and connects to the host PC via the ByteBlaster II download cable.

### 3.1.5 Software development with debugger for the new design

A simple application is written to utilize the debugging module. The application prints the numbers 0-10 with a time interval on the console. Software breakpoints are added and stepping through the application is made, while observing variables and assembler instructions that are being executed, etc.

Clicking on **Run > Debug As > Nios II Hardware** in Nios II IDE enables debugging mode. The following tools and windows can be accessed in debugging mode, these make debugging easy and effective.
Figure 3.1.5b. Debugging windows
3.2. Time Analyze method

To make performance analysis and measurements in applications a good time measurement is needed. This is achieved with the help of a high resolution timer. The system is extended by integrating an interval timer, see figure 3.2. below. In this design the timer is used to measure several executions, this is described on the next page.

3.2.1. Adding the timer

In SOPC Builder the timer is added and a various options are set. The timeout period is set to 1msec. In the Hardware options menu users can select between preset configurations like Watchdog, Simple Periodic Interrupt and Full Featured. The timer is not used as a watchdog nor as a periodic IRQ generator therefore the Full Featured option is chosen.

After the system is generated, the Block Symbol File must be updated in the Quartus II Block Editor, then the system can be compiled and the FPGA can be configured without any pin assignment or other changes. The timer does not need any external output/input pins, it is only connected to the system bus, and therefore pin assignment is not necessary.
3.2.2. Time measurement techniques

The following operations are measured with the help of the timer.

```c
value_a = 5;
sum = sum +1;
sum = sum -1;
printf("hi");
printf("hej");
```

First the theoretical value is calculated, then with the help of the timer the actual number of clock cycles for the operation is measured. The time it takes to execute a operation can be calculated with the help of the following formula:

\[
T = \frac{1}{f},
\]

where \(T\) is time in seconds, \(n\) number of clock cycles, \(f\) is the clock frequency.

3.2.2.1. Calculating the theoretical value for a operation

By debugging the application, monitoring the assembly instructions for every C code execution and finding out how many clock cycles every assembly instruction takes, it is relatively easy to calculate the theoretical value for the whole operation.

The assembly instructions for

```c
value_a = 5;
```

are:

1. 0x000080b4 <main+128>: movi r2,5
2. 0x000080b8 <main+132>: stw r2,16(fp)

(1) Writes the value 5 to register 2
(2) Stores the content of register 2 on address 16 in the memory.

According to the Nios II Handbook `movi` and `stw` both has a duration of 6 clock cycles + the Avalon bus delay (see table 3.2. on the next page). The theoretical delay for the operation is: \(n = 12\) cycles and \(f = 48\text{MHz}\) the formula gives

\[
\frac{1}{48 \cdot 10^6} = 0.00000025 = 250\text{ns}
\]

The assembly instructions for

```c
sum = sum + 1;
```

are:

1. 0x00008120 <main+236>: ldw r2,24(fp)
2. 0x00008124 <main+240>: addi r2,r2,1
3. 0x00008128 <main+244>: stw r2,20(fp)

(1) Loads register 2 with the value from address 24 in the memory, which is the value of sum before the addition.
(2) Adds one to register 2.
(3) Stores the content of register 2 on address 24 in the memory.

The instructions `movi` and `stw` both has a duration of 6 clock cycles + the Avalon bus delay and `ldw` has a duration of 10 clock cycles + the Avalon bus delay (see table 3.2. on the next page) in this case \(n = 22\) and the theoretical delay for the operation is \(458\text{ns}\). The methods illustrated above are also used to calculate

```c
sum = sum -1;
```
Table 3.2.2.1. Instruction execution performance for Nios II/e Core

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal ALU instructions (e.g., add, cmplt)</td>
<td>6</td>
</tr>
<tr>
<td>branch, jmp, ret, call, callr</td>
<td>6</td>
</tr>
<tr>
<td>trap, break, eret, bret, flushp, wrctl, rdctl,</td>
<td>6</td>
</tr>
<tr>
<td>unimplemented</td>
<td></td>
</tr>
<tr>
<td>load word</td>
<td>6 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>load halfword</td>
<td>9 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>load byte</td>
<td>10 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>store</td>
<td>6 + Duration of Avalon write transfer</td>
</tr>
<tr>
<td>Shift, rotate</td>
<td>7 to 38</td>
</tr>
<tr>
<td>All other instructions</td>
<td>6</td>
</tr>
<tr>
<td>Combinatorial custom instructions</td>
<td>6</td>
</tr>
<tr>
<td>Multi-cycle custom instructions</td>
<td>≥6</td>
</tr>
</tbody>
</table>

Table 3.2.2.1. Instruction execution performance for Nios II/e Core

3.2.2.2 Time measurement with the timer

The HAL provides high resolution timing functions using a timestamp driver. A timestamp driver provides a monotonically increasing counter which can be sampled to obtain timing information. Calling the function \textit{alt_timestamp_start()} starts the counter. Calls to \textit{alt_timestamp()} returns the current value of the timestamp counter, which is number of clock cycles. Calling \textit{alt_timestamp_start()} again resets the counter to zero, \textit{alt_timestamp_freq()} returns the rate at which the timestamp counter increments by.

To be able to use the timestamp driver provided in HAL, the timer must be selected as the time stamp timer in the system library properties in Nios II IDE and also the alt_timestamp.h file needs to be included in the application C file.

The following technique is used to measure the time for the instructions. First start the timer, then execute the instruction and straight after read the value from the timer. Even though it seems straight forward, it is a little bit more complicated than this. The mentioned method needs a little modification. In fact, to start the timer takes a couple of clock cycles and so does to read it, therefore if the mentioned technique is used without any modifications, then the extra clock cycles that it takes to start and read the timer will be added to the actually number of clock cycles it takes to execute the instruction that is being measured. So the following modification is made: first the timer is started right after that the value is read. This value will be the number of clock cycles it takes to start and read the timer. This value is called offset (see appendix for the C code).

Now when the offset is known, the first mentioned technique can be used after that, the offset is subtracted from the time that was measured; this gives the actual number of clock cycles, for comparison between theoretical and measured values see table 3.2.2.2. below. Due to the fact that \textbf{Small C library} option is turned on, it affects the printf(“hi”) and printf(“hej”).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal ALU instructions (e.g., add, cmplt)</td>
<td>6</td>
</tr>
<tr>
<td>branch, jmp, ret, call, callr</td>
<td>6</td>
</tr>
<tr>
<td>trap, break, eret, bret, flushp, wrctl, rdctl,</td>
<td>6</td>
</tr>
<tr>
<td>unimplemented</td>
<td></td>
</tr>
<tr>
<td>load word</td>
<td>6 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>load halfword</td>
<td>9 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>load byte</td>
<td>10 + Duration of Avalon read transfer</td>
</tr>
<tr>
<td>store</td>
<td>6 + Duration of Avalon write transfer</td>
</tr>
<tr>
<td>Shift, rotate</td>
<td>7 to 38</td>
</tr>
<tr>
<td>All other instructions</td>
<td>6</td>
</tr>
<tr>
<td>Combinatorial custom instructions</td>
<td>6</td>
</tr>
<tr>
<td>Multi-cycle custom instructions</td>
<td>≥6</td>
</tr>
</tbody>
</table>
Table 3.2.2. Time measurement result

<table>
<thead>
<tr>
<th>Operation</th>
<th>Theoretical value s/clock cycles</th>
<th>Measured value s/clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_a = 5;</td>
<td>250ns/12 cycles</td>
<td>290ns/14 cycles</td>
</tr>
<tr>
<td>sum = sum +1;</td>
<td>458ns/22 cycles</td>
<td>458ns/22 cycles</td>
</tr>
<tr>
<td>sum = sum -1;</td>
<td>458ns/22 cycles</td>
<td>458ns/22 cycles</td>
</tr>
<tr>
<td>printf(&quot;hi&quot;);</td>
<td>–</td>
<td>64.7us/3106 cycles</td>
</tr>
<tr>
<td>printf(&quot;hej&quot;);</td>
<td>–</td>
<td>94.6us/4542 cycles</td>
</tr>
</tbody>
</table>

3.4. Controlling LEDs with a pushbutton

In this part of the project the goal is to control four light-emitting diodes (LEDs) with a push-button. When a push-button is pressed, a counter counts up and the value of the counter is shown by the LEDs in binary number representation. The pushbutton and the four LEDs can be located on the UP3 development board (see Appendix 1). Since the board only supports four LEDs, therefore the counter will count from 0 -15. Two parallel input/output (PIO) cores are integrated in the system, one drives the LEDs and acts as an output, and the second scans the button 10 times/second and acts as an input (see figure 3.4. below).

![Figure 3.4. Hardware architecture of the system](image)

The PIO cores provides memory-mapped interface between an Avalon slave port and general-purpose I/O ports. The PIO core provides easy I/O access to user logic or external devices. Some example uses are:

- Controlling LEDs
- Acquiring data from switches
- Controlling display devices
- Configuring and communicating with off-chip devices, such as application-specific standard products (ASSP)
3.4.1. Adding the PIOs to the system

In SOPC Builder the two PIOs are integrated to the system, the PIO that scans the push-button is selected as an input with one bit data width, also the synchronously capture option is turned on, it is set to the rising edge of the system clock, this PIO is named Button_pio. The other PIO which drives the four LEDs is set to be an output with four bits data width and is named Led_pio. After the system generation is completed successfully, analysis and synthesis are made and pin assignment is completed (see figure 3.4.1. below) the design is compiled. The FPGA is ready to be configured by this stage.

![Block Diagram of the system](image)

Figur 3.4.1. Block Diagram of the system

3.4.2. Software development for the system

HAL provides the C-language macros IORD and IOWR, these macros are used by HAL device drivers to access device registers, in this case the it is used to access the registers in the PIOs. These macros are defined in the `io.h` header file.

$IORD_{8DIRECT}(BASE, OFFSET)$ which is used in the application to scan the button PIO, make an 8-bit read access at the location with address $BASE+OFFSET$. The $BASE$ in this case is the Button_pio_Base which is the base address of the Button_pio, which was assigned in SOPC Builder. Button_pio_Base is defined in the header file system.h. The header file system.h defines all system peripherals, base addresses, etc, This header file is created when the project is built. Nios II IDE creates this file out of the system information which it finds in the .ptf file that SOPC Builder created after system generation.

$IOWR_{8DIRECT}(BASE, OFFSET, DATA)$ which is used in the application to drive the LEDs, make an 8-bit write access to write the value $DATA$ at the location with address $BASE+OFFSET$. The $BASE$ in this case is the Led_pio_Base and it is the base address of the Led_pio.

Alternatively the following macros can be used instead of those described previously:

$IOWR_{ALTERA_AVALON_PIO_DATA}(base, data);
IORD_{ALTERA_AVALON_PIO_DATA}(base);

These macros are defined in the altera_avalon_pio_regs.h

The applications consist of a main function, which scans the Button_pio monotonically. When a zero on the Button_pio is detected, (this indicates that the pushbutton is pressed) it writes the value of a variable named ledpio to the Led_pio (the LEDs) and then increases variable ledpio by one(see appendix for the c code). The main function stays in idle state until it detects a 1 on the Button_pio. This indicates that the push-button is released,
afterwards it starts over and scans the *Button_pio*. This is done to ensure that the counter only counts up once, even if the push button is held down for a longer time period.

### 3.5. Migrating Software to Hardware

Generally, software is used for features and flexibility, while hardware is used for performance. To improve performance and unload the Nios II CPU, the counter and the LED control part from the former software is moved to hardware see figure 3.5. below. A function is added to the software which prints the value of the counter on the console.

![System overview](image)

**Figure 3.5.** System overview

#### 3.5.1. Designing the hardware

The Led function is designed with the hardware description language Very High Speed Integrated Circuit Hardware Description Language (VHDL). The Led function consists of two blocks named *One_pulse* and *Counter* (see figure 5.1. below). The *One_pulse* block generates a pulse on a signal named *Count_up* when it detects that the push-button is pressed down. Even if the pushbutton is held down for a longer time period it generates only one pulse which is one clock cycle long. The *One_pulse* block also filters out any ripples, which may occur on the push-button, this is achived with a counter.

The *Counter* block is a simple 3bits binary counter, it counts from 0-15. The counter counts up when it detects a one on the *Count_up* signal.
In SOPC Builder the system is modified by removing the PIO named Led_pio, the one that drove the LEDs in the former design. The one bit PIO that scanned the button in the former design is renamed to Led_in_pio and its data width is extended to 4bits, this PIO reads the value of the four LEDs in this design. After successful system generation, a symbol file is created of the Led_function, so that it can be connected to the system according to figure above.

3.5.2. Software development for the system

The application starts by reading from the Led_in_pio, then it signs the value to a variable, next it prints the value on the console. To avoid that it prints the same value over and over again, the next time it scans the Led_in_pio, it tests if the Led_in_pio has the same value as before. This is done by comparing the variable with the new read value. If the value is the same, then no printing will occur. The application reads the Led_in_pio by using the following macro:

\[ \text{IORD}_8\text{DIRECT}(\text{LED}_\text{IN}_\text{PIO}_\text{BASE}, 0); \]

The \text{LED}_\text{IN}_\text{PIO}_\text{BASE} is defined in the header file \text{system.h}. The application uses a eternal while loop to scan the Led_in_pio periodically.

3.5.3. Software vs. Hardware

The execution time for the software solution of the LED control is measured and compared to the hardware solution execution time. The interval timer is used to measure the performance of the software (see appendix for the C code). The following execution is measured:

- Scan of the Button_pio
- Writing the value of the counter to the Led_pio
- Increasing the counter by one

These three steps take ONE clock cycle for the hardware solution to “execute”. The execution time for these three steps for the software solution is 90 clock cycles. This proves that the software solution unmatched the hardware solution. And that system performance can be boosted by migrating less complex software solutions to hardware.
3.6. Integration of a RTOS kernel in the design

An RTOS (real-time operating system) is an operating system that supports a timely response to external events. For example, a robot that moves around in real time requires real-time operating systems. RTOS must respond to events as they occur, they operate within strict time deadlines to their users and to the surrounding world. A real-time system supports multiple events, react to events in a predictable way, and provide dependability. An RTOS also gives the ability to create reusable software components.

The RTOS kernel that is used in this project is named Sierra. The Sierra is implemented in hardware; this gives a very high performance. It gives support for task handling, semaphores, timers and external interrupts. All operations are carried out in the Sierra, the very thin layer software that comes with the package is a driver for communication between CPU and hardware kernel.

3.6.1. Integrating the Sierra hardware component

The Sierra is integrated into the design with the help of a tool named Component Editor. The Component Editor is a feature within the SOPC Builder. The system generation is made by SOPC Builder. In the Component Editor, the Sierra VHDL files are added, signals and there interfaces are specified and software drives are added see figures below.

![Figure 3.6. Hardware architecture of the design with SIERA RTOS integrated](image-url)
Figure 3.6.1a. VHDL files are added.

Figure 3.6.1b. Signals and interfaces are specified, the signal type for each signal is also configured.

Figure 3.6.1c. Software files are added.
By this stage the system is ready to be generated. After the system is generated by SOPC Builder, the Block Symbol File must be updated, then the system can be compiled and the FPGA can be configured without any pin assignment or other changes. The Sierra does not need any external output/input pins, it is only connected to the system bus, and therefore pin assignment is not necessary.

3.6.2. Using the RTOS kernel Sierra in software development

To be able to use the Sierra properly, some things has to be done before the software can be developed. First, all the software files which are needed for the Sierra to work must be copied to the project folder. Then the Sierra needs to be initialized correctly before it can be used (see appendix for the C code).

The software that is developed creates two tasks, one task named Kalle and the other named Tommy. The Kalle task prints its name on the console with a time interval, this task has priority 3. The Tommy task print its name on the console and besides that it scans the Led_pio 10 times per second and prints its value on the console, this task has priority 2. The scanning of the Led_pio is made by using periodic start. This is achieved with the help of the following functions which are defined in sierra.c:

\[ \text{init\_period\_time}(100); \]

This function call initializes the period time for the calling task. In the task Tommy the period time is set to 100, which is 100ms. To get a 1ms time base, the time base registers need to be initialized correctly. The following function initializes the time base register:

\[ \text{Sierra\_Time\_base\_reg}=0x30; \]

30(hex) is given by the following formula, which is used to calculate the time base register value. \[ 1\text{ms} \times \frac{48\text{MHz}}{1000} = 30(\text{hex}) \] where 1m is the wanted tick time and 48MHz is the system clock frequency.

\[ \text{wait\_for\_next\_period}(); \]

This function call suspends the calling task until the start of next period time.

Also a semaphore named PRINT\_SEM is used, this is done so that the two tasks do not interrupt each another while printing. If a task has a semaphore, then the other task must wait until the semaphore is released.

\[ \text{sem\_take} (\text{PRINT\_SEM}); \]

\[ \text{sem\_release} (\text{PRINT\_SEM}); \]

The task currently using the print semaphore releases it. If there are one ore more tasks waiting for the semaphore, the first task in the queue will get the semaphore and can start executing.

The time for semaphore release is measured with the interval timer. According to the measurements, a semaphore-release takes 323 clock cycles to execute. The majority of these 323 clock cycles is likely caused by the function call \text{sem\_release} (\text{PRINT\_SEM}); and the execution of the device drives, the actually hardware execution for releasing a semaphore likely takes much fewer clock cycles. (See appendix for the whole C code for the application).
3.7. Adding an external RAM to the system

As applications are getting larger, more memory is needed, the on-chip RAM is no longer enough and due to the fact that nearly all memory blocks in the FPGA is used, the on-chip RAM can not be expanded. This is solved by adding an external (off-chip) RAM to the system. (See figure 7.1. below)

![Hardware architecture of the system](image)

The external RAM that is used can be located on the UP3 development board. The memory is an asynchronous Static Random Access Memory (SRAM) its size is 128Kbytes organized as 65,536 words by 16 bits.

A test program is developed, to test the external RAM, then the test program is included in the former software project as a task. Every time a push-button is pressed the task will start the memory test program. For this a second PIO that is used as an input is integrated to the system (See figure 7.1. on the previous page).

3.7.1. Connecting the external SRAM to the system

The SRAM component that is added in SOPC Builder can be downloaded from SLS website at [http://www.slscorp.com/UP3Support/pages/documents.php](http://www.slscorp.com/UP3Support/pages/documents.php) it is actually the interface to the external SRAM not the SRAM, it is just the name of the component. The external SRAM has bidirectional data pins, therefore an Avalon tristate bridge is connected between the Avalon system bus and the external SRAM. The Avalon tristate bridge is a component that can be added in SOPC builder. The tristate bridge creates I/O signals on the SOPC Builder system module, which must be connected to the FPGA pins in the top-level Quartus II project (See figure 7.2. below). These pins represent the Avalon switch fabric to the off-chip external RAM.
In SOPC Builder a second PIO is added and set to be an input with one bit data width, this PIO is connected to a second pushbutton which starts the memory test program, also the synchronously capture option is turned on, it is set to the rising edge of the system clock. After synthesis and analysis the FPGAs pins are assigned as the figure above shows. The system is then compiled and the FPGA is configured.

3.7.2. Testing the external SRAM

To test the functionality of the external SRAM, a small application is developed. This application is executed by a task called SRAM_test. The application writes a known pattern to all the addresses of the external SRAM, and then it reads back from all those addresses and controls that the read value is the same as that was written. This is achieved with two “for loops”, one that writes the known pattern and the other reads back from these addresses and test if the read pattern is correct. The writing and reading is done with the following two macros:

\[
\text{IOWR\_32DIRECT}(\text{memory\_base}, \text{offset}, \text{pattern});
\]

\[
\text{value} = \text{IORD\_32DIRECT}(\text{memory\_base}, \text{offset});
\]

The \text{memory\_base} is the base address for the SRAM this was assigned in SOPC Builder, and is defined in the header file \text{system.h}. The \text{offset} is the value that is increased by the “for loops”, this gives the address which is written to or read from. So the address 0 of the SRAM is the \text{memory\_base}+0, so here the offset is equal with 0, so to access higher addresses the offset is increased, until the last address is reached which is \text{memory\_base}+0x1FFFF. The \text{pattern} is the value that is being written to the \text{offset} address of the SRAM, the pattern is 1,2,3…., it is also increased by the “for loop” (See appendix for the C code)

The execution of \text{value\_a = 5}; was also compared, when the software was executed from the on-chip RAM then it took 14 clock cycles and from the external SRAM it took 22 clock cycles to execute \text{value\_a = 5};.
3.8. Creating a netlist based component

The previous designs including the Sierra RTOS component, are intended to be used in projects in the course named HW/SW construction of embedded systems given at Mälardalen University. As seen in the former designs the Sierra RTOS component is described with the hardware description language VHDL, these VHDL files that build up the Sierra can at this stage easily be accessed, they can be found in the Sierra RTOS component directory. However this is not preferred. As the VHDL files are under strict license, any unauthorized access can be severely damaging for the copywriter, thus all files are converted into a netlist. The following method is used to convert the Sierra RTOS component into a so called netlist based component.

First the VHDL files that describe the Sierra RTOS are converted into a netlist file. Then in the Component Editor tool the netlist that was created is added, instead of the VHDL files as before, then the Component Editor create the component out of the netlist. This way the VHDL file that describes the Sierra RTOS will not be present in the Sierra RTOS component directory. However the Component Editor tool in SOPC builder does not recognize netlist files, this makes the whole thing a little bit tricky, actually only HDL files are accepted by the Component Editor tool. To solve this problem a so called wrap VHDL file is created; the netlist that describes the Sierra RTOS is instantiated into this warp VHDL file. Now it is possible to import this wrap VHDL file in Component Editor and also at the same time import the netlist file. After the component is generated, then the Component Editor tool will copy the wrap file and the netlist file to the component directory.

3.8.1. Converting the VHDL files into a netlist file

The Quartus II software has a build in feature which makes it possible to export designs as a netlist, this feature is called Quartus VQM writer. The netlist that is created/exported by the Quartus VQM writer is an ASCII text file with the extension .vqm. A VQM file is an atom-based Verilog HDL netlist file that defines all logic for a particular top-level design entity. VQM is acronym for Verilog Quartus Mapping File. The Quartus VQM writer is therefore used to create the netlist file that describes the Sierra RTOS. First a new Quartus project is created and then the VHDL files which describe the Sierra RTOS are added to this project. There are three different ways to generate the .vqm file, see below:

- Turn on Save a node-level netlist in the Compilation Process Settings page in the Settings dialog box (Assignments menu), and then compile the design.
- Turn on Save a node-level netlist in the Back-Annotate Assignments dialog box (Assignments menu) and back-annotate a design.
- Choose Start > Start VQM Writer (Processing menu).

The last method expressed is used in this case.
3.8.2. Creating the wrap VHDL file & creating the netlist based component

Now that the netlist file is generated a so called wrap VHDL file is written. The .vqm file is treated as a normal VHDL file. The wrap VHDL file is the top module which instantiate the down module, the down module in this case is the netlist that describes the Sierra RTOS, see illustration below.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY Sierra16 IS
  PORT(addr : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        clk  : IN STD_LOGIC;
        reset_n : IN STD_LOGIC;
        read_n : IN STD_LOGIC;
        cs_n  : IN STD_LOGIC;
        din   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        extirq_n : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        dout  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        irq   : OUT STD_LOGIC);
END Sierra16;

ARCHITECTURE rtl OF Sierra16 IS
  COMPONENT sierra_top
    PORT(addr : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          clk  : IN STD_LOGIC;
          cs_n  : IN STD_LOGIC;
          read_n : IN STD_LOGIC;
          din   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          extirq_n : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          reset_n : IN STD_LOGIC;
          write_n : IN STD_LOGIC;
          dout  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
          irq   : OUT STD_LOGIC);
  END COMPONENT;
  BEGIN
    Sierra_inst :sierra_top PORT MAP(addr,clk,cs_n,read_n,din,extirq_n,reset_n,write_n,dout,irq);
  END rtl;
```

Now it is possible to import this wrap VHDL file in the Component Editor and also at the same time import the netlist file. The Component Editor will give an error message when it tries to analyze the .vqm file, this is because it cannot recognize it, but this error message can be ignored. The information that enclosed in the wrap file is sufficient to create the interface to the Avalon data bus; the .vqm file will only be used when the design is synthesized. Then the port type in the signal tab can be adjusted and so on, the component can no be created as usual. After the component is generated, SOPC builder will copy the wrap file and the netlist file to the component directory.
Chapter 4

Design, implementation & testing of a VGA Controller

This chapter presents the VGA controller IP that was developed during this thesis work. Initially the VGA industry standard and common VGA design solutions are discussed, for instance what common design methods are used to minimize the video buffer in existing VGA designs. In section 4.2.3. the current VGA design and the objective with the design is discussed. In section 4.3. the current VGA controller architecture and its design solution is illustrated and explained. Section 4.4. demonstrates the predefined software functions, which are supported by the VGA Controller, and how these functions are utilized. The section also describes how the software accesses the hardware through device drives. Finally, in section 4.5. the results, including simulation and synthesis result, test application which was developed to the VGA Controller are illustrated.

The main objective with the VGA Controller design is that it shall give an understanding on how intellectual property (IP) blocks or components are developed and integrated into embedded systems. The VGA design will be used in educational purposes in digital hardware design courses given at Mälardalen University, it also can be utilized as a demonstrative tool for simple game development. The goal is that the VGA controller shall be compatible with the industry VGA standard; therefore it is suitable for graphically displaying data from example industrial processes on a standard monitor. As it is compatible with any standard VGA monitor it gives a simple graphical user interface when integrated in electronic control systems.

Features

- Provide a simple and easy interface to the Avalon data bus
- Compatible with the industry VGA standard
- Refresh rate 60Hz
- Supports two screen resolution modes
  - 640x480 pixels
  - 320x240 pixels
- 8 color support
- Fully on-chip design, development board independent
- Character ROM support
  - Character size 7x8 pixels
  - Implemented in HW
4.1. The VGA industry standard

VGA industry standard" 640x480 pixel mode
General characteristics
Clock frequency 25.175 MHz
Line frequency 31469 Hz
Field frequency 59.94 Hz
One line
  8 pixels front porch
  96 pixels horizontal sync
  40 pixels back porch
  8 pixels left border
  640 pixels video
  8 pixels right border
---------------------------------
800 pixels total per line
One field
  2 lines front porch
  2 lines vertical sync
  25 lines back porch
  8 lines top border
  480 lines video
  8 lines bottom border
---------------------------------
525 lines total per field

Figure 4.1. VGA Timing
4.2. Design method

4.2.1. VGA signal production

A VGA display signal is produced with the help of five signals. Two signals, horizontal sync and vertical sync, which are used for synchronization of the video. Three signals which are used to control the color. The color signals are Red, Green, and Blue. They are often collectively referred to as the RGB signals. By changing the analog levels of the three RGB signals all other colors can be produced. This VGA Controller in this thesis supports 8 colors see figure below, no digital to analog conversion is made.

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Blue</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Green</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Cyan</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Magenta</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Yellow</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>White</td>
</tr>
</tbody>
</table>

In standard VGA format, the screen contains 640 by 480 picture elements (pixels). The video signal must redraw the entire screen 60 times per second to provide 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 Hz. The screen refresh process begins in the top left corner and paints one pixel at a time from left to right. 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 (See figure 1.). Once the entire screen has been painted, the refresh process begins again.

4.2.2. Common design solutions

Generally in existing VGA designs the Horizontal sync and the Vertical sync signals are generated by two counters clocked by the pixel clock. These two continuously running counters also form the address into a video display buffer, which temporary stores the RGB data for each pixel in a frame. The screen is constantly being refreshed, and it constantly fetches for new pixel data, therefore a video buffer is required. A frame consists of 640x480 pixels, as a result the video buffer requires a large amount of memory. For example a frame consisting of 640x480 pixels requires 113kByte memory size. Due to a limited amount of RAM blocks in lower density FPGAs this becomes a design issue. Generally the following methods are used in existing designs to minimize the memory usage.

**External RAM**

This solves the above mentioned issue. Use of an external RAM device usually enables access to a large amount of memory compared to an on-chip RAM. A large VGA buffer can easily fit into an off-chip RAM. Although there is a draw back with this method, different types of development boards contain different types of memory devices. For instance some memory devices use bidirectional port for data input/output while other RAM devices use an input for data in and a separate output for data out. Therefore if the VGA design is moved to another development board, the off-chip RAM controller in the design needs to be modified or even redesigned, which can be relatively time consuming.
Super Pixels

This method maps a large block of monitor pixels to a single memory location, which means that these pixels will have the same RGB color value. This technique creates a virtual resolution depending on the size of the “super pixel”. For example if the “super pixel size is” 4 by 4 pixel then the virtual resolution is 160x120 “super pixels”. This technique gives a minimal memory requirement. The drawback with this method is that the image quality is significantly reduced and may not be suited for many applications.

![Figure 4.2.2. Super pixels](image)

Black & White

By using two colors, the memory requirement can be decreased by three. By using two colors, only 1 bit is needed for the RGB color information. The draw back with this technique is obvious; only two colors can be displayed.
4.2.3. Current VGA design

The ambition is to make the VGA design development board independent and easy to integrate in embedded systems, therefore the video buffer will be implemented using memory blocks which are located in the FPGA, so no external memory is used. The goal is that the design shall be a fully on-chip design. As mentioned on the previous page, the video buffer requires a large amount of memory, and the design is therefore hard to implement in low density FPGAs. For this reason the mentioned technique Super Pixel is used in this design. Two modes are supported, 1x1 which gives 640x480 pixel resolution and 2x2 super pixel mode which gives 320x240 pixel resolution, this will be referred to as 320x240 resolution mode instead of super pixel mode. The design is made generic, which means that parameters can be set before synthesis; this is made to adopt the design for the user’s needs and available memory resource in the FPGA that is used. The parameters that can be set are memory size and editable screen size and the resolution mode. The editable screen size option dramatically decreases memory requirement, and combining this option with the 320x240 resolution mode results that the design can be implemented in most low density FPGAs (see table 3.1 below). There is no need to set these options when the design is implemented in high density FPGAs, 640x480 pixel mode and 640x480 editable screen size can be used.

<table>
<thead>
<tr>
<th>320x240 pixel mode disabled</th>
<th>Editable screen size</th>
<th>Required memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>640x480</td>
<td>113kByte</td>
<td></td>
</tr>
<tr>
<td>600x400</td>
<td>88kByte</td>
<td></td>
</tr>
<tr>
<td>512x384</td>
<td>72kByte</td>
<td></td>
</tr>
<tr>
<td>320x240</td>
<td>29kByte</td>
<td></td>
</tr>
<tr>
<td>300x200</td>
<td>22kByte</td>
<td></td>
</tr>
<tr>
<td>320x240 pixel mode enabled</td>
<td>640x480</td>
<td>29kByte</td>
</tr>
<tr>
<td>600x400</td>
<td>22kByte</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2.3a. Memory requirement when changing the editable screen size and/or changing to 320x240 resolution mode

Figure 4.2.3b. Editable screen size set to 600x400 pixels

The table on the next page shows the generic parameters that can be set before integrating the VGA controller into a system. The table explains what these parameters are, how they should be set and how they affect the design.
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen SIZE horizontal</td>
<td>Sets the horizontal size of the editable screen area. The maximum value is 640. Odd values should be avoided. For example 512 should be set instead of 513.</td>
</tr>
<tr>
<td>Screen SIZE vertical</td>
<td>Sets the vertical size of the editable screen area. The maximum value is 480. Odd values should be avoided.</td>
</tr>
<tr>
<td>Resolution320x240</td>
<td>Enables/Disables 320x240 pixel resolution mode. 0 = 640x480 pixel resolution. 1 = 320x240 pixel resolution.</td>
</tr>
<tr>
<td>RAM SIZE</td>
<td>This parameter sets the number of memory addresses, the actual memory size is three times the number of memory addresses, for the reason that each address consists of 3bits. This parameter depends on the Screen size horizontal and Screen size vertical parameter values. When setting this parameter, the following formulas are used to calculate the value: When 320x240 pixel resolution is disabled: $\text{Screen SIZE horizontal} \times \text{Screen SIZE vertical} = \text{RAM SIZE}$ When 320x240 pixel resolution is enabled: $\frac{\text{Screen SIZE horizontal} \times \text{Screen SIZE vertical}}{4} = \text{RAM SIZE}$</td>
</tr>
<tr>
<td>RAM ADDRESS BUS WIDTH</td>
<td>Sets the memory address bus width, this parameter is dependent on the Memory SIZE parameter. For example if the Memory SIZE parameter is set to 60000, then the RAM ADDRESS BUS WIDTH parameter should be set to 16, thus $2^{16} = 65536$, or if the Memory SIZE parameter is set to 307200, then the RAM ADDRESS BUS WIDTH parameter should be set to 19.</td>
</tr>
</tbody>
</table>

*Table 4.2.3c. Generic parameters for the VGA Controller*
4.3. Design Solution
The VGA Controller design flow can be divided into several steps:

- Design of the VGA Controller HW architecture
- Design of the interface to the Avalon data bus
- Writing Device drives
- Creating a test application

The VGA Controller hardware architecture is developed using the hardware description language VHDL. As VHDL enables component based design, which is called structural VHDL, the VGA controller is built up of several blocks. This design method helps to focus on each block individually, which gives less complexity to handle at once and gives the ability to verify each design block separately. This all eases the design of the hardware, resulting in more efficient error tracking, and consequently speeds up the development time. See figure below which show the component hierarchy in the VGA controller.

As it was mentioned prior, a video buffer is needed to temporary store the RGB value for each pixel in a frame. For this reason a memory block named **Pixel RAM** is developed. A block named **Character ROM**, which stores the supported character shapes, is constructed. The VGA Controller needs to communicate with the software trough the Avalon data bus, for this reason a block named **VGA Regs** is designed. The **VGA Regs** block acts as the interface to the Avalon data bus. The VGA monitor needs to be provided with synchronization signals, which define the ends of each line and frame. For this purpose a block named **VGA Sync Generator** is developed. As all these blocks need to be controlled, for instance the **VGA Sync Generator** block needs to be provided with RGB data from the **Pixel RAM**, the **Character ROM** needs to be read and the shapes converted to pixels and stored in the **Pixel RAM**. These blocks also need to communicate with the **VGA Regs** block, hence a block named **Control Unit**, responsible for providing the **VGA Sync Generator** block with RGB data from the **Pixel RAM**, handling the **Character ROM** and the communication with the...
**VGA Regs** block, creates. The VGA Controller has a global asynchronous active low reset, this signal is named Reset_n. The system Clock runs at 48MHz, the **VGA Sync Generator** block has a 24MHz internal clock frequency, and this is achieved with a simple frequency divider which generates a ripple free output signal named Clk24. Parts of the **Control Unit** block is also clocked by the Clk24. The **Control Unit** block and the **VGA Sync Generator** block is described in one VHDL file named **VGA_core.vhd**.

---

**4.3.1. Pixel RAM**

The Pixel RAM is one of the five blocks that builds up the VGA Controller. Its function is to temporary store the RGB value for each pixel in a frame, therefore it can be referred to as the video buffer, the RGB value for the first pixel in a frame is stored on the first address of the Pixel RAM, this pixel has the coordinate \((x=0, y=0)\). The number of addresses in the video buffer can be customized before synthesis since it is designed as a generic block. Each address contains 3bits. The video buffer is a fully synchronous component, its inputs and outputs are synchronized to the system clock see Appendix 3 for the VHDL source code. The video buffer is built up of memory blocks that are located in the Cyclone FPGA.

---

**Figure 4.3b. VGA controller architecture**

When the input signal \(RAM\_we\_n\) which is the write enable signal, switches from high to low on the positive edge of the system clock, then the value on the input signal \(RAM\_din\) is stored on the address in the memory which is given by the input signal named \(RAM\_addr\). The value of each address in the memory can be read from the output signal named \(RAM\_dout\).
4.3.2. Character ROM

The Character ROMs architecture is somewhat similar to the architecture of the video buffer, except that it does not have a write data input and write enable input. It is a read only memory (ROM). The content of this memory is pre initialized, that means that the initial values for each address are set when the FPGA is being configured. The character ROMs function is to store the pre defined character table that the VGA Controller supports. It actually only stores the shape information for each character, not the color information itself. It stores a total of 91 character shapes; each character size is 7x8 pixels, see Appendix 45 for the whole character table and which characters are supported. The Character ROM is organized as 728 words by 7 bits. The value one on each address represent a part of a character shape, a zero represent the background of a character, see figure 4.3.2. A new character starts every eight address. For easy access to any specific character each character start address is given a number, this is called the Char nr. By multiplying the Char nr with 8 will give the start address to a specific character. For instance character ‘B’ has the starts address 0x008 in the character ROM and it has the Char nr 1, multiplying 1 with 8 gives 8, which is the start address to character ‘B’. For example for the character ‘*’ which has the Char nr 90, multiplying 90 with 8 gives 720 which is the start address for the character ‘*’. The Character ROM is also build up of memory blocks that are located in the FPGA. The total size of the Character ROM is 637 byte.

<table>
<thead>
<tr>
<th>Value</th>
<th>Address</th>
<th>Char nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0 0 0</td>
<td>0e000</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 0 1 0 0</td>
<td>0e001</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e002</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e003</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1 1</td>
<td>0e004</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e005</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e006</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e007</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1 0</td>
<td>0e008</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e009</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 1 0</td>
<td>0e00a</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1 0</td>
<td>0e00b</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e00c</td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 1 1 0</td>
<td>0e0d</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 1 0</td>
<td>0e0e</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 0 0</td>
<td>0e0f</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e10</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 1 0</td>
<td>0e11</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 0 0</td>
<td>0e12</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d1</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d2</td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 1 1 1</td>
<td>0e2d3</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d4</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>0e2d5</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d6</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0e2d7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.3.2. Character ROM content

4.3.3. VGA Sync Generator

This block generates the synchronization signals for the VGA monitor. The signals are V_sync and H_sync, this stand for vertical synchronization and horizontal synchronization. See timing information on page 30. Since the UP3 development board does not support 25MHz clock frequency which is used in the VGA standard, therefore the 48MHz clock frequency, which is supported by the UP3 development board, is divided by two, this gives 24MHz and is the internal clock frequency for the VGA Sync Generator block. This result a slight difference in the signal timing, compared to the description on page 30, the difference is almost insignificant in this case. 24MHz clock frequency should work on most standard VGA monitors. The V_sync and H_sync signals are produced with the help of two separate counters, named V_count and H_count, these counters are clocked by the 24MHz clock. H_count counts from 0 to 799, this gives the line frequency 30kHz. The V_count counts from 0 to 524, this gives the field frequency 57.14Hz. V_count increases by one each time the H_count indicates end of a line.

The VGA Sync Generator block also creates a signal called RGB Data en. This signal gives an indication to the Control Unit block to obtain RGB data from the video buffer and provide
it to the VGA monitor. This indication happens when the RGB Data_en signal is high, otherwise when it is low, the RGB data output is disabled and set to zero, this gives the color black on the monitor screen.

According to the figure above $RGB\_Data\_en = RGB\_Data\_en\_h \cdot RGB\_Data\_en\_v$. By altering in which intervals $RGB\_Data\_en\_h$ and $RGB\_Data\_en\_v$ should be equal to one, the size of the editable screen area can be modified. This is achieved with the help of the two counters $V\_count$ and $H\_count$. As figure above shows the $RGB\_Data\_en\_h$ signal is high in the interval when the $H\_count$ value is between 0 and 640, and $RGB\_Data\_en\_v$ is high on the interval when the $V\_count$ value is 0 to 480. This will result in an editable screen area that is 640x480 pixels. By altering with generic parameters on which interval of the two counters the two signals $RGB\_Data\_en\_h$ and $RGB\_Data\_en\_v$ should be equal to one, the editable screen area size can be set. The parameters are named Screen_SIZE_horizontal and Screen_SIZE_vertical.

Figure 4.3.3a. Synchronization signal timing, editable screen size is set to 640x480 pixels

Figure 4.3.3b. Synchronization signal timing

Figure 4.3.3b. shows that $RGB\_Data\_en\_h$ is high on the interval when $H\_count$ is between 170 and 470, and that $RGB\_Data\_en\_v$ is high on the interval when $V\_count$ is between 140
and 340. In this case the generic parameters are set to 300 respectively 200, this gives 300x200 pixel size editable screen area, as $470-170 = 300$ and $340-140 = 200$. The editable screen size would have the same size if $RGB\_Data\_en\_h$ was equal to one on the interval, when $H\_count$ is between 0 and 299 and $RGB\_Data\_en\_v$ was equal to one on the interval, when $V\_count$ is between 0 and 199, the difference would be that the editable screen size would not be centered. It would be positioned towards the upper left corner of the monitor screen. The centering method that is used is achieved with four simple equations, for more detailed explanation see the source code for the VGA Sync Generator block in Appendix 3.

4.3.4. Control Unit

The Control Unit controls the majority in the VGA Controller. It provides the VGA Sync Generator block with RGB data from the Pixel RAM, it handles the Character ROM and also communicates with the VGA Regs block. The Control Unit obtains instructions from the software through the VGA Regs block and act accordingly to these instructions. A basic instruction can be to store a pixel at a specific coordinate and color information in the Pixel RAM, or to store a particular character at a specific coordinate and color information in the Pixel RAM. How the control unit handles these different types of operations are described on the following pages.

4.3.4.1. Selection between the two screen resolution modes & providing the VGA monitor with RGB data

As mentioned previously, the VGA Controller supports two screen resolution modes, this is controlled by the Control Unit block. The selection between the two resolution modes are selected with a generic parameter named: Resolution320x240. If the parameter value is set to zero, then the 320x240 pixel resolution mode is disabled and the VGA controller uses 640x480 pixel resolution. When the parameter value is set to one, then 320x240 pixel resolution mode is enabled. This is solved with a case statement in the VHDL code, the choice is selected with a generic parameter. Since the generic parameter is set before synthesis, the selected branch in the case statement is kept and the other is deleted by the synthesis tool under optimization, because this branch of the case statement will never be used. See Appendix 234 for the Control Units VHDL code. How the Control Unit handles the two resolution modes and how the Pixel RAM is read in these two resolution modes or cases is described in more detail below.

Case 640x480 pixel resolution mode

When the $RGB\_Data\_en$ signal from the VGA Sync Generator block switches to high, then RGB data needs to be provided to the VGA monitor. This is achieved with the Control Unit block by reading the values from each address of the Pixel RAM and simultaneously outputting the read value from a specific address of the Pixel RAM on the RGB output port. The RGB color value for a pixel needs to be stable approximately 40ns. All this is solved with a counter named Address.Counter, which is clocked with the 24MHz clock named Clk24. The counter increases its value with approximately 42ns delay, until it reaches the value that is equal to the last address of the Pixel RAM. This counter forms the address signal to the Pixel RAM. While the Pixel RAM is being read, a signal named Memory.Busy is set high, this signal indicates that the Pixel RAM is being read and can not be written to at this time, more descriptive information about this signal is discussed in section 4.3.4.2 Storing RGB data to the Pixel RAM.
Case 320x240 pixel resolution mode

In the former case a new pixel was outputted every 42ns when the \( RGB\_Data\_en \) signal was high, yielding a 640x480 pixel screen resolution. A 320x240 pixel resolution will thus result in twice as large pixels. So to produce a virtual 320x240 pixel resolution, every pixel needs to be twice as large in the horizontal and vertical axis (see figure 4.2.2 on page 32) as a pixel was in the former case. This is achievable by reading the RGB data for a pixel twice from each memory address and also reading each line/row in a frame twice from the Pixel RAM. This means, that the \( Address\_Counter \) is now increased not with 42ns delay interval as in the former case but with 84ns interval; producing twice pixel size in the horizontal axis. The following method is used to read the RGB Data for each line/row in a frame twice. The \( Address\_Counter \) counting pattern is modified, so that it increases its value with one, each 84ns as in the former case, but after every second line/row the counters value is decreased by 320, this will result in that the RGB data for every line/row is outputted twice in a frame, see figure below. This all will produce double sized pixels and give 320x240 pixel screen resolution.

Ass figure above shows the counter decreases with the value 320 after every second line/row in a frame, the assumption is made that the editable screen area is set to 640x480 pixels. This results that in a line/row there is totally 320 pixels when 320x240 pixel screen resolution is enabled. For instance if the editable screen area is set to 600x400 pixels instead, then a line/row contains 300 pixels and the \( Address\_Counter \) would decrease with the value 300 after every second line/row. This adaptation is made automatically when the generic parameters \( Screen\_SIZE\_horizontal \) and \( Screen\_SIZE\_vertical \) are set see VHDL solution in Appendix 3 for more detailed explanation on this.
4.3.4.2. Storing a pixel in the Pixel RAM

To store a pixel in the Pixel RAM, the Control Unit needs to obtain information of the RGB value and the coordinate for the pixel. This is provided by the software, which stores these values in the registers. When these values are stored, the VGA Regs block responds to the Control Unit block by setting low a signal named; Write_Done. This is an indication from the VGA Regs block that it has received the RGB value and the coordinate for the pixel from the software. If the Memory_Busy signal is low then the Control Unit stores the RGB value of the pixel in the Pixel RAM, at the address, which corresponds to the coordinate where this pixel should be located at. Figure below illustrates how the address that corresponds to the coordinate is computed.

![Diagram of converting coordinates to the corresponding Pixel RAM address](image)

4.3.4.3. Storing Characters in the Pixel RAM

To store a specific character in the Pixel RAM, the Control Unit needs to know the Character number that defines which character it is, the RGB value for the character, the RGB value for the background of the character and the start coordinate for the character. These values are provided by the software, which stores these values in the registers. When these values are stored, the VGA Regs block responds to the Control Unit block by sending a low pulse on a signal named Char_en, this pulse is one clock cycle wide The storage of a character is performed by a finite state machine (FSM), this FSM is called Char FSM. When the Char FSM detects that the input Char_en switches from one to zero and that the Memory_Busy signal is low, it starts to read the Charater ROM and indicate this by setting a signal named Char_Busy high. The Char_Busy signal is kept high until the whole character is stored in the Pixel RAM. The Char FSM starts by examining the specific characters shape in the character ROM, then according to the shape, color and coordinate saving the character, pixel by pixel in the Pixel RAM. The first pixel is stored at the coordinate provided from the VGA Regs block, which is the start coordinate for the character. For the other pixels that build up the specific character the coordinate is calculated automatically, the start coordinate is used as a reference in this calculation. The storing procedure begins at the start address which is given by the signal named Char_nr,
this signal is provided by the VGA Regs block and defines what specific character should be stored to the Pixel RAM. As mentioned before if the Char_nr is multiplied with the number 8, then it results the start address to a particular character shape in the Character ROM. Each 7bit on a specific Character ROM address is examined separately, which means that the Char_FSM examines only one bit at the time. If the value of a specific bit is equal to one, the pixel that is stored in the Pixel RAM gets the RGB value that is defined for the Character. Otherwise if the value is equal to zero then the pixel gets the RGB value that defines the characters background. When all 7bits on an address is examined, the Char_FSM increases the Chratio ROM address and examines the next 7bits and so on. This procedure continues until the Char_FSM reaches the last address of the specific character. The Char_FSM uses the same method to compute the correct Pixel RAM address as it was discussed in section (4.3.4.2 Storing a pixel in the Pixel RAM). The mentioned storing procedure takes several clock cycles to execute for the Char_FSM which means that it can be interrupted by the VGA Synch Generator block, which sets the RGB_data_en signal high. This indicates that the VGA monitor need to be provided with RGB data, which in turn means that the Pixel RAM needs to be read. As the Pixel RAM can not be read and written to simultaneously, the Char_FSM needs to abort the writing action and wait until the Pixel RAM is available for writing, this indication is made by the signal named Memory_Busy. When the Pixel RAM is once again available for writing then the Char_FSM can carry on writing to the Pixel RAM. This is achieved with a state named Wait_st. When the Char_FSM is interrupted it switches to this state and when the Pixel RAM is no longer busy it switches back to the state where it was before it was interrupted. This can be compared to when a CPU obtains an interrupt request.

### 4.3.5. Registers and the interface to the Avalon data bus

The VGA Regs block acts as the interface to the Avalon data bus, it receives information from the software, which is then stored in registers within the VGA Regs block. The block also provides the rest of the VGA Controller with control signals. As discussed prior, the VGA Regs block also receives signals from the rest of the VGA Controller, these signals are indications to the software. The architecture of the VGA Regs block and the function of these signals are described below.

![Figure 4.3.5a. Entity of the VGA Regs block](image)

The Avalon interface for the VGA Controller requires a single slave port using a small set of Avalon signals, to handle simple read and write transfers, to the registers in the Register block. These signals can be seen on the left side of the VGA Regs block on figure 4.3.5.

- Write enable signal named We_n
- Read enable signal named Re_n
- Address signal
- Write data signal named Write_data and
- Read data signal which is named Read_data
Figure 4.3.5b. Description of the registers

<table>
<thead>
<tr>
<th>Register name</th>
<th>Offset</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_reg</td>
<td>0</td>
<td>Write</td>
<td>Stores the X coordinate for a pixel or a character</td>
</tr>
<tr>
<td>Y_reg</td>
<td>1</td>
<td>Write</td>
<td>Stores the Y coordinate for a pixel or a character</td>
</tr>
<tr>
<td>RGB_reg</td>
<td>2</td>
<td>Write</td>
<td>Stores the RGB value for a pixel or a character</td>
</tr>
<tr>
<td>BG_RGB_reg</td>
<td>3</td>
<td>Write</td>
<td>Stores the RGB value for a character background</td>
</tr>
<tr>
<td>Char_nr_reg</td>
<td>4</td>
<td>Write</td>
<td>Stores a number which corresponds to a specific character</td>
</tr>
<tr>
<td>Write_Done_reg</td>
<td>5</td>
<td>Read/Write</td>
<td>Stores values for communication between HW and SW</td>
</tr>
</tbody>
</table>

Figure above shows how the signals, which run to the Control Unit, are connected to the registers. The blue color filed areas illustrate how many bits of a particular 16bit register is used. The value in Char_nr_reg is 7bits and connects to the signal named Char_nr which is 10bits wide. This may seem strange but in fact the value in Char_nr_reg is left shifted three times before it is assigned to the Char_nr signal. As known, a three times left shifting results a multiplication with eight. This provides the correct start address in the Character ROM, for a specific character defined by the Char_nr signal.
The `write_done_reg` can be written to and read from by the software and the VGA Controller. How the `Write_Done_reg` is used for communication between the VGA Controller and the software is described in the table below.

<table>
<thead>
<tr>
<th>Write_done_reg</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second bit value</td>
<td>First bit value</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 4.3.5d. Write_done_reg description*
4.4. Predefined functions & Device Drives

In this section the software functions that are supported by the VGA Controller are explained, an example is illustrated on how these functions can be used in application development. How these functions communicate with the hardware and access the registers is also explained.

4.4.1. The predefined functions that are supported by the VGA Controller

**Function:**

`print_pix(unsigned int address, unsigned int x, unsigned int y, unsigned int rgb);`

**Function description:**

Prints a pixel with the color rgb at the coordinate (x, y).

**Function:**

`print_hline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);`

**Function description:**

Prints a horizontal line with the color rgb with the length len which starts at the coordinate (x_start, y_start).

**Function:**

`print_vline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);`

**Function description:**

Prints a vertical line with the color rgb with the length len which starts at the coordinate (x_start, y_start).

**Function:**

`print_charsp(unsigned int address, unsigned int x, unsigned int y, unsigned int rgb, unsigned int BG_RGB, unsigned int Charnr);`

**Function description:**

Prints a character with the color rgb and the background color BG_RGB at the coordinate (x, y). The character is selected with the variable Charnr.

**Function:**

`print_char(unsigned int address, unsigned int x, unsigned int y, unsigned int rgb, unsigned int BG_RGB, char Character);`

**Function description:**

Prints the Character with the color rgb and the background color BG_RGB at the coordinate (x, y). some characters are not supported by this function, see Appendix33

**Function:**

`void delay(int t);`

**Function description:**

Delaying with the time-unit t
4.4.1.1. Software example

The simple application below provides an example on how the predefined functions, which are supported by the VGA Controller, may be used in various applications. These functions can also be applied in additional more complex applications, in order to example create more refined animations.

```
#include "altera_aavalon_VGARegs.h"
#include "altera_aavalon_VGRoutines.h"
#include "altera_aavalon_pior_regs.h"
#include "system.h"
#include <stdio.h>

#define base VGA_CONTROLLER_BASE

int main(void)
{
    // Prints the character A, with blue color and green background color
    // with the start coordinate (1, 2)
    print_char(base,1,2,1,2,'A');

    // Prints a horizontal blue line with five pixel length at (11, 2)
    print_hline(base,11,2,5,1);

    // Prints a vertical blue line with six pixel length at (19, 2)
    print_vline(base,19,2,6,1);

    // Prints a red pixel at (11, 5)
    print_pix(base,11,5,4);

    return 0;
}
```

This simple application prints the following items on the VGA monitor screen see Figure below.

![Figure 4.4.1. 1. Graphics output from the simple application](image)
4.4.2 Device Drives

How the previous functions access the registers are described below. First a so called register file is written, this register file declares macros to read and write each register in the Regs Block. The register file named `altera_avalon_VGA_regs.h` for the VGA Controller can be viewed below:

```c
#include <stdio.h>
void print_pix(unsigned int address, unsigned int x, unsigned int y, unsigned int rgb)
{
    unsigned int write_ok = 0;
    do{
        write_ok = IORD_ALTERA_AVALON_VGA_Write_Done_reg(address);
    }while(write_ok != 0x3);
    IOWR_ALTERA_AVALON_VGA_X_reg(address, x);
    IOWR_ALTERA_AVALON_VGA_Y_reg(address, y);
    IOWR_ALTERA_AVALON_VGA_RGB_reg(address, rgb);
    IOWR_ALTERA_AVALON_VGA_Write_Done_reg(address, 2);
}
```

A part of `altera_avalon_VGA_routines.c` it illustrates how the function `print_pix` uses these macros to access the registers. As mentioned this function prints a pixel with a particular color and coordinates. The function starts by reading from the Write_Done_reg, it confirms so that the value of the Write_done_reg is equal to 3, which means that it can start storing the coordinates and RGB color for the pixel. If the `Write_done_reg` is equal to 3 then it stores the RGB value and coordinates for the pixel. When the writing is finished then the function gives a indication to the hardware by setting the last bit in the Write_done_reg to zero.

```c
#include "altera_avalon_VGA_routines.h"
#include <stdio.h>
```
4.5. Verification & Results

As previously mentioned the blocks which make up the VGA Controller were constructed separately, enabling each design to be verified individually. Meaning; as soon as a block construction was completed, its correct functions were verified by the act of simulation. Following simulation, all blocks are connected by a top module named; *VGA_top*. This *VGA_top* module represents the entire VGA Controller hardware architecture, apart from the registers. In order to verify this top module two, level 2, test benches were written in VHDL. Level 2 test benches are able to send input stimuli signals, and also read and verify the output signals accuracy. If an error, during verification should occur, the test bench alerts and reports that particular error with a message. See Appendix 3 for total description of the test benches. However, due to the fact that the character ROM was not yet developed and available, when the test benches were constructed, the test benches do not include them in the simulations. Below several illustrations are sown of the simulation results.

This wave for illustrates the address counters counting pattern 640x480 pixel screen resolution mode. Also the H_sync signal can be observed.

![Wave-form window from simulation. 640*480 pixel resolution mode](image)

**Figure 4.5a.** Wave-form window from simulation. 640*480 pixel resolution mode

This wave illustrates the address counters counting pattern 320x240 pixel screen resolution mode. Also the H_sync signal can be observed.

![Wave-form window from simulation. 320*240 pixel resolution mode](image)

**Figure 4.5b.** Wave-form window from simulation. 320*240 pixel resolution mode
Close-up of the RGB\textunderscore data\textunderscore en signal and the Address\textunderscore counter signal

![Wave-form window from simulation. 320*240 pixel resolution mode](image)

**Figure 4.5c.** Wave-form window from simulation. 320*240 pixel resolution mode

### Synthesis results

FPGA Resource usage after compilation, 320x240 pixel screen resolution mode enabled:

- Total logic elements (LEs): \(486/12,060 (4\%)\)

FPGA Resource usage after compilation, 320x240 pixel screen resolution mode disabled:

- Total logic elements (LEs): \(363/12,060 (4\%)\)

\(F_{\text{max}} = 56.42\text{MHz},\) can vary, depend on pixel 320x240 screen resolution mode disabled or enabled and the editable screen size area selection.
When the register blocks were constructed and the device drives were completely written, the VGA controller component was created by the help of the Component Editor tool. The VGA Controller was integrated into a system, consisting the Nios II CPU, a 4bit input PIO for push-buttons and a external SRAM.

The external memory was used for downloading the SW, since the VGA Controller uses the majority of the FPGA memory resources. As the SOPC Builder is able to automatically generate a test bench and simulation model for the whole system, this procedure was selected, and the system was simulated. Below several illustrations are sown of the simulation results.

Wave-form below illustrates the Character_FSMs behavior when being interrupted by the Memory_busy signal while storing the character shapes in the Pixel RAM. Also the registers can be observed.

![Figure 4.5d. VGA Controller integrated in the system](image)

![Figure 4.5e. Wave-form window from simulation.](image)
Finally, a test application, which displays a simple image on top of the screen, along with a cursor, which can be maneuvered with the four separate push-buttons located on the UP3 development board, were developed.
Chapter 5

Conclusion & Future Work

5.1. Conclusion
The paper has covered and discussed a hardware/software design, implementation, and testing of a basic embedded system. As demonstrated, it is quite obvious that the tools, which support the FPGAs, has reached a high level of maturity. Meaning that reconfigurable technologies and supporting design-tools are finally reaching a level that allows system designers to perform hardware/software co-design of complex systems. Embedded systems can no longer depend on independent hardware or software solutions to real-time problems due to cost, development efficiency, flexibility, upgradeability, and development time. With traditional approaches, system development would require weeks or even months for completion. With the help of today’s design tools, all of this is (literally) achieved in a matter of few minutes. As Altera states: “from concept to systems in minutes”. The first part of the paper covered the design and development of an embedded system, with the help of ready-to-use intellectual property blocks.
The subsequent part presented a VGA controller design, which was independently developed during this thesis work. The development was done with the purpose and objective to give an understanding on how intellectual property (IP) blocks or components are developed and integrated into embedded systems. It will also be applied for educational purposes in digital hardware design classes/seminars given at Mälardalen University. Furthermore, it can be utilized as a demonstrative tool for simple game development. Since the VGA Controller is compatible with the industry VGA standard it is also suitable for graphically displaying data from example industrial processes on a standard monitor. As it is compatible with any standard VGA monitor, it gives a simple graphical user interface when integrated, i.e. in electronic control systems.

5.2. Future Work
As the Quartus II software supports a tool named SignalTap II, and even though it was not included in the thesis, it evoked my curiosity and it was tested. SignalTap II logic analyzer is a second-generation system-level debugging tool that captures and displays real-time signal behavior in a system on a programmable chip.
According to experience, it would be a great tool to learn more about, given the extensive advantages and benefits it provides for future design activities.

The VGA Controller presents a tempting idea in improving and extending the current VGA controller, by connecting a digital to analog converter DAC to the FPGA and extend the bit width of the RGB output from the FPGA, in order for more colors to be supported. This would subsequently demand a larger memory. In order to solve this, an external memory could be used, for example a fast DDR (double data rate) memory. One additional appealing idea is, to make the VGA Controller compatible with another data bus, which supports a faster CPU. In its current state the VGAs performance is mainly set by the Nios II CPU. A faster CPU can efficiently provide the VGA controller with pixel information and is also able to execute complex mathematical algorithms, which in turn would enable the VGA Controller to undertake 3D graphics.
Bibliography

[1] Stefan Sjoholm, Lennart Lindh. VHDL för konstruktion. Tredje upplagan


[5] THE FUTURE OF FIELD-PROGRAMMABLE GATE ARRAYS
Peter Alfke, Xilinx, Inc.,San Jose

http://www.fpga-faq.org/FPGA_Boards.shtml


[8] A Brief Introduction to FPGAs
http://www.xilinx.com/products/software/sysgen/app_docs/user_guide_Chapter_3_Section_1.htm

[9] A Simple VGA Controller for the ALTERA UP1 Board
http://www.eecg.toronto.edu/~singhd/241/vgacon.htm
Appendix 1. The UP3-1C12 education kit

The UP3-1C12 Development Boards features
The following are some of the features of the UP3-1C12 Board.
• Features an Altera EP1C12Q240 Device and EPCS4 configuration device
• Supports intellectual property based (IP-Based) design both with and without a microprocessor
• USB 1.1 (Full speed & Low speed)
• RS 232 Port
• Parallel port (IEEE 1284)
• PS/2 Port
• VGA port
• IDE (Integrated Drive Electronics)
• 2KBytes of I2C PROM(Expandable)
• 128KBytes of SRAM
• 2MBytes of FLASH
• 8MByte SDRAM
• Supports multiple clocks like CPU clock, USB clock, PCI clock, and IOAPIC clock.
• JTAG and Active Serial download capability
• 5V Santa Cruz Expansion Card Header provides 42 I/Os for the development of additional boards providing various functionalities plus 20 additional user I/O pins are provided giving total of 62 user definable I/O pins
• One user definable 4-bit switch block
• Four user definable push button switches, and one global reset switch
• Four user definable LEDs
• One 16X2 character LCD Module
• I2C Real Time Clock
Appendix 2: Source codes for chapter 3

//************************************************************************
//************* Lab1a , Development of a simple HW/SW platform *************
******************************************************************************

#include <stdio.h>

// Delay function
void my_sleep(unsigned int seconds) {
    int i = 0;
    unsigned int delay = 350000;
    for (i=0; i < (seconds * delay); i++) {
    }
}

/*------------ Main ----------------*/
main() {
    int g=1;
    printf(" ** Main Starts **\n ");
    while (1) {
        printf(" ####### Korning nr %d #######\n", g);
        printf("# #\n");
        printf("# Hejsan Kursdeltagare #\n");
        printf("# #\n");
        printf("# #\n");
        printf("###\n");
        printf("\n");
        printf("\n");
        printf("\n");
        g++;
        my_sleep(1); // One second delay
    }
}

******************************************************************************

******* Lab1b, Adding debugging support to the platform **************

#include <stdio.h>

void my_sleep(unsigned int seconds) {
    int i = 0;
    unsigned int delay = 900000;
    for (i=0; i < (seconds * delay); i++) {
    }
}

/*------------ Main ----------------*/
int main() {
    int g;
    printf(" ** Main Starts **\n ");
    while (1) {
        for(g = 1; g < 11; g++) {
            printf("#\d\n", g);
            my_sleep(1);
        }
        printf("\n");
    }
    return 0;
}
//****************************************************************************
//************************ Lab2, Time Analyze method **************************
#include <stdio.h>
#include "sys/alt_timestamp.h"

int main()
{
    int Time = 0, Time_tot = 0;
    int Offset = 0, Freq = 0;
    int value_a = 0;
    int sum = 0;

    alt_timestamp_start();
    Freq = alt_timestamp_freq();
    printf("** The Clock frequency is %dMHz    **\n", (Freq/1000000));
    printf("-----------------------------------------------\n");
    alt_timestamp_start();
    Offset = alt_timestamp();
    printf("** Calculation for execution 'value_a = 5;' **\n
    ");
    alt_timestamp_start();
    value_a = 5;
    Time = alt_timestamp();
    Time_tot = Time - Offset;
    printf("Time = %d\n",Time);
    printf(" Offset = %d\n", Offset);
    printf(" So the total time for value_a = 5; is Time - Offset = %d clock
    
    ");
    alt_timestamp_start();
    printf("-------------------------\n ");
    printf("** Calculation for execution 'sum = sum +1;' **\n
    ");
    alt_timestamp_start();
    sum = sum + 1;
    Time = alt_timestamp();
    Time_tot = Time - Offset;
    printf("Time = %d\n",Time);
    printf(" Offset = %d\n", Offset);
    printf(" So the total time for 'sum = sum +1;' is Time - Offset = %d clock
    
    ");
    alt_timestamp_start();
    printf("-------------------------\n ");
    printf("** Calculation for execution 'sum = sum -1;' **\n
    ");
    alt_timestamp_start();
    sum = sum - 1;
    Time = alt_timestamp();
    Time_tot = Time - Offset;
    printf("Time = %d\n",Time);
    printf(" Offset = %d\n", Offset);
    printf(" So the total time for 'sum = sum -1;' is Time - Offset = %d clock
    
    ");
    alt_timestamp_start();
    printf("-------------------------\n ");
    printf("** Calculation for execution print 'hi' **\n
    ");
    alt_timestamp_start();
    printf("hi");
    Time = alt_timestamp();
    Time_tot = Time - Offset;
    printf("\n Time = %d\n",Time);
    printf(" Offset = %d\n", Offset);
    printf(" So the total time to print 'hi' is Time - Offset = %d clock
    
    ");
    alt_timestamp_start();
    printf("-------------------------\n ");
    printf("** Calculation for execution print 'hej' **\n
    ");
    alt_timestamp_start();
    printf("hej");
    Time = alt_timestamp();
    Time_tot = Time - Offset;
    printf("\n Time = %d\n",Time);
    printf(" Offset = %d\n", Offset);
    printf(" So the total time to print 'hej' is Time - Offset = %d clock
    
    ");

    return 0;
}
/**************************************************************************
//*************** Lab3, Controlling LEDs with a pushbutton  *******************/
#include <io.h>
#include <system.h>
#include <stdio.h>
#include "sys/alt_timestamp.h"
// The delay function is used to make a slow sample of the button pio
void delay(void)
{
    int i=0;
    for(i=0; i < 100000; i++)
    {
    }
}
int main(void)
{
    int ledpio = 0;
    int buttonpio = 0;
    int Offset, Time;
    int Time_tot;
    // Finds out the offset value
    alt_timestamp_start();
    Offset = alt_timestamp();
    while(1)
    {
        // Starts/restarts time measurement
        alt_timestamp_start();
        // Takes a sample from the button pio
        buttonpio = IORD_8DIRECT(BUTTON_PIO_BASE, 0);
        // If the button is pressed increase the Led pio by one
        if(buttonpio == 0)
        {
            // Light the Leds
            IOWR_8DIRECT(LED_PIO_BASE , 0, ledpio);
            ledpio++;
            // Reads the timer value and prints it on the console
            Time = alt_timestamp();
            Time_tot = Time - Offset;
            printf("Execution time = %d\n",Time_tot);
            // Reset the ledpio when it reaches 16
            if(ledpio == 16)
            {
                ledpio = 0;
            }
    delay();
    // Wait untill the button is released
    while(buttonpio == 0)
    {
        buttonpio = IORD_8DIRECT(BUTTON_PIO_BASE, 0);
    }
    }    }
    return 0;
}
```c
#include <io.h>
#include <system.h>
#include <stdio.h>

int main(void)
{
    int ledsin = 0;
    int temp;
    // Takes a sample from the Led in Pio
    ledsin = IORD_8DIRECT(LED_IN_PIO_BASE, 0);
    // Prints the value on the console
    printf("****(%d)****\n", ledsin);
    while(1)
    {
        // This is to determine the change on the Led in Pio input
        temp = ledsin;
        // Takes a sample from the Led in Pio
        ledsin = IORD_8DIRECT(LED_IN_PIO_BASE, 0);
        // Checks if a change has accured.
        if(temp != ledsin)
        {
            // Prints the value on the console
            printf("****(%d)****\n", ledsin);
        }
    }
    return 0;
}
```

```c
#include <io.h>
#include "system_niosII.h"
#include "altera_avalon_sierra_regs.h"
#include "altera_avalon_sierra_ker.h"
#include "altera_avalon_sierra_io.h"
#include "sys/alt_timestamp.h"

// TASK
#define IDLE 0
#define Kalle_id 1
#define Tommy_id 2

// STATES
#define BLOCKED 0
#define READY 1

// SEMAPHORES
#define PRINT_SEM 1

// TASK STACKS
#define STACK_SIZE 400

char stack1[STACK_SIZE];
char stack2[STACK_SIZE];
char idle_stack[STACK_SIZE];

// GLOBAL VARIABLES
int ledsin = 0;
int temp;
// These variables are used to measure the time for semaphore release
int Offset, Time;
int Time_tot;
volatile unsigned int first_csw=0;

// Register Sierra HW IRQ with ISR (HW_CTX_SWTICH in csw.S)
#ifdef SIERRA_BASE
static void init_Sierra_interrupt()
{
    int ret=0xffff;
    ret = alt_irq_register( SIERRA_IRQ, NULL, HW_CTX_SWITCH );
}
#endif
```
/************** Tasks ************
// Prio 2, BLOCKED
void Tommy(void)
{
    init_period_time(100);
    printf("Tommy starts\n");
    task_start(Kalle_id);
    // Takes a sample from the Led_in_Pio
    ledsin = IORD_8DIRECT(LED_INPIO_BASE, 0);
    while(1)
    {
        sem_take(PRINT_SEM);
        printf("Tommy runs\n");
        // This is to determine the change on the Led_in_Pio input
        temp = ledsin;
        // Takes a sample from the Led in Pio
        ledsin = IORD_8DIRECT(LED_INPIO_BASE, 0);
        // Checks if a change has accured.
        if(temp != ledsin)
        {
            printf("Press %d\n", ledsin);
        }
        sem_release(PRINT_SEM);
        wait_for_next_period();
    }
}
// Prio 3, BLOCKED
void Kalle(void)
{
    init_period_time(200);
    // Finds out the offset value
    alt_timestamp_start();
    Offset = alt_timestamp();
    printf("Kalle starts\n");
    while(1)
    {
        sem_take(PRINT_SEM);
        printf("Kalle runs\n");
        // Starts the counter before semaphore release
        alt_timestamp_start();
        sem_release(PRINT_SEM);
        // Reads the timer value and prints it on the console
        Time = alt_timestamp();
        Time_tot = Time - Offset;
        printf("Sem release time = %d, Time_tot = %d\n", Time, Time_tot);
        wait_for_next_period();
    }
}
void idle(void)
{
    int i;
    printf("Idle st\n");
    task_start(Tommy_id);
    while(1){
        for(i=0; i<50000; i++);
    }
    printf("Shouldn't be here!\n");
}
/** ********** Main - test system ***********/
void main(void)
{
    DECLARE_VARIABLESUSEDONLYINMAIN */
    #ifdef SIERRA_BASE
    init_SierraInterrupt();
    #endif
    printf("Shouldn't be here!\n");
}
* Initialize time base register.
  * This example     : 48 MHz system-clock
  * Wanted tick time : 1 ms
  * Formula gives    : 1 ms x 48 MHz / 1000 => 48(dec) or 30(hex)
***************************************************************************/
SierraTime_base_reg_init(0x30);
print_SierraTime_base_reg();
print_SierraVersion_reg();
// Initialize OS internals
os_init();
// TaskID=1, Prio=3, State=BLOCKED
task_create(Kalle_id, 3, BLOCKED, Kalle, stack1, STACK_SIZE);
// TaskID=2, Prio=2, State=BLOCKED
task_create(Tommy_id, 2, BLOCKED, Tommy, stack2, STACK_SIZE);
// TaskID=0, Prio=7, State=ready
task_create(IDLE, 7, READY, idle, idle_stack, STACK_SIZE);
tsw_on();
  while(1) {
    printf("Shouldn't be here!\n");
  }
}[/code]

```c
#include <io.h>
#include "system_niosII.h"
#include "altera_avalon_sierra_regs.h"
#include "altera_avalon_sierra_ker.h"
#include "altera_avalon_sierra_io.h"

// TASK
#define IDLE 0
#define Kalle_id 1
#define Tommy_id 2
#define SRAM_TEST_id 3

// STATES
#define BLOCKED 0
#define READY 1

// SEMAPHORES
#define PRINT_SEM 1

// TASK STACKS
#define STACK_SIZE 400
char stack1[STACK_SIZE];
char stack2[STACK_SIZE];
char stack3[STACK_SIZE];
char idle_stack[STACK_SIZE];

// GLOBAL VARIABLES
int buttonpio = 0;
int ledsin = 0;
int temp;

volatile unsigned int first_csw=0;

// Register Sierra HW IRQ with ISR (HW_CTX_SWTICH in csw.S)
#ifdef SIERRA_BASE
static void init_Sierra_interrupt()
{
  int ret=0xffff;
  ret = alt_irq_register( SIERRA_IRQ, NULL, HW_CTX_SWITCH );
}
#endif

//*************** SRAM Test function *********************
void RAM_test(void) {
```
int memory_base = EXT_SRAM_BASE;
int mem_size = 0x10000;
int value;
unsigned int offset;
unsigned int pattern;
int status;

printf("****Memory test starting*****\n");
printf("Writing test pattern to memory...\n");

//Fill memory with a known pattern.
for (pattern = 1, offset = 0; offset < mem_size; pattern++, offset+=4)
{
    IOWR_32DIRECT(memory_base, offset, pattern); // Write pattern to the specific address
}
printf("Write done\nStarting verifying process...\n");

//Check each location, if it is the same as the test pattern.
for (pattern = 1, offset = 0; offset < mem_size; pattern++, offset+=4)
{
    value = IORD_32DIRECT(memory_base, offset);
    if(pattern != value)
    {
        status = 1;
        break;
    }
    else
    {
        status = 0;
    }
}
printf("Verifying process done\n");
if(status == 1)
{
    printf("Memory test FAILED!!\n");
}
else
{
    printf("Memory test completed successfully\n");
}

/****************** Tasks *****************
// Prio 2, BLOCKED
void Tommy(void)
{
    init_period_time(100);
    printf("Tommy starts\n");
task_start(Kalle_id);
    // Takes a sample from the Led in Pio
    ledsin = IORD_8DIRECT(LED_INPIO_BASE, 0);
    while(1)
    {
        sem_take(PRINT_SEM);
        printf("Tommy Runs\n");
        // This is to determine the change on the Led in Pio input
        temp = ledsin;
        // Takes a sample from the Led in Pio
        ledsin = IORD_8DIRECT(LED_INPIO_BASE, 0);
        // Checks if a change has occurred.
        if(temp != ledsin)
        {
            // Prints the value on the console
            printf("Press nr%d\n", ledsin);
        }
        sem_release(PRINT_SEM);
        wait_for_next_period();
    }
    /* Should never end up here */
    printf("Shouldn't be here!2\n");
}

// Prio 3, BLOCKED
void Kalle(void)
{init_period_time(200);
printf("Kalle starts\n");

while(1)
{
    sem_take(PRINT_SEM);
    printf("Kalle runs\n");
    sem_release(PRINT_SEM);
    wait_for_next_period();
} /* Should never end up here */
printf("Shouldn't be here!3\n");}

// Prio 4, BLOCKED
void SRAM_TEST(void)
{
    init_period_time(100);
    printf("SRAM_TEST starts\n");
task_start(Tommy_id);

    while(1)
    {
        sem_take(PRINT_SEM);
        printf("SRAM_TEST runs\n");
// Takes a sample from the button pio
        buttonpio = IORD(BUTTON_PIO_BASE, 0);
        if(buttonpio == 0)
        {
// Run SRAM Test Function
            RAM_test();
        }
        // Wait until the button is released
        while(buttonpio == 0)
        {
            buttonpio = IORD(BUTTON_PIO_BASE, 0);
        }
        sem_release(PRINT_SEM);
        wait_for_next_period();
} /* Should never end up here */
printf("Shouldn't be here!4\n");}

void idle(void)
{
    int i;
    printf("Idle st\n");
task_start(SRAM_TEST_id);

    while(1){
        for(i=0; i<50000; i++);
    } /* Should never end up here */
    printf("Shouldn't be here!1\n");
}

//*****************************************************************************
//********* Main - test system *************
void main(void)
{
    /* Declare variables used only in main() */

    #ifdef SIERRA_BASE
    init_Sierra_interrupt();
    #endif

    /***************************************************************
    * Initialize time base register.
    * This example : 48 MHz system-clock
    * Wanted tick time : 1 ms
    }
* Formula gives: 1 ms x 48 MHz / 1000 -> 48 (dec) or 30 (hex)
**************************************************************/
SierraTime_base_reg_init(0x30);
print_SierraTime_base_reg();
os_init();

// TaskID=3, Prio=1, State=BLOCKED
task_create(SRAM_TEST_id, 1, BLOCKED, SRAM_TEST, stack3, STACK_SIZE);

// TaskID=1, Prio=3, State=BLOCKED
task_create(Kalle_id, 3, BLOCKED, Kalle, stack1, STACK_SIZE);

// TaskID=2, Prio=2, State=BLOCKED
task_create(Tommy_id, 2, BLOCKED, Tommy, stack2, STACK_SIZE);

// TaskID=0, Prio=7, State=ready
task_create(IDLE, 7, READY, idle, idle_stack, STACK_SIZE);

tsw_on();
    while (1)
    {
        printf("Shouldn't be here!\n");
    }
}
VHDL source code for the Led_function in chapter 3

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY OnePulse IS
  GENERIC (filt_cnt_width  : POSITIVE := 23;
            filt_cnt_max_value : POSITIVE := 4800000);
  PORT(Clk, Reset_n, Ctr_pio : IN STD_LOGIC; -- Ctr_pio is the input from the Push-Button
       Leds    : OUT STD_LOGIC_VECTOR(3 downto 0));-- Output to the LEDs
END OnePulse;

ARCHITECTURE rtl OF OnePulse IS
  SIGNAL Count_up, filt_cnt_start : STD_LOGIC;
  SIGNAL filt_cnt   : STD_LOGIC_VECTOR(filt_cnt_width-1 downto 0);
  SIGNAL Ctr_pio_int1, Ctr_pio_int2 : STD_LOGIC;
  SIGNAL LED_cnt    : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
  One_Pulse: PROCESS(Clk, Reset_n)
  BEGIN
    IF Reset_n = '0' THEN
      Ctr_pio_int1 <= '0';
      Ctr_pio_int2 <= '0';
      Count_up <= '0';
    ELSIF Clk'event and Clk = '1' THEN
      Ctr_pio_int1 <= Ctr_pio;
      Ctr_pio_int2 <= Ctr_pio_int1;
      IF filt_cnt = 0 THEN
        Count_up <= ((NOT Ctr_pio_int2) AND (Ctr_pio_int1));
      END IF;
    END IF;
  END PROCESS;
END rtl;
----- This Process makes sure that once a pulse on the signal named "Count_up" has been -----
--- it can not be generated again between a time interval set by the filt_cnt. This filters ---
------- out any ripples that may additionally occur when a Push-Button is pressed down -----
---------------------------------------------------------------------------------------------

Filter_Counter: PROCESS(Clk, Reset_n)
BEGIN
  IF Reset_n = '0' THEN
    filt_cnt_start <= '0';
    filt_cnt <= (Others => '0');
  ELSIF Clk'event and Clk = '1' THEN
    IF Count_up = '1' THEN
      filt_cnt_start <= '1';
    END IF;
    IF filt_cnt_start = '1' THEN
      IF filt_cnt < filt_cnt_max_value THEN
        filt_cnt <= filt_cnt + 1;
      ELSE
        filt_cnt <= (Others => '0');
        filt_cnt_start <= '0';
      END IF;
    END IF;
  END IF;
END PROCESS;

Counter: PROCESS(Clk, Reset_n)
BEGIN
  IF Reset_n = '0' THEN
    LED_cnt <= (Others => '0');
  ELSIF Clk'event and Clk = '1' THEN
    IF Count_up = '1' THEN
      IF LED_cnt < 15 THEN
        LED_cnt <= LED_cnt + 1;
      ELSE
        LED_cnt <= (Others => '0');
      END IF;
    END IF;
  END IF;
END PROCESS;

Leds <= LED_cnt;
END rtl;
Appendix 3: Source codes for the VGA Controller in chapter 4

VHDL source codes for the VGA Controller

-- File Name : VGA_interface.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY VGA_interface IS
  -- Screen_SIZE_horizontal => Sets the horizontal size of the editable screen area
  -- Screen_SIZE_vertical => Sets the vertical size of the editable screen area
  -- RAM_SIZE => Sets the memory size that will be used,
  -- RAM_ADDRESS_SIZE => Memory address bus width
  -- Resolution320x240 => Selects screen resolution, 0 for 640x480 pixels and 1 for 320x240 pixels

  GENERIC (Screen_SIZE_horizontal  : POSITIVE := 600;
           Screen_SIZE_vertical  : POSITIVE := 400;
           RAM_SIZE    : POSITIVE := 60000;
           RAM_ADDRESS_SIZE  : POSITIVE := 16;
           Resolution320x240  : INTEGER := 1);

  PORT(Clk, Reset_n   : IN STD_LOGIC;
       We_n, Re_n   : IN STD_LOGIC;  -- Avalon Write and Read enable signals
       Write_Data   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);  -- Avalon Writedata signal
       Address   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);   -- Avalon Address signal
       Read_Data      : OUT STD_LOGIC_VECTOR(15 DOWNTO 0); -- Avalon Readdata signal
       H_sync, V_sync    : OUT STD_LOGIC;                     -- Sync signals to the VGA monitor
       RGB     : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)); -- RGB signal to the VGA monitor

END VGA_interface;

ARCHITECTURE rtl OF VGA_interface IS

SIGNAL Memory_Busy_sig, Char_busy_sig  : STD_LOGIC;
SIGNAL Char_en_sig, Write_Done_sig  : STD_LOGIC;
SIGNAL RGB_reg_out_sig, BG_RGB_reg_out_sig : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL X_reg_out_sig, Y_reg_out_sig : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL Char_nr_reg_out_sig   : STD_LOGIC_VECTOR(9 DOWNTO 0);

COMPONENT VGA_regs

PORT(Clk, Reset_n   : IN STD_LOGIC;
       Memory_Busy, Char_busy
       We_n, Re_n   : IN STD_LOGIC;-- From the VGA_Top
       Write_Data   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);-- Avalon Writedata signal
       Address   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);-- Avalon Address signal
       Write_Done, Char_en
       RGB_reg_out, BG_RGB_reg_out
       X_reg_out, Y_reg_out, Char_nr_reg_out: OUT STD_LOGIC_VECTOR(9 DOWNTO 0);-- Output signals to the VGA_Top
       Read_Data           : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));-- Avalon Readdata signal

END COMPONENT;

COMPONENT VGA_Top

GENERIC (Screen_SIZE_horizontal   : POSITIVE;
          Screen_SIZE_vertical     : POSITIVE;
          RAM_SIZE    : POSITIVE;
          RAM_ADDRESS_SIZE  : POSITIVE;
          Resolution320x240  : INTEGER);

END COMPONENT;

END VGA_interface;
PORT(Clk, Reset_n, Pix_RAM_we_n : IN STD_LOGIC;
    Char_en                      : IN STD_LOGIC;
    BG_RGB_Data_in, RGB_in      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    Char_nr                  : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    X_in, Y_in          : IN STD_LOGIC_VECTOR(9 downto 0);
    H_sync, V_sync  : OUT STD_LOGIC;
    Memory_Busy, Char_busy : OUT STD_LOGIC;
    RGB                  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END COMPONENT;

----------------------------------------------------------------------------
BEGIN

REGISTERS : VGA_regs PORT MAP(Clk, Reset_n, Memory_Busy_sig, Char_busy_sig, We_n,
    Re_n, Write_Data, Address, Write_Done_sig, Char_en_sig,
    RGB_reg_out_sig, BG_RGB_reg_out_sig, X_reg_out_sig,
    Y_reg_out_sig, Char_nr_reg_out_sig, Read_Data);

VGA : VGA_Top GENERIC MAP(Screen_SIZE_horizontal, Screen_SIZE_vertical,
    RAM_SIZE, RAM_ADDRESS_SIZE, Resolution320x240)
    PORT MAP(Clk, Reset_n, Write_Done_sig, Char_en_sig, BG_RGB_reg_out_sig,
        RGB_reg_out_sig, Char_nr_reg_out_sig, X_reg_out_sig, Y_reg_out_sig,
        H_sync, V_sync, Memory_Busy_sig, Char_busy_sig, RGB);

----------------------------------------------------------------------------
END rtl;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY VGA_regs IS
    PORT(Clk, Reset_n             : IN STD_LOGIC;
         Memory_Busy, Char_busy   : IN STD_LOGIC; -- From the VGA_Top
         We_n, Re_n               : IN STD_LOGIC; -- Avalon Write and Read enable signals
         Write_Data               : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- Avalon Write data signal
         Address                  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);  -- Avalon Address signal
         Write_Done, Char_en      : OUT STD_LOGIC; -- Output signal to the VGA_Top
         RGB_reg_out, BG_RGB_reg_out  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0); -- Output signal to the VGA_Top
         X_reg_out, Y_reg_out, Char_nr_reg_out : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);  -- Output signals to the VGA_Top
         Read_Data           : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));-- Avalon Read data signal
    END VGA_regs;

ARCHITECTURE rtl OF VGA_regs IS
----------------------------- Registers ----------------------------------
SIGNAL X_reg             : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL Y_reg             : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL RGB_reg           : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL BG_RGB_reg        : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL Char_nr_reg       : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL Write_Done_reg    : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL event_c, Char_en_d  : STD_LOGIC;

BEGIN
    PROCESS(Clk, Reset_n)
    BEGIN
        IF Reset_n = '0' THEN
            Read_Data <= (Others => '0');
            X_reg <= (Others => '0');
            Y_reg <= (Others => '0');
            RGB_reg <= (Others => '0');
            BG_RGB_reg <= (Others => '0');
            Char_nr_reg <= (Others => '0');
            Write_Done <= '1';
            event_c <= '0';
            Char_en_d <= '1';
            Write_Done_reg <= "11";
        ELSIF Clk'event AND Clk = '1' THEN
            IF (We_n = '0') AND (Re_n = '1') THEN
                CASE Address is
                WHEN "000" => X_reg <= Write_Data(9 DOWNTO 0);
                WHEN "001" => Y_reg <= Write_Data(9 DOWNTO 0);
                WHEN "010" => RGB_reg <= Write_Data(2 DOWNTO 0);
                WHEN "011" => BG_RGB_reg <= Write_Data(2 DOWNTO 0);
                WHEN "100" => Char_nr_reg <= Write_Data(9 DOWNTO 0);
                WHEN Others => Write_Done_reg <= Write_Data(1 DOWNTO 0);
                END CASE;
            ELSIF (We_n = '1') AND (Re_n = '0') THEN
                CASE Address is
                WHEN "000" => Read_Data <= (Others => '-');
                WHEN "001" => Read_Data <= (Others => '-');
                WHEN "010" => Read_Data <= (Others => '-');
                WHEN "011" => Read_Data <= (Others => '-');
                WHEN "100" => Read_Data <= (Others => '-');
                WHEN Others => Read_Data(15 DOWNTO 0) <= Write_Done_reg;
                Read_Data(15 DOWNTO 2) <= (Others => '-');
            END CASE;
        END IF;
    END PROCESS;
END rtl;
ELSE
    IF Memory_BUSY = '0' THEN
        IF Char_busy = '0' THEN
            Write.Done <= Write.Done_reg(0);
            IF Write.Done_reg(0) = '0' THEN
                Write.Done_reg(0) <= '1';
            END IF;
            IF Write.Done_reg(1) = '0' THEN
                IF event_c = '0' THEN
                    IF Char_en_d = '1' THEN
                        Char_en_d <= '0';
                        event_c <= '1';
                    END IF;
                    ELSE
                        IF Char_en_d = '1' THEN
                            event_c <= '0';
                            Write.Done_reg(1) <= '1';
                        END IF;
                    END IF;
                ELSE
                    Char_en_d <= '1';
                    Write.Done_reg(1) <= '0';
                END IF;
            END IF;
        END IF;
    ELSE
        Write.Done <= '1';
    END IF;
END IF;
END PROCESS;
----------------------------------------------------------------------------------------
Char_en <= Char_en_d;
X_reg_out <= X_reg;
Y_reg_out <= Y_reg;
RGB_reg_out <= RGB_reg;
BG_RGB_reg_out <= BG_RGB_reg;
Char_nr_reg_out(9 DOWNTO 3) <= Char_nr_reg(6 DOWNTO 0);
Char_nr_reg_out(2 DOWNTO 0) <= (Others => '0');
----------------------------------------------------------------------------------------
END rtl;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY VGA_Top IS
  -- Screen_SIZE_horizontal  => Sets the horizontal size of the editable screen area
  -- Screen_SIZE_vertical => Sets the vertical size of the editable screen area
  -- RAM_SIZE   => Sets the memory size that will be used,
  -- RAM_ADDRESS_SIZE  => Memory adddress bus width
  -- Resolution320x240  => Selects screen resolution, 0 for 640x480 pixels and 1 for 320x240 pixels
  GENERIC (Screen_SIZE_horizontal  : POSITIVE := 600;
    Screen_SIZE_vertical    : POSITIVE := 400;
    RAM_SIZE    : POSITIVE := 60000;
    RAM_ADDRESS_SIZE  : POSITIVE := 16;
    Resolution320x240  : INTEGER := 1);

  PORT(Clk, Reset_n, RAM_we_n : IN STD_LOGIC;
    Char_en   : IN STD_LOGIC;
    BG_RGB_Data_in, RGB_in : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    Char_nr   : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    X_in, Y_in         : IN STD_LOGIC_VECTOR(9 downto 0);
    H_sync, V_sync    : OUT STD_LOGIC;
    Memory_Busy, Char_busy     : OUT STD_LOGIC;
    RGB     : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END VGA_Top;

ARCHITECTURE rtl OF VGA_Top IS
  SIGNAL RAM_we_n_out_sig     : STD_LOGIC;
  SIGNAL RGB_data_to_RAM_sig      : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL RAM_Dout     : STD_LOGIC_VECTOR(2  DOWNTO 0);
  SIGNAL ROM_Data_sig             : STD_LOGIC_VECTOR(6  DOWNTO 0);
  SIGNAL ROM_Address_sig          : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL RAM_addr    : STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE-1) DOWNTO 0);

COMPONENT PIX_RAM
  GENERIC (RAM_SIZE            : positive;
            RAM_ADDRESS_SIZE    : positive);

  PORT(Clk, we_n : IN STD_LOGIC;
    din   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    addr              : IN STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE-1) DOWNTO 0);
    dout  : OUT STD_LOGIC_VECTOR(2  DOWNTO 0));
END COMPONENT;

COMPONENT Char_ROM
  GENERIC (ROM_SIZE           : positive := 728;
            ROM_ADDRESS_SIZE      : positive := 10);

  PORT(Clk  : IN STD_LOGIC;
    addr   : IN STD_LOGIC_VECTOR((ROM_ADDRESS_SIZE -1) DOWNTO 0);
    dout  : OUT STD_LOGIC_VECTOR(6  DOWNTO 0));
END COMPONENT;
COMPONENT VGA_core
GENERIC (Screen_SIZE_horizontal  : POSITIVE;
    Screen_SIZE_vertical    : POSITIVE;
    RAM_SIZE    : POSITIVE;
    RAM_ADDRESS_SIZE  : POSITIVE;
    Resolution320x240  : INTEGER);

PORT(Clk, Reset_n, RAM_we_n     : IN STD_LOGIC;
    Char_en     : IN STD_LOGIC;
    Char_nr     : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    ROM_Data                  : IN STD_LOGIC_VECTOR(6  DOWNTO 0);
    BG_RGB_Data_in, RGB_Data_from_RAM  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    X_in, Y_in            : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    RAM_we_n_out                    : OUT STD_LOGIC;
    H_sync, V_sync     : OUT STD_LOGIC;
    Memory_Busy, Char_busy    : OUT STD_LOGIC;
    RGB_Data_to_RAM     : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    RGB          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    ROM_Address                      : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
    RAM_Address     : OUT STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE-1) DOWNTO 0));

END COMPONENT;

-----------------------------------------------------------------------------
BEGIN
VIDEO_BUFFER : PIX_RAM GENERIC MAP(RAM_SIZE, RAM_ADDRESS_SIZE)
    PORT MAP(Clk, RAM_we_n_out_sig, RGB_data_to_RAM_sig, RAM_addr, RAM_Dout);

CHARACTER_ROM : Char_ROM PORT MAP(Clk, ROM_Address_sig, ROM_Data_sig);

VGA_SYNC_AND_CONTROL_BLOCK : VGA_core GENERIC MAP(Screen_SIZE_horizontal, Screen_SIZE_vertical,
    RAM_SIZE, RAM_ADDRESS_SIZE, Resolution320x240)
    PORT MAP(Clk, Reset_n, RAM_we_n, Char_en, Char_nr, ROM_Data_sig,
        BG_RGB_Data_in, RAM_Dout, RGB_in, X_in, Y_in,
        Pix_RAM_we_n_out_sig, H_sync, V_sync, Memory_Busy,
        Char_busy, RGB_data_to_RAM_sig, RGB, ROM_Address_sig, RAM_addr);

-----------------------------------------------------------------------------
END rtl;
"0000010", "0000010", "0111010", "1000110", "1000010", "0111010", "0000000", -- q
"1000000", "1000000", "1100110", "1000110", "1100110", "1000110", "0000000", -- p
"0000000", "0011100", "0100010", "0100010", "0100010", "0011100", "0000000", -- o
"0000000", "1001001", "1001001", "1001001", "1001001", "1101001", "1011110", "0000000", -- n
"0000000", "0010000", "0010000", "0010000", "0010000", "0010000", "0000000", -- m
"0000000", "0111000", "0100010", "0100010", "0100010", "0100010", "0111000", -- l
"0000000", "0100000", "0010000", "0001000", "0001000", "0001000", "0000000", -- k
"0000000", "0001000", "0001000", "0001000", "0001000", "0001000", "0001000", -- j
"0000000", "1000110", "1000110", "1000110", "1000110", "1000110", "1000110", -- i
"0000000", "1000010", "0010000", "0010000", "0010000", "0010000", "0010000", -- h
"1111100", "0111100", "0111100", "0111100", "0111100", "0111100", "0111100", -- g
"0000000", "0010000", "0010000", "0010000", "0010000", "0010000", "0010000", -- f
"0000000", "1111110", "1111110", "1111110", "1111110", "1111110", "1111110", -- e
"0000000", "0111010", "1000110", "1000010", "0111010", "0000010", "0000010", -- d
"0000000", "0111010", "1000110", "1000010", "1100010", "0111100", "0000000", -- c
"0000000", "1011100", "1100010", "1100010", "1100010", "1100010", "1100010", -- b
"0000000", "0111010", "1000110", "1000010", "0111110", "1000010", "0111110", -- a
"0000000", "1111110", "0100000", "0010000", "0001000", "0001000", "1111110", -- Z
"0000000", "0001000", "0001000", "0001000", "0001000", "0001000", "1000000", -- Y
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- X
"0000000", "0100100", "0100100", "0100100", "0100100", "0100100", "0100100", -- W
"0000000", "0011000", "0111000", "0111000", "0111000", "0111000", "0111000", -- V
"0000000", "0010000", "0010000", "0010000", "0010000", "0010000", "0010000", -- U
"0000000", "0111100", "1000010", "1000010", "1000010", "1000010", "1000010", -- T
"0000000", "1111110", "1000010", "1000110", "1001010", "1001010", "1000010", -- S
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- R
"0000000", "1111100", "1000000", "1000000", "1111000", "1000000", "1000000", -- Q
"0000000", "0111000", "0111000", "0111000", "0111000", "0111000", "0111000", -- P
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- O
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- N
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- M
"0000000", "0111100", "0100000", "0100000", "0100000", "0100000", "0111000", -- L
"0000000", "1000010", "1000010", "1000010", "1000010", "1000010", "1000010", -- K
"0000000", "0110000", "0110000", "0110000", "0110000", "0110000", "0110000", -- J
"0000000", "0111000", "0010000", "0010000", "0010000", "0010000", "0010000", -- I
"0000000", "0100000", "0100000", "0100000", "0100000", "0100000", "0100000", -- H
"0000000", "1000110", "1000110", "1000110", "1000110", "1000110", "1000110", -- G
"0000000", "1000000", "1000000", "1000000", "1000000", "1000000", "1000000", -- F
"0000000", "1111110", "1111110", "1111110", "1111110", "1111110", "1111110", -- E
"0000000", "1111110", "1111110", "1111110", "1111110", "1111110", "1111110", -- D
"0000000", "0011000", "0011000", "0011000", "0011000", "0011000", "0011000", -- C
"0000000", "1111100", "1111100", "1111100", "1111100", "1111100", "1111100", -- B
"0000000", "1000000", "1000000", "1000000", "1000000", "0110000", "0011000", -- A addr 0

BEGIN
PROCESS(Clk)
BEGIN
  IF Clk'event AND Clk = '1' THEN
    dout <= ROM(conv_integer(addr));
  END IF;
END PROCESS;
END rtl;
-- File Name : PIX_RAM.vhd
-------------------------------------------------------------------------------
-- Synthesis Level : Rtl VHDL
-------------------------------------------------------------------------------
-- Revisions List :
-- Author Date
-- Zoltan Nagy 2005-11-09
-------------------------------------------------------------------------------
-------- X*3bit gnereric fully synchronous RAM ----------------------------
-------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY PIX_RAM IS
-- Only default values
GENERIC (RAM_SIZE          : positive := 60000;
         RAM_ADDRESS_SIZE   : positive := 16);
PORT(Clk, we_n       : IN STD_LOGIC;
     din        : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
     addr              : IN STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE -1) DOWNTO 0);
     dout        : OUT STD_LOGIC_VECTOR(2  DOWNTO 0));
END PIX_RAM;

ARCHITECTURE rtl OF PIX_RAM IS
TYPE ram_type IS ARRAY((RAM_SIZE -1) DOWNTO 0) OF STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL  RAM          : ram_type :=(others =>(others =>'0'));
BEGIN
PROCESS(Clk)
BEGIN
IF Clk'event and Clk = '1' THEN
IF we_n = '0' THEN
   RAM(CONV_INTEGER(addr)) <= din;
END IF;
   dout <= RAM(CONV_INTEGER(addr));
END IF;
END PROCESS;
END rtl;
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
USE ieee.STD_LOGIC_unsigned.all;
USE ieee.STD_LOGIC_arith.all;

ENTITY VGA_core IS
  -- Screen_SIZE_horizontal => Sets the horizontal size of the editable screen area
  -- Screen_SIZE_vertical   => Sets the vertical size of the editable screen area
  -- RAM_SIZE               => Sets the memory size that will be used,
  -- RAM_ADDRESS_SIZE      => Memory address bus width
  -- Resolution320x240     => Selects screen resolution, 0 for 640x480 pixels and 1 for 320x240 pixels
  GENERIC (      -- Only default values --
    Screen_SIZE_horizontal : POSITIVE := 300;
    Screen_SIZE_vertical : POSITIVE := 200;
    RAM_SIZE   : POSITIVE := 60000;
    RAM_ADDRESS_SIZE : POSITIVE := 16;
    Resolution320x240 : INTEGER := 0);
  PORT(Clk, Reset_n, RAM_we_n : IN STD_LOGIC;
    Char_en     : IN STD_LOGIC;
    Char_nr     : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    ROM_Data    : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    BG_RGB_Data_in, RGB_Data_from_RAM : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    RGB_Data_in     : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    X_in, Y_in    : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    RAM_we_n_out    : OUT STD_LOGIC;
    H_sync, V_sync    : OUT STD_LOGIC;
    Memory_Busy, Char_busy   : OUT STD_LOGIC;
    RGB_Data_to_RAM    : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    RGB      : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    ROM_Address    : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
    RAM_Address    : OUT STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE-1) DOWNTO 0));
END VGA_core;

ARCHITECTURE rtl of VGA_core IS
  ------------ Signal declarations for the VGA Sync Generator block ------------
  SIGNAL Clk24       : STD_LOGIC;
  SIGNAL H_sync_int, V_sync_int     : STD_LOGIC;
  SIGNAL Pre_RGB_Data_en_h, RGB_Data_en_h, RGB_Data_en_v : STD_LOGIC;
  SIGNAL RGB_Data_en     : STD_LOGIC;
  SIGNAL X_in_sig, Y_in_sig     : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL RGB_Data       : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL H_count, V_count     : STD_LOGIC_VECTOR(10 DOWNTO 0);
  SIGNAL Memory_Busy_sig, Mem_b, RAM_we_n_sig_out  : STD_LOGIC;
  ---------------- Signal declarations for the Control Unit block ----------------
  SIGNAL Repeat_on, Line_done, Line_done_shifted : STD_LOGIC;
  SIGNAL Two_Pix_Count : STD_LOGIC;
  SIGNAL Memory_Busy_sig, Mem_b, RAM_we_n_sig_out  : STD_LOGIC;
  SIGNAL ROM_Address_sig       : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL Addr_Counter       : STD_LOGIC_VECTOR((RAM_ADDRESS_SIZE-1) DOWNTO 0);
  SIGNAL Soft_RAM_Addr      : STD_LOGIC_VECTOR(19 DOWNTO 0);
  SIGNAL Screen_SIZE_h_bin, Screen_SIZE_h_bin_shifted  : STD_LOGIC_VECTOR(9 DOWNTO 0);
  TYPE STATE_TYPE IS (Idle, Wait_st, Char_line_nr0, Char_line_nr1, Char_line_nr2,
                        Char_line_nr3, Char_line_nr4, Char_line_nr5, Char_line_nr6);
  SIGNAL STATE        : STATE_TYPE;
  SIGNAL Char_count, Char_count_delay1, Char_count_delay2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL Char_count_ok, Char_count_up, Char_count_rst  : STD_LOGIC;
BEGIN
  --------------------------------------------- Frequence divider Process, creates the 24MHz clock ---------------------------------------------
END VGA_core;
Freq_div : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
    Clk24 <= '1';
ELSIF Clk'event AND Clk = '1' THEN
    IF Clk24 = '1' THEN
        Clk24 <= '0';
    ELSE
        Clk24 <= '1';
    END IF;
END IF;
END PROCESS;

--*******************************************************************************--
------------------------- VGA Sync Generator block -----------------------------
--*******************************************************************************--

-- This Process generates the VGA sync signals Horizontal Sync and Vertical Sync --

VGA_SYNC: Process(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
    H_count <= (Others => '0');
    V_count <= (Others => '0');
    H_sync_int <= '0';
    V_sync_int <= '0';
    Line_done <= '0';
    RGB_Data_en_H <= '0';
    RGB_Data_en_V <= '0';
    Pre_RGB_Data_en_h <= '0';
ELSIF Clk'event AND Clk = '1' THEN
    IF Clk24 = '1' THEN
        -- <-Clock out RGB Pixel Row Data ->   <-H Sync->
        -- ------------------------------------__________--------
        -- 0                           640   659       755    799
        IF H_count >= 799 THEN
            H_count <= (Others => '0');
        ELSE
            H_count <= H_count + 1;
        END IF;
        -------- Horizontal Sync Generation --------
        IF (H_count <= 755) AND (H_count >= 659) THEN
            H_sync_int <= '0';
        ELSE
            H_sync_int <= '1';
        END IF;
        -- <-480 Horizontal Sync (pixel rows)-> ->V Sync<-
        -- ---------------------------------------_______----------
        -- 0                                480   493-494       524
        IF (V_count >= 524) AND (H_count >= 699) THEN
            V_count <= (Others => '0');
        ELSIF H_count = 699 THEN
            Line_done <= '1';
            V_count <= V_count + 1;
        ELSE
            Line_done <= '0';
        END IF;
        -------- Vertical Sync Generation --------
        IF (V_count <= 494) AND (V_count >= 493) THEN
            V_sync_int <= '0';
        ELSE
            V_sync_int <= '1';
        END IF;
    END IF;
END PROCESS;

-- Generate Video on Screen Signals for Pixel Data --
-- If the editable screen size is smaller then 640x480 it is centered --
IF (H_count <= ((320 + (Screen_SIZE_horizontal/2)) -1)) AND (H_count >= (320 - (Screen_SIZE_horizontal/2))) THEN
  Pre_RGB_Data_en_H <= '1';
ELSE
  Pre_RGB_Data_en_H <= '0';
END IF;

THEN
  RGB_Data_en_V <= '1';
ELSE
  RGB_Data_en_V <= '0';
END IF;
END IF;

END PROCESS VGA_SYNC;

-- RGB_Data_enable disables pixel color data when not in the editable screen area --

RGB_Data_en <= RGB_Data_en_H AND RGB_Data_en_V;

Output_Sync : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
  RGB <= (Others => '0');
  H_sync <= '0';
  V_sync <= '0';
ELSIF Clk'event AND Clk = '1' THEN
  RGB(0) <= RGB_Data(0) AND RGB_Data_en;
  RGB(1) <= RGB_Data(1) AND RGB_Data_en;
  RGB(2) <= RGB_Data(2) AND RGB_Data_en;
  H_sync <= H_sync_int;
  V_sync <= V_sync_int;
END IF;
END PROCESS;

RGB_Data <= RGB_Data_from_RAM;

--********************************************************************************--
--- Control Unit block -------********--
---******************************************************************************
--------------- 640x480 pixels or 320x240 pixel resolution option ---------------
--------------- 640x480 pixels or 320x240 pixel resolution option ---------------

Pixel_1x1_or_2x2 : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
  Addr_Counter <= (Others => '0');
  Repeat_on <= '0';
  Line_done_shifted <= '0';
  Two_Pix_Count <= '0';
ELSIF Clk'event AND Clk = '1' THEN
  IF Clk24 = '1' THEN
    CASE Resolution320x240 IS
      WHEN 0 => IF RGB_Data_en = '1' THEN
        IF Addr_Counter < (RAM_size-1) THEN
          Addr_Counter <= Addr_Counter +1;
        ELSE
          Addr_Counter <= (Others => '0');
        END IF;
      END IF;
      WHEN OTHERS => Line_done_shifted <= Line_done;
    END CASE;
  END IF;
END IF;
END PROCESS;

-- Case 640x480 pixel resolution
WHEN 0 => IF RGB_Data_en = '1' THEN
  IF Addr_Counter < (RAM_size-1) THEN
    Addr_Counter <= Addr_Counter +1;
  ELSE
    Addr_Counter <= (Others => '0');
  END IF;
END IF;

-- Case 320x240 pixel resolution
-- In this case two samples are taken from each memory address,
-- and every pixel row is read twice from the memory
-- This creates double sized pixels
WHEN OTHERS => Line_done_shifted <= Line_done;
IF RGB_Data_en = '1' THEN
  IF Two_Pix_Count = '1' THEN
    Two_Pix_Count <= '0';
    IF Addr_Counter < ((RAM_size)) THEN
      Addr_Counter <= Addr_Counter +1;
    ELSE
      Addr_Counter <= (Others => '0');
    END IF;
  ELSE
    Two_Pix_Count <= '1';
  END IF;
END IF;
IF RGB_Data_en_V = '1' THEN
  IF Line_done = '1' THEN
    Repeat_on <= NOT Repeat_on;
  END IF;
  IF (Line_done_shifted = '1') AND (Repeat_on = '1') THEN
    Addr_Counter <= Addr_Counter - (((Screen_SIZE_horizontal)/2));
  ELSIF (Line_done_shifted = '1') AND (Addr_Counter = (RAM_size)) THEN
    Addr_Counter <= (Others => '0');
  END IF;
END CASE;
END IF;
END PROCESS;
-------------------- Conversion to STD_LOGIC_VECTOR ------------------------------
Screen_SIZE_h_bin <= CONV_STD_LOGIC_VECTOR (Screen_SIZE_horizontal, 10);
--------------------------- Division by two --------------------------------------
Screen_SIZE_h_bin_shifted(9) <= '0';
Screen_SIZE_h_bin_shifted(8 DOWNTO 0) <= Screen_SIZE_h_bin(9 DOWNTO 1);
------------- Converts X,Y coordinates to RAM address in both -------------------
--------------- 640x480 pixel and 320x240 pixel resolution---------------------
Soft_RAM_Addr <= (( Screen_SIZE_h_bin * Y_in_sig) + X_in_sig) WHEN Resolution320x240 = 0 ELSE
  ((Screen_SIZE_h_bin_shifted) * Y_in_sig) + X_in_sig);
------------------------- RAM address multiplexer -------------------------------
RAM_Address <= Addr_Counter WHEN (RAM_we_n_sig_out = '1') AND (Addr_Counter < (RAM_size)) ELSE
  Soft_RAM_Addr((RAM_ADDRESS_SIZE -1) DOWNTO 0);
RAM_we_n_out <= RAM_we_n_sig_out;
----------------------------------------------------------------------------------
Mem_b<='1' WHEN (H_count<=(321+(Screen_SIZE_horizontal/2)))AND(H_count>=(320-(Screen_SIZE_horizontal/2))) ELSE
  '0';
Memory_Busy_sig <= (Memory_bus) AND (RGB_Data_en_V);
Memory_Busy <= Memory_Busy_sig;
----------------------------------------------------------------------------------
Char_FSM_beh : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
  STATE <= Idle;
  Char_count_rst <= '0';
  Char_busy <= '0';
ELSIF Clk'event AND Clk = '1' THEN
  CASE STATE is
    WHEN Idle => IF Char_en = '0' AND Memory_Busy_sig = '0' THEN
      Char_count_rst <= '0';
      STATE <= Wait_st;
      Char_busy <= '1';
    ELSE
      Char_count_rst <= '1';
      Char_busy <= '0';
    END IF;
    WHEN Wait_st => IF Memory_Busy_sig = '0' THEN
      Char_count_rst <= '0';
      STATE <= Char_line_nr0;
    ELSE
      Char_count_rst <= '1';
    END IF;
  END CASE;
END CASE;
END IF;
END PROCESS;
Char_busy <= '1';

WHEN Char_line_nr0 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr1;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Char_line_nr1 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr2;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Char_line_nr2 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr3;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Char_line_nr3 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr4;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Char_line_nr4 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr5;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Char_line_nr5 -> IF Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr6;
  Char_count_rst <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
END IF;
Char_busy <= '1';

WHEN Others -> IF Char_count_ok = '0' AND Memory_Busy_sig = '0' THEN
  STATE <= Char_line_nr0;
  Char_count_rst <= '0';
  Char_busy <= '1';
ELSIF Memory_Busy_sig = '0' THEN
  STATE <= Idle;
  Char_count_rst <= '1';
  Char_busy <= '0';
ELSE
  STATE <= Wait_st;
  Char_count_rst <= '1';
  Char_busy <= '1';
END IF;
END CASE;
END IF;
END PROCESS;

------------------------------- FSM outputs ---------------------------------
Char_FSM_Otputs : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
  X_in_sig <= (Others => '0');
  Y_in_sig <= (Others => '0');
  RGB_Data_to_RAM <= (Others => '0');
  RAM_we_n_sig_out <= '0';
  Char_count_up <= '0';
  ROM_Address_sig <= (Others => '0');
ELSIF Clk'event AND Clk = '1' THEN
  CASE STATE is
    WHEN Idle => X_in_sig <= X_in;
    WHEN Wait_st => Char_count_up <= '0';
    WHEN Char_line_nr0 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr1 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr2 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr3 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr4 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr5 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr6 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr7 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr8 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr9 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr10 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr11 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr12 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr13 => RAM_we_n_sig_out <= '0';
    WHEN Char_line_nr14 => RAM_we_n_sig_out <= '0';
    WHEN Others => RAM_we_n_sig_out <= '0';
  END CASE;
END IF;

END PROCESS;
WHEN Char_line_nr2 => RAM_we_n_sig_out <= '0';
  Char_count_up <= '1';
  ROM_Address_sig <= Char_nr + Char_count;
  IF Memory_Busy_sig = '0' THEN
    RAM_we_n_sig_out <= '0';
    X_in_sig <= X_in + 3;
    Y_in_sig <= Y_in + Char_count_delay2;
  ELSE
    RAM_we_n_sig_out <= '1';
    X_in_sig <= X_in + 3;
    Y_in_sig <= Y_in;
  END IF;
  IF ROM_Data(3) = '0' THEN
    RGB_Data_to_RAM <= BG_RGB_Data_in;
  ELSE
    RGB_Data_to_RAM <= RGB_Data_in;
  END IF;
WHEN Char_line_nr3 => RAM_we_n_sig_out <= '0';
  Char_count_up <= '0';
  ROM_Address_sig <= Char_nr + Char_count;
  IF Memory_Busy_sig = '0' THEN
    RAM_we_n_sig_out <= '0';
    X_in_sig <= X_in + 4;
    Y_in_sig <= Y_in + Char_count_delay2;
  ELSE
    RAM_we_n_sig_out <= '1';
    X_in_sig <= X_in;
    Y_in_sig <= Y_in;
  END IF;
  IF ROM_Data(2) = '0' THEN
    RGB_Data_to_RAM <= BG_RGB_Data_in;
  ELSE
    RGB_Data_to_RAM <= RGB_Data_in;
  END IF;
WHEN Char_line_nr4 => RAM_we_n_sig_out <= '0';
  Char_count_up <= '0';
  ROM_Address_sig <= Char_nr + Char_count;
  IF Memory_Busy_sig = '0' THEN
    RAM_we_n_sig_out <= '0';
    X_in_sig <= X_in + 5;
    Y_in_sig <= Y_in + Char_count_delay2;
  ELSE
    RAM_we_n_sig_out <= '1';
    X_in_sig <= X_in;
    Y_inSig <= Y_in;
  END IF;
  IF ROM_Data(1) = '0' THEN
    RGB_Data_to_RAM <= BG_RGB_Data_in;
  ELSE
    RGB_Data_to_RAM <= RGB_Data_in;
  END IF;
WHEN Char_line_nr5 => Char_count_up <= '0';
  ROM_Address_sig <= Char_nr + Char_count;
  IF Memory_Busy_sig = '0' THEN
    RAM_we_n_sig_out <= '0';
    X_in_sig <= X_in + 6;
    Y_in_sig <= Y_in + Char_count_delay2;
  ELSE
    RAM_we_n_sig_out <= '1';
    X_inSig <= X_in;
    Y_inSig <= Y_in;
  END IF;
  IF ROM_Data(0) = '0' THEN
    RGB_Data_to_RAM <= BG_RGB_Data_in;
  ELSE
    RGB_Data_to_RAM <= RGB_Data_in;
  END IF;
WHEN Others => Char_count_up <= '0';
  ROM_Address_sig <= Char_nr + Char_count;
IF Memory_Busy_sig = '0' THEN
  RAM_we_n_sig_out <= Char_count_ok;
  X_in_sig <= X_in;
  Y_in_sig <= Y_in + Char_count_delay2;
ELSE
  RAM_we_n_sig_out <= '1';
  X_in_sig <= X_in;
  Y_in_sig <= Y_in;
END IF;
IF ROM_Data(6) = '0' THEN
  RGB_Data_to_RAM <= BG_RGB_Data_in;
ELSE
  RGB_Data_to_RAM <= RGB_Data_in;
END IF;
END CASE;
END IF;
END PROCESS;

CHAR_Y_COUNTER : PROCESS(Clk, Reset_n)
BEGIN
IF Reset_n = '0' THEN
  Char_count <= (Others => '0');
  Char_count_ok <= '0';
  Char_count_delay1 <= "0000";
  Char_count_delay2 <= "0000";
ELSIF Clk'event AND Clk = '1' THEN
  IF Char_count_rst = '1' THEN
    Char_count <= (Others => '0');
  ELSIF Char_count_up = '1' THEN
    IF Char_count < 8 THEN
      Char_count <= Char_count + 1;
    ELSE
      Char_count <= (Others => '0');
    END IF;
  END IF;
  IF Char_count < 8 THEN
    Char_count_ok <= '0';
  ELSE
    Char_count_ok <= '1';
  END IF;
  Char_count_delay1 <= Char_count;
  Char_count_delay2 <= Char_count_delay1;
END IF;
END PROCESS;
end rtl;
VHDL source code for the two test benches discussed in chapter 4

```vhdl
-- File Name : VGA_Test_bench1x1.vhd
-----------------------------------------------------------------------------
-- Synthesis Level : Rtl VHDL
-----------------------------------------------------------------------------
-- Revisions List :
--                      Author Date
--            Zoltan Nagy  2005-11-24
-----------------------------------------------------------------------------
-- For this test 320x240 pixels resolution mode is disabled
-- and the screen size is set to 300x200.
-- The testbench creates test inputs according to the figure below
-- it controls that right outputs are received,
-- it also controls the the timing for the V_sync and H_sync signal.
-- The testbench reports where the failure has occurred, if a failure occurs.
-----------------------------------------------------------------------------
-- Monitor Screen
-- ---------------------------------------------------------------
--          Row1        |.....| <= Five red pixels at x=0,1,2,3,4 and y=0
--                  |   |
--                  |   |
--                  |   |
--          Row100     |.....| <= Five blue pixels at x=90,91,92,93,94 and y=99
--                  |   |
--                  |   |
--                  |   |
--          Row200     |.....| <= Five white pixels on x=195,196,197,198,199 and y=199
-- ---------------------------------------------------------------
-----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY VGA_Test_bench1x1 IS
GENERIC (Screen_SIZE_horizontal   : POSITIVE := 300;
          Screen_SIZE_vertical   : POSITIVE := 200;
          RAM_SIZE               : POSITIVE := 60000;
          RAM_ADDRESS_SIZE      : POSITIVE := 16;
          Resolution320x240      : INTEGER := 0);
PORT(Test_passed                : OUT STD_LOGIC := 'H'); -- Should stay 'H'
END VGA_Test_bench1x1;

ARCHITECTURE rtl OF VGA_Test_bench1x1 IS
-----------------------------------------------------------------------------
SIGNAL Clk, Reset_n    : STD_LOGIC := '0';
SIGNAL Pix_RAM_we_n    : STD_LOGIC := '1';
SIGNAL RGB_in       : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
SIGNAL X_in, Y_in    : STD_LOGIC_VECTOR(9 downto 0) := "0000000000";
SIGNAL H_sync, V_sync    : STD_LOGIC;
SIGNAL Memory_Busy     : STD_LOGIC;
SIGNAL RGB      : std_logic_vector(2 DOWNTO 0) := "000";
-----------------------------------------------------------------------------
COMPONENT VGA_Top
GENERIC (Screen_SIZE_horizontal   : POSITIVE;
          Screen_SIZE_vertical   : POSITIVE;
          RAM_SIZE               : POSITIVE;
          RAM_ADDRESS_SIZE      : POSITIVE;
          Resolution320x240      : INTEGER);
PORT(Clk, Reset_n, Pix_RAM_we_n  : IN STD_LOGIC;
     RGB_in                 : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
     X_in, Y_in            : IN STD_LOGIC_VECTOR(9 downto 0);
     H_sync, V_sync        : OUT STD_LOGIC;
     Memory_Busy           : OUT STD_LOGIC;
     RGB                   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END COMPONENT;
```
BEGIN
VGA_Top_L : VGA_Top GENERIC MAP(Screen_SIZE_horizontal, Screen_SIZE_vertical,
RAM_SIZE, RAM_ADDRESS_SIZE, Resolution320x240)
PORT MAP(Clk, Reset_n, Pix_RAM_we_n, RGB_in, X_in, Y_in, H_sync, V_sync,
Memory_Busy, RGB);
--------------------------------------------------------------------------------
---------------------------- Setting input signals ---------------------------
CLk <= NOT Clk AFTER 10ns; -- Sets the Clock frequency to 50MHz
Reset_n <= '1' AFTER 20ns; -- Disables Reset after one clock cycle;
--------------------------------------------------------------------------------

Input: Process
BEGIN
WAIT FOR 100ns;
WAIT UNTIL Clk = '1';
-- First five pixels on row nr0 (first row) with red color
-- Pixel nr1
Pix_RAM_we_n <= '0';
Y_in <= "0000000000"; -- First row
X_in <= "0000000000"; -- First pixel
RGB_in <= "100"; -- Red
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0000000001"; -- Next pixel, same row and same color
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0000000010";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0000000011";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0000000100";
WAIT FOR 20ns;

-- Next five pixels on row nr100 with blue color
Y_in <= "0001100011";
X_in <= "0001011010"; -- First pixel
RGB_in <= "001"; -- Blue
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0001011011";
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0001011100";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0001011101";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0001011110";
WAIT FOR 20ns;

-- Next five pixels on row nr200 with blue color
Y_in <= "0011000111";
X_in <= "0011000101"; -- First pixel at x=195
RGB_in <= "111"; -- White
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0011000100";
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0011000101";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0011000110";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0011000111";
WAIT FOR 20ns;

Pix_RAM_we_n <= '1';
WAIT;
END PROCESS;

------------------------------------------ Tests the outputs ------------------------------------------

H_sync1 : PROCESS
BEGIN
  WAIT FOR 50ns; -- 20 + 30 After Reset is '1' wait two positive clock edges, v_sync and h_sync should then be zero
  WAIT UNTIL Clk = '1';
  IF (H_sync = '0') OR (V_sync = '0') THEN
    Test_passed <= '0';
    assert false
    report "******* ERROR AFTER RESET !!! ********" severity warning;
  END IF;
  WAIT;
END PROCESS;

H_sync2 : PROCESS
BEGIN
  -- <-Clock out RGB Pixel Row Data -> <-H Sync->
  -- ------------------------------------__________--------
  -- 0                           640   659       755    799
  -- |------------- 26360ns -------------|
  -- This process tests the timing for the first H_sync pulse
  WAIT FOR 26410ns; -- 50 + 659*2
  WAIT UNTIL Clk = '1';
  IF H_sync = '1' THEN
    Test_passed <= '0';
    assert false
    report "******* ERROR FOUND IN THE FIRST HORIZONTAL SYNC SIGNAL!!! ********" severity warning;
  END IF;
  WAIT;
END PROCESS;

H_sync4 : PROCESS
BEGIN
  --                <-H Sync-><-     Clock out RGB Pixel Row Data       -><-H Sync->
  --             ---__________--------------------------------------------__________---------
  --         640   659       755    799 0                         640   659       755    799
  -- Time interval |-------------------- 32000ns -------------------------|
  -- This process measures the correct timing between the H_sync pulses
  WAIT UNTIL H_sync = '1';
  WAIT UNTIL H_sync = '0';
  WAIT FOR 32000ns;
  WAIT UNTIL Clk = '1';
  IF H_sync = '1' THEN
    Test_passed <= '0';
    assert false
    report "******* TIMING ERROR FOUND IN THE HORIZONTAL SYNC SIGNAL!!! ********" severity warning;
  END IF;
END PROCESS;

V_sync4 : PROCESS
BEGIN
  -- -> V Sync<-   <-480 Horizontal Sync (pixel rows)-> -> V Sync<-  
  -- ------_______-------------------------------------------------_______---
  --       493-494    524 0                                   480
  --       |------------------------------------------------------|
  -- This process measures the correct timing between the V_sync pulses
  WAIT UNTIL V_sync = '1';
  WAIT UNTIL V_sync = '0';
  WAIT FOR 16768000ns; -- 800*524*two_clock_cycles
  WAIT UNTIL Clk = '1';
  IF V_sync = '1' THEN
Test_passed <= '0';
assert false
report "******** TIMING ERROR FOUND IN THE VERTICAL SYNC SIGNAL!!! ********"
severity warning;
END IF;
END PROCESS;

----------------------------------------------------------------------------------------------------------------------
--------------------------------- Werifies if the RGB output is correct -----------------------------------
----------------------------- Werifies the six first pixels in the first row -----------------------
-- The RGB value of the first five should be 100 (red) and for the sixth it should be 000 (black) --
RGB_Red : PROCESS
BEGIN
-- One clock cycles after the Memory_Busy signal gets high the pixel data output should start
WAIT UNTIL Memory_Busy = '1'; -- First row(Y=0)
WAIT FOR 40ns;
WAIT UNTIL Clk = '1';
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN THE FIRST ROW !!! ********"
  severity warning;
END IF;
-- The RGB value should be 2 clock cycles stable in "1x1 pixel" mode
WAIT FOR 40ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN THE FIRST ROW !!! ********"
  severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN THE FIRST ROW !!! ********"
  severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN THE FIRST ROW !!! ********"
  severity warning;
END IF;
WAIT FOR 40ns;
-- The RGB value should be black here
IF RGB /= "000" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN THE FIRST ROW !!! ********"
  severity warning;
END IF;
WAIT FOR 40ns;
-- Now verifies the five pixels at the following coordinates ---------------------------------------
-- 99 rows takes ((800*99)+90)*(two_clock_cycles), 90 for x=90 on the 100th row equals 3171600ns
-- and 3171600-6*40 (the six previous pixels) equals 317360ns
WAIT FOR 3171600ns;
IF RGB /= "001" THEN
  Test_passed <= '0';
  assert false
  report "******** PIXEL ERROR FOUND IN ROW Nr100 !!! ********"
  severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 40ns;
IF RGB /= "001" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr100 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "001" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr100 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "001" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr100 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "001" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr100 !!! *******" severity warning;
END IF;
-- The RGB value should be black here
IF RGB /= "000" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr100 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
-- The RGB value should be white here
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
-- From row100(Y=99) to row200(Y=199) it is 100 rows
-- The 100 rows takes ((800*100)+99)*(two_clock_cycles),
-- 99 is the difference between the x coordinates in row100 and row 200
-- so it equals 3203960ns
WAIT FOR 3203960ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
WAIT FOR 40ns;
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;
-- The RGB value should be white here
IF RGB /= "111" THEN
  Test_passed <= '0';
  assert false
  report "******* PIXEL ERROR FOUND IN ROW Nr200 !!! *******" severity warning;
END IF;

WAIT FOR 40ns;
IF RGB /= "111" THEN
    Test_passed <= '0';
    assert false
    report "******** PIXEL ERROR FOUND IN ROW Nr200 !!! ********" severity warning;
END IF;
WAIT;
END PROCESS;
END rtl;
For this test 320x240 pixels resolution mode is enabled
-- and the screen size is set to 640x480.
-- The testbench creates test inputs according to the figure bellow
-- it controls that the right outputs are received,
-- it also controls the the timing for the V_sync and H_sync signal.
-- The testbench reports where the failure has occured, if a failure occurs.

Monitor Screen

Row2 |..... |<= Five red pixels at x=0,1,2,3,4 and y=1

Row121 |..... |<= Five blue pixels at x=118,119,120,121,122 and y=120

Row240 |..... |<= Five white pixels on x= 315,316,317,318,319 and y=239

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY VGA_Test_bench IS
GENERIC (Screen_SIZE_horizontal : POSITIVE := 640;
Screen_SIZE_vertical     : POSITIVE := 480;
RAM_SIZE    : POSITIVE := 76800;
RAM_ADDRESS_SIZE  : POSITIVE := 17;
Resolution320x240         : INTEGER := 1);
PORT(Test_passed                : OUT STD_LOGIC := 'H'); -- Should stay 'H'
END VGA_Test_bench;

ARCHITECTURE rtl OF VGA_Test_bench IS

BEGIN
VGA_Top_L : VGA_Top GENERIC MAP(Screen_SIZE_horizontal, Screen_SIZE_vertical,
RAM_SIZE, RAM_ADDRESS_SIZE, Resolution320x240)
PORT MAP(Clk, Reset_n, Pix_RAM_we_n, RGB_in, X_in, Y_in, H_sync, V_sync,
Memory_Busy, RGB);
--- Setting input signals --------------------------
CLk <= NOT Clk AFTER 10ns; -- Sets the Clock frequency to 50MHz
Reset_n <= '1' AFTER 20ns; -- Disables Reset after one clock cycle;
--- Setting input signals --------------------------

Input: Process
BEGIN
WAIT UNTIL Clk = '1';
WAIT UNTIL Memory_Busy = '1';
WAIT UNTIL Memory_Busy = '0';
-- First five pixels on row nr1 (second row) with red color
-- Pixel nr1
Pix_RAM_we_n <= '0';
Y_in <= "0000000001"; -- Second row
X_in <= "0000000000"; -- First pixel
RGB_in <= "100"; -- Red
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0000000001"; -- Next pixel, same row and same color
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0000000010";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0000000011";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0000000100";
WAIT FOR 20ns;
-- Next five pixels on row nr121 with blue color
Y_in <= "0001111000";
X_in <= "0001111010"; -- First pixel
RGB_in <= "001"; -- Blue
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0001111011";
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0001111000";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0001111001";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0001111010";
WAIT FOR 20ns;
-- Next five pixels on row nr240 with blue color
Y_in <= "0011101111";
X_in <= "01001111011"; -- First pixel at x=315
RGB_in <= "111"; -- White
WAIT FOR 20ns; -- 20ns one clock period to save data to memory
-- Pixel nr2
X_in <= "0100111100";
WAIT FOR 20ns;
-- Pixel nr3
X_in <= "0100111110";
WAIT FOR 20ns;
-- Pixel nr4
X_in <= "0100111110";
WAIT FOR 20ns;
-- Pixel nr5
X_in <= "0100111110";
WAIT FOR 20ns;
Pix_RAM_we_n <= '1';
WAIT;
END PROCESS;
--- Setting input signals --------------------------
H_sync1 : PROCESS
BEGIN
  WAIT FOR 50ns; -- 20 + 30 After Reset is '1' wait two positive clock edges, v_sync and h_sync
  should then be zero
  WAIT UNTIL Clk = '1';
  IF (H_sync = '0') OR (V_sync = '0') THEN
    Test_passed <= '0';
    assert False
    report "******** ERROR AFTER RESET !!! ********"
    severity warning;
  END IF;
  WAIT;
END PROCESS;

H_sync2 : PROCESS
BEGIN
  -- Clock out RGB Pixel Row Data -> <-H Sync>
  -- ------------------------------------__________--------
  -- 0                           640 659 755 799
  -- |------------- 26360ns -------------|
  -- This process tests the timing for the first H_sync pulse
  WAIT FOR 26410ns; -- 50 + 659*2
  WAIT UNTIL Clk = '1';
  IF H_sync = '1' THEN
    Test_passed <= '0';
    assert False
    report "******** ERROR FOUND IN THE FIRST HORIZONTAL SYNC SIGNAL!!! ********"
    severity warning;
  END IF;
  WAIT;
END PROCESS;

H_sync4 : PROCESS
BEGIN
  -- <-H Sync-> Clock out RGB Pixel Row Data -><->H Sync>
  -- 640 659 755 799
  -- Time interval |-------------------------- 32000ns --------------------------|
  -- This process measures the correct timing between the H_sync pulses
  WAIT UNTIL H_sync = '1';
  WAIT UNTIL H_sync = '0';
  WAIT FOR 32000ns;
  WAIT UNTIL Clk = '1';
  IF H_sync = '1' THEN
    Test_passed <= '0';
    assert False
    report "******** TIMING ERROR FOUND IN THE HORIZONTAL SYNC SIGNAL!!! ********"
    severity warning;
  END IF;
END PROCESS;

V_sync4 : PROCESS
BEGIN
  -- V Sync<-- <-480 Horizontal Sync (pixel rows)-> -- V Sync<-
  -- 493-494 524 0
  -- Time interval |---------------------------------------------|
  -- This process measures the correct timing between the V_sync pulses
  WAIT UNTIL V_sync = '1';
  WAIT UNTIL V_sync = '0';
  WAIT FOR 16768000ns; -- 800*524*two_clock_cycles
  WAIT UNTIL Clk = '1';
  IF V_sync = '1' THEN
    Test_passed <= '0';
    assert False
    report "******** TIMING ERROR FOUND IN THE VERTICAL SYNC SIGNAL!!! ********"
    severity warning;
  END IF;
END PROCESS;
------------------------------------------------------------------------------------------------------------------------
--------------------------------- Werifies if the RGB output is correct ----------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------- This process veriflies the six first pixels in the second row -------------------
-- The RGB value of the first five should be 100 (red) and for the sixth it should be 000 (black) --
RGB_Red : PROCESS
BEGIN
-- One row takes 800*(two_clock_cycles), in super pixel mode it is 800*2*(two_clock_cykles) which equals 64000ns
WAIT FOR 64000ns;
-- One clock cycles after the Memory_Busy signal gets high the pixel data output should start
WAIT UNTIL Memory_Busy = '1';
WAIT FOR 40ns;
WAIT UNTIL Clk = '1';
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 80ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
-- The RGB value should be black here
IF RGB /= "000" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
-- Tests the next row, it should be the same as the previous
WAIT FOR 31600ns; -- (800 * two_clock_cycles)-(5*80ns)
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 80ns;
IF RGB /= "100" THEN
  Test_passed <= '0';
  assert False
  report "******* PIXEL ERROR FOUND IN THE SECOND ROW !!! *******"
  severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "100" THEN
    Test_passed <= '0';
    assert False
    report "******** PIXEL ERROR FOUND IN THE SECOND ROW !!! ********"
    severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "100" THEN
    Test_passed <= '0';
    assert False
    report "******** PIXEL ERROR FOUND IN THE SECOND ROW !!! ********"
    severity warning;
END IF;
WAIT FOR 80ns;
-- The RGB value should be black here
IF RGB /= "000" THEN
    Test_passed <= '0';
    assert False
    report "******** PIXEL ERROR FOUND IN THE SECOND ROW !!! ********"
    severity warning;
END IF;
WAIT;
END PROCESS;

-- This process verifies the five pixels at the following coordinates --------------
-- Y=120 (row 120) and X=118,119,120,121 and 122 has the RGB value 001 (blue) --
RGB_Blue : PROCESS
BEGIN
    -- 120 rows takes (800*120)*(two_clock_cycles),
    -- in super pixel mode it takes (800*120)*2*(two_clock_cycles) which equals 7680000ns
    WAIT FOR 7680000ns;
    -- One clock cycles after the Memory_Busy signal gets high the pixel data output should start
    WAIT UNTIL Memory_Busy = '1';
    -- The 118-th pixel and the following five should have the rgb value 001 (blue)
    WAIT FOR 9520ns; -- 4*clock cycle *119
    WAIT UNTIL Clk = '1';
    IF RGB /= "001" THEN
        Test_passed <= '0';
        assert False
        report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********"
        severity warning;
    END IF;
    -- The RGB value should be 4 clock cycles stable in super pixel mode
    WAIT FOR 80ns;
    IF RGB /= "001" THEN
        Test_passed <= '0';
        assert False
        report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********"
        severity warning;
    END IF;
    WAIT FOR 80ns;
    IF RGB /= "001" THEN
        Test_passed <= '0';
        assert False
        report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********"
        severity warning;
    END IF;
    WAIT FOR 80ns;
    IF RGB /= "001" THEN
        Test_passed <= '0';
        assert False
        report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********"
        severity warning;
    END IF;
    WAIT FOR 80ns;
    IF RGB /= "001" THEN
        Test_passed <= '0';
        assert False
        report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********"
        severity warning;
    END IF;
    WAIT FOR 80ns;
END PROCESS;
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;
WAIT FOR 80ns;

-- The RGB value should be black here
IF RGB /= "000" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;

-- Tests the next row, it should be the same as the previous
WAIT FOR 31600ns; -- (800 * two_clock_cycles)-(5*80ns)
IF RGB /= "001" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;

-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 80ns;
IF RGB /= "001" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "001" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "001" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;
WAIT FOR 80ns;

-- The RGB value should be black here
IF RGB /= "000" THEN
Test_passed <= '0';
assert False
report "******** PIXEL ERROR FOUND IN ROW Nr121 !!! ********" severity warning;
END IF;
WAIT;
END PROCESS;

------------------- This process verifies the five pixels at the following coordinates --------------
----------- Y=239 (row 239)  and X=315,316,317,318 and 319 has the RGB value 111 (white) ------------
RGB_White : PROCESS
BEGIN
-- 239 rows takes (800*239)*(two_clock_cycles),
-- in super pixel mode it takes (800*239)*2*(two_clock_cycles) which equals 15296000ns
WAIT FOR 15296000ns;
-- One clock cycles after the Memory_Busy signal gets high the pixel data output should start
WAIT UNTIL Memory_Busy = '1';
-- The last five pixels should have the the RGB value 111 (white)
WAIT FOR 25240ns; -- 4*clock cycle *316
WAIT UNTIL Clk = '1';
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
-- Tests the next row, it should be the same as the previous row
WAIT FOR 31680ns; -- (800 * two_clock_cycles)-(4*80ns)
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
-- The RGB value should be 4 clock cycles stable in super pixel mode
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT FOR 80ns;
IF RGB /= "111" THEN
Test_passed <= '0';
assert False
report "****** PIXEL ERROR FOUND IN ROW Nr239 !!! *******"
severity warning;
END IF;
WAIT;
END PROCESS;
END rtl;
C source codes for the VGA Controller device drives

The register file altera_avalon_VGA_regs.h

```c
#ifndef __ALTERA_AVALON_VGA_REGS_H__
#define __ALTERA_AVALON_VGA_REGS_H__

#include <io.h>
#define IORD_ALTERA_AVALON_VGA_X_reg(base)         IORD(base, 0)
#define IOWR_ALTERA_AVALON_VGA_X_reg(base, data)   IOWR(base, 0, data)
#define IORD_ALTERA_AVALON_VGA_Y_reg(base)         IORD(base, 1)
#define IOWR_ALTERA_AVALON_VGA_Y_reg(base, data)   IOWR(base, 1, data)
#define IORD_ALTERA_AVALON_VGA_RGB_reg(base)       IORD(base, 2)
#define IOWR_ALTERA_AVALON_VGA_RGB_reg(base, data) IOWR(base, 2, data)
#define IORD_ALTERA_AVALON_VGA_BG_RGB_reg(base)       IORD(base, 3)
#define IOWR_ALTERA_AVALON_VGA_BG_RGB_reg(base, data) IOWR(base, 3, data)
#define IORD_ALTERA_AVALON_VGA_Char_nr_reg(base)       IORD(base, 4)
#define IOWR_ALTERA_AVALON_VGA_Char_nr_reg(base, data) IOWR(base, 4, data)
#define IORD_ALTERA_AVALON_VGA_Write_Done_reg(base)       IORD(base, 5)
#define IOWR_ALTERA_AVALON_VGA_Write_Done_reg(base, data) IOWR(base, 5, data)
#endif /* __ALTERA_AVALON_VGA_REGS_H__ */
```

altera_avalon_VGA_routines.c

```c
#include "altera_avalon_VGA_routines.h"
#include <stdio.h>

//****************************************************************************************************//
void print_pix(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb)
{
    unsigned int write_ok = 0;
    do{
        write_ok = IORD_ALTERA_AVALON_VGA_Write_Done_reg(address);
    }while(write_ok != 0x3);
    IOWR_ALTERA_AVALON_VGA_X_reg(address, x);
    IOWR_ALTERA_AVALON_VGA_Y_reg(address, y);
    IOWR_ALTERA_AVALON_VGA_RGB_reg(address, rgb);
    IOWR_ALTERA_AVALON_VGA_Write_Done_reg(address, 2);
}

//****************************************************************************************************//
void print_charsp(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG_RGB,unsigned int Charnr)
{
    unsigned int write_ok1 = 0;
    do{
        write_ok1 = IORD_ALTERA_AVALON_VGA_Write_Done_reg(address);
    }while(write_ok1 != 0x3);
    IOWR_ALTERA_AVALON_VGA_X_reg(address, x);
    IOWR_ALTERA_AVALON_VGA_Y_reg(address, y);
    IOWR_ALTERA_AVALON_VGA_RGB_reg(address, rgb);
    IOWR_ALTERA_AVALON_VGA_BG_RGB_reg(address, BG_RGB);
    IOWR_ALTERA_AVALON_VGA_Char_nr_reg(address, Charnr);
    IOWR_ALTERA_AVALON_VGA_Write_Done_reg(address, 1);
}

//****************************************************************************************************//
void print_char(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG_RGB,char Character)
```
unsigned int charnr = 4;
switch(Character)
{
    case 'A' : charnr=0; break;
    case 'B' : charnr=1; break;
    case 'C' : charnr=2; break;
    case 'D' : charnr=3; break;
    case 'E' : charnr=4; break;
    case 'F' : charnr=5; break;
    case 'G' : charnr=6; break;
    case 'H' : charnr=7; break;
    case 'I' : charnr=8; break;
    case 'J' : charnr=9; break;
    case 'K' : charnr=10; break;
    case 'L' : charnr=11; break;
    case 'M' : charnr=12; break;
    case 'N' : charnr=13; break;
    case 'O' : charnr=14; break;
    case 'P' : charnr=15; break;
    case 'Q' : charnr=16; break;
    case 'R' : charnr=17; break;
    case 'S' : charnr=18; break;
    case 'T' : charnr=19; break;
    case 'U' : charnr=20; break;
    case 'V' : charnr=21; break;
    case 'W' : charnr=22; break;
    case 'X' : charnr=23; break;
    case 'Y' : charnr=24; break;
    case 'Z' : charnr=25; break;
    case 'a' : charnr=26; break;
    case 'b' : charnr=27; break;
    case 'c' : charnr=28; break;
    case 'd' : charnr=29; break;
    case 'e' : charnr=30; break;
    case 'f' : charnr=31; break;
    case 'g' : charnr=32; break;
    case 'h' : charnr=33; break;
    case 'i' : charnr=34; break;
    case 'j' : charnr=35; break;
    case 'k' : charnr=36; break;
    case 'l' : charnr=37; break;
    case 'm' : charnr=38; break;
    case 'n' : charnr=39; break;
    case 'o' : charnr=40; break;
    case 'p' : charnr=41; break;
    case 'q' : charnr=42; break;
    case 'r' : charnr=43; break;
    case 's' : charnr=44; break;
    case 't' : charnr=45; break;
    case 'u' : charnr=46; break;
    case 'v' : charnr=47; break;
    case 'w' : charnr=48; break;
    case 'x' : charnr=49; break;
    case 'y' : charnr=50; break;
    case 'z' : charnr=51; break;
    case '1' : charnr=52; break;
    case '2' : charnr=53; break;
    case '3' : charnr=54; break;
    case '4' : charnr=55; break;
    case '5' : charnr=56; break;
    case '6' : charnr=57; break;
    case '7' : charnr=58; break;
    case '8' : charnr=59; break;
    case '9' : charnr=60; break;
    case '0' : charnr=61; break;
    case '.' : charnr=62; break;
    case ',' : charnr=63; break;
    case '_' : charnr=64; break;
    case '-' : charnr=65; break;
    case '+' : charnr=66; break;
    case '/' : charnr=67; break;
    case '(' : charnr=69; break;
    case ')' : charnr=70; break;
case '&' : charnr=73; break;
case '#' : charnr=74; break;
case '?' : charnr=75; break;
case '!': charnr=76; break;
case '[' : charnr=77; break;
case ']': charnr=78; break;
case ':': charnr=79; break;
case ';': charnr=80; break;
case '<': charnr=81; break;
case '>': charnr=82; break;
case '^': charnr=83; break;
case '@': charnr=87; break;
case '$': charnr=88; break;
case '£': charnr=89; break;
case '*': charnr=90; break;
default : charnr=4; break;
}
print_charsp(address, x,y,rgb,BG_RGB,charnr);
}

/********************************************
void delay(int t)
{
    int i;
    for(i=0; i<t; i++)
    {
    }
}

/********************************************
void print_hline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len,
unsigned int RGB)
{
    unsigned int x;
    for(x= x_start; x<len + x_start; x++)
    {
        print_pix(address, x, y_start, RGB);
    }
}

/********************************************
void print_vline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len,
unsigned int RGB)
{
    unsigned int y;
    for(y= y_start; y<len + y_start; y++)
    {
        print_pix(address, x_start, y, RGB);
    }
}

altera_avalon_VGA_routine.h
#include "altera_avalon_VGA_regs.h"
#define ALTERA_AVALON_VGA_TYPE (volatile unsigned int*)
void print_pix(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb);
void print_charsp(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG_RGB,unsigned int Charnr);
void print_char(unsigned int address, unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG_RGB,char Character);
void delay(int t);
void print_hline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);
void print_vline(unsigned int address, unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);
Appendix 4: Character table for the supported characters

Note that the yellow marked characters are not supported by the `print_char` function, only by the `print_charsp` function. The number at the upper left corner of each character defines its `Char_nr`.

<table>
<thead>
<tr>
<th>Character</th>
<th>Address</th>
<th>Character</th>
<th>Address</th>
<th>Character</th>
<th>Address</th>
<th>Character</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;</td>
<td>0</td>
<td>&quot;H&quot;</td>
<td>56</td>
<td>&quot;O&quot;</td>
<td>112</td>
<td>&quot;V&quot;</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>57</td>
<td></td>
<td>113</td>
<td></td>
<td>169</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>58</td>
<td></td>
<td>114</td>
<td></td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>59</td>
<td></td>
<td>115</td>
<td></td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>60</td>
<td></td>
<td>116</td>
<td></td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>61</td>
<td></td>
<td>117</td>
<td></td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>62</td>
<td></td>
<td>118</td>
<td></td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td>63</td>
<td></td>
<td>119</td>
<td></td>
<td>175</td>
</tr>
<tr>
<td>&quot;B&quot;</td>
<td>8</td>
<td>&quot;I&quot;</td>
<td>80</td>
<td>&quot;P&quot;</td>
<td>120</td>
<td>&quot;W&quot;</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td>81</td>
<td></td>
<td>121</td>
<td></td>
<td>177</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>82</td>
<td></td>
<td>122</td>
<td></td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>83</td>
<td></td>
<td>123</td>
<td></td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td>84</td>
<td></td>
<td>124</td>
<td></td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td>85</td>
<td></td>
<td>125</td>
<td></td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td>86</td>
<td></td>
<td>126</td>
<td></td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td>87</td>
<td></td>
<td>127</td>
<td></td>
<td>183</td>
</tr>
<tr>
<td>&quot;C&quot;</td>
<td>16</td>
<td>&quot;J&quot;</td>
<td>72</td>
<td>&quot;Q&quot;</td>
<td>128</td>
<td>&quot;X&quot;</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td></td>
<td>73</td>
<td></td>
<td>129</td>
<td></td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td></td>
<td>74</td>
<td></td>
<td>130</td>
<td></td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
<td>75</td>
<td></td>
<td>131</td>
<td></td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td>76</td>
<td></td>
<td>132</td>
<td></td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td>77</td>
<td></td>
<td>133</td>
<td></td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td></td>
<td>78</td>
<td></td>
<td>134</td>
<td></td>
<td>190</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td>79</td>
<td></td>
<td>135</td>
<td></td>
<td>191</td>
</tr>
<tr>
<td>&quot;D&quot;</td>
<td>24</td>
<td>&quot;K&quot;</td>
<td>80</td>
<td>&quot;R&quot;</td>
<td>136</td>
<td>&quot;Y&quot;</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td></td>
<td>81</td>
<td></td>
<td>137</td>
<td></td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td></td>
<td>82</td>
<td></td>
<td>138</td>
<td></td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td></td>
<td>83</td>
<td></td>
<td>139</td>
<td></td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td></td>
<td>84</td>
<td></td>
<td>140</td>
<td></td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td></td>
<td>85</td>
<td></td>
<td>141</td>
<td></td>
<td>197</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td></td>
<td>86</td>
<td></td>
<td>142</td>
<td></td>
<td>198</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td></td>
<td>87</td>
<td></td>
<td>143</td>
<td></td>
<td>199</td>
</tr>
<tr>
<td>&quot;E&quot;</td>
<td>32</td>
<td>&quot;L&quot;</td>
<td>88</td>
<td>&quot;S&quot;</td>
<td>144</td>
<td>&quot;Z&quot;</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td></td>
<td>89</td>
<td></td>
<td>145</td>
<td></td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td></td>
<td>90</td>
<td></td>
<td>146</td>
<td></td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td></td>
<td>91</td>
<td></td>
<td>147</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td></td>
<td>92</td>
<td></td>
<td>148</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td></td>
<td>93</td>
<td></td>
<td>149</td>
<td></td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td></td>
<td>94</td>
<td></td>
<td>150</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td></td>
<td>95</td>
<td></td>
<td>151</td>
<td></td>
<td>207</td>
</tr>
<tr>
<td>&quot;F&quot;</td>
<td>40</td>
<td>&quot;M&quot;</td>
<td>96</td>
<td>&quot;T&quot;</td>
<td>152</td>
<td>&quot;a&quot;</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td>41</td>
<td></td>
<td>97</td>
<td></td>
<td>153</td>
<td></td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td></td>
<td>98</td>
<td></td>
<td>154</td>
<td></td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td></td>
<td>99</td>
<td></td>
<td>155</td>
<td></td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td></td>
<td>100</td>
<td></td>
<td>156</td>
<td></td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td></td>
<td>101</td>
<td></td>
<td>157</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td></td>
<td>102</td>
<td></td>
<td>158</td>
<td></td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td></td>
<td>103</td>
<td></td>
<td>159</td>
<td></td>
<td>215</td>
</tr>
<tr>
<td>&quot;G&quot;</td>
<td>48</td>
<td>&quot;N&quot;</td>
<td>104</td>
<td>&quot;U&quot;</td>
<td>160</td>
<td>&quot;b&quot;</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>49</td>
<td></td>
<td>105</td>
<td></td>
<td>161</td>
<td></td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td></td>
<td>106</td>
<td></td>
<td>162</td>
<td></td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td></td>
<td>107</td>
<td></td>
<td>163</td>
<td></td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td></td>
<td>108</td>
<td></td>
<td>164</td>
<td></td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>53</td>
<td></td>
<td>109</td>
<td></td>
<td>165</td>
<td></td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td></td>
<td>110</td>
<td></td>
<td>166</td>
<td></td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td></td>
<td>111</td>
<td></td>
<td>167</td>
<td></td>
<td>223</td>
</tr>
<tr>
<td>Character Address</td>
<td>Character Address</td>
<td>Character Address</td>
<td>Character Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>224</td>
<td>35</td>
<td>280</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;c&quot;</td>
<td>225</td>
<td>&quot;j&quot;</td>
<td>281</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>226</td>
<td></td>
<td>282</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>227</td>
<td></td>
<td>283</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>228</td>
<td></td>
<td>284</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>229</td>
<td></td>
<td>285</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>230</td>
<td></td>
<td>286</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>231</td>
<td></td>
<td>287</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>232</td>
<td>36</td>
<td>288</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;d&quot;</td>
<td>233</td>
<td>&quot;k&quot;</td>
<td>289</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>234</td>
<td></td>
<td>290</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>235</td>
<td></td>
<td>291</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>236</td>
<td></td>
<td>292</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>237</td>
<td></td>
<td>293</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>238</td>
<td></td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>239</td>
<td></td>
<td>295</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>240</td>
<td>37</td>
<td>296</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;e&quot;</td>
<td>241</td>
<td>&quot;l&quot;</td>
<td>297</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>242</td>
<td></td>
<td>298</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>243</td>
<td></td>
<td>299</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>244</td>
<td></td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>245</td>
<td></td>
<td>301</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>246</td>
<td></td>
<td>302</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>247</td>
<td></td>
<td>303</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>248</td>
<td>38</td>
<td>304</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;f&quot;</td>
<td>249</td>
<td>&quot;m&quot;</td>
<td>305</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>250</td>
<td></td>
<td>306</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>251</td>
<td></td>
<td>307</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>252</td>
<td></td>
<td>308</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>253</td>
<td></td>
<td>309</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>254</td>
<td></td>
<td>310</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>255</td>
<td></td>
<td>311</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>256</td>
<td>39</td>
<td>312</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;g&quot;</td>
<td>257</td>
<td>&quot;n&quot;</td>
<td>313</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>258</td>
<td></td>
<td>314</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>259</td>
<td></td>
<td>315</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>260</td>
<td></td>
<td>316</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>261</td>
<td></td>
<td>317</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>262</td>
<td></td>
<td>318</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>263</td>
<td></td>
<td>319</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>264</td>
<td>40</td>
<td>320</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;h&quot;</td>
<td>265</td>
<td>&quot;o&quot;</td>
<td>321</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>266</td>
<td></td>
<td>322</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>267</td>
<td></td>
<td>323</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>268</td>
<td></td>
<td>324</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>269</td>
<td></td>
<td>325</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>270</td>
<td></td>
<td>326</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>271</td>
<td></td>
<td>327</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>272</td>
<td>41</td>
<td>328</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;i&quot;</td>
<td>273</td>
<td>&quot;p&quot;</td>
<td>329</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>274</td>
<td></td>
<td>330</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>275</td>
<td></td>
<td>331</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>276</td>
<td></td>
<td>332</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>277</td>
<td></td>
<td>333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>278</td>
<td></td>
<td>334</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>279</td>
<td></td>
<td>335</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>280</td>
<td>42</td>
<td>336</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>281</td>
<td>&quot;q&quot;</td>
<td>337</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>282</td>
<td></td>
<td>338</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>283</td>
<td></td>
<td>339</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>284</td>
<td></td>
<td>340</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>285</td>
<td></td>
<td>341</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>286</td>
<td></td>
<td>342</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>287</td>
<td></td>
<td>343</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>288</td>
<td>43</td>
<td>344</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>289</td>
<td>&quot;r&quot;</td>
<td>345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>290</td>
<td></td>
<td>346</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>291</td>
<td></td>
<td>347</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>292</td>
<td></td>
<td>348</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>293</td>
<td></td>
<td>349</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>294</td>
<td></td>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>295</td>
<td></td>
<td>351</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>296</td>
<td>44</td>
<td>352</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>297</td>
<td>&quot;s&quot;</td>
<td>353</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>298</td>
<td></td>
<td>354</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>299</td>
<td></td>
<td>355</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>300</td>
<td></td>
<td>356</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>301</td>
<td></td>
<td>357</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>302</td>
<td></td>
<td>358</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>303</td>
<td></td>
<td>359</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>304</td>
<td>45</td>
<td>360</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>305</td>
<td>&quot;t&quot;</td>
<td>361</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>306</td>
<td></td>
<td>362</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>307</td>
<td></td>
<td>363</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>308</td>
<td></td>
<td>364</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>309</td>
<td></td>
<td>365</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>310</td>
<td></td>
<td>366</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>311</td>
<td></td>
<td>367</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>312</td>
<td>46</td>
<td>368</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>313</td>
<td>&quot;u&quot;</td>
<td>369</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>314</td>
<td></td>
<td>370</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>315</td>
<td></td>
<td>371</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>316</td>
<td></td>
<td>372</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>317</td>
<td></td>
<td>373</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>318</td>
<td></td>
<td>374</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>319</td>
<td></td>
<td>375</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>320</td>
<td></td>
<td>376</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>321</td>
<td>47</td>
<td>377</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>322</td>
<td>&quot;v&quot;</td>
<td>378</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>323</td>
<td></td>
<td>379</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>324</td>
<td></td>
<td>380</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>325</td>
<td></td>
<td>381</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>326</td>
<td></td>
<td>382</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>327</td>
<td></td>
<td>383</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>328</td>
<td>48</td>
<td>384</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>329</td>
<td>&quot;w&quot;</td>
<td>385</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>330</td>
<td></td>
<td>386</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>331</td>
<td></td>
<td>387</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>332</td>
<td></td>
<td>388</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>333</td>
<td></td>
<td>389</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>334</td>
<td></td>
<td>390</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>335</td>
<td></td>
<td>391</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character Address</td>
<td>Character Address</td>
<td>Character Address</td>
<td>Character Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>448</td>
<td>63</td>
<td>504</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;5&quot;</td>
<td>&quot;,&quot;</td>
<td>&quot;)&quot;</td>
<td>[]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>456</td>
<td>64</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;6&quot;</td>
<td>&quot;-&quot;</td>
<td>&quot;%%&quot;</td>
<td>&quot;[&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>464</td>
<td>65</td>
<td>520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;7&quot;</td>
<td>&quot;+&quot;</td>
<td>&quot;,&amp;&quot;</td>
<td>&quot;.&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>472</td>
<td>66</td>
<td>528</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;8&quot;</td>
<td>&quot;/&quot;</td>
<td>&quot;&gt;#&quot;</td>
<td>&quot;&gt;&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>480</td>
<td>67</td>
<td>536</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;9&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;?&quot;</td>
<td>&quot;=&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>488</td>
<td>68</td>
<td>544</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>&quot;;&quot;</td>
<td>&quot;!&quot;</td>
<td>&quot;A&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>496</td>
<td>69</td>
<td>552</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;.&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;\n&quot;</td>
<td>672</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>673</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>674</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>675</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>676</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>677</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>678</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>679</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;}&quot;</td>
<td>680</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>681</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>682</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>684</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>685</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>686</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>687</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;{&quot;</td>
<td>688</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>689</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>690</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>691</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>692</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>693</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>694</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>695</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>696</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>697</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>698</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>699</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>700</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>701</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>702</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>703</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;@&quot;</td>
<td>704</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>705</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>706</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>707</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>708</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>709</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>711</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;$&quot;</td>
<td>712</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>713</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>714</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>715</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>716</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>717</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>718</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>719</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;£&quot;</td>
<td>720</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>721</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>722</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>723</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>724</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>725</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>726</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>727</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix 5: Licensing guide for Quartus II Web Edition V 5.0

1. Start by downloading `quartusii_50_web_edition_single.exe` and `nios2_5.0_eval.exe` to your computer from `http://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp`. You have to fill out a form to be able to download. Make sure that when you fill out the form, you give the correct email address. It will be needed later, the license file will be sent to that specific email address.

2. When the download process is complete double click on `quartusii_50_web_edition_single.exe` and the installation program guides you through the installation process. Repeat the same procedure with `nios2_5.0_eval.exe`. Remember to NOT install Quartus II and Nios II in a directory which contains space characters.

3. When you are done with the installations you need a license file for Quartus II to work correctly. Go back to `http://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp` and click on “Get a License File (required)” See Figure 1 below.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Download</th>
<th>File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>quartusii_50_sp1_web_edition_single.exe</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2) Download Nios II Embedded Processor, Evaluation Edition</td>
<td>368 MB</td>
</tr>
<tr>
<td></td>
<td><code>nios2_5.0_eval.exe</code></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1. Quartus II Web Edition download page*

Now you will have to provide your network interface card (NIC) number. You can get your NIC easily by typing `ipconfig /all` in the **Command Prompt**. see Figure 2 on the next page.
You can find the **Command Prompt** in the windows start menu, by clicking **Start>Programs>Accessories>Command Prompt**. When you are done, Altera will send an attachment with a license file to your email account.

4. The next step is to download the license file and point out the location of the file to the Quartus II software. Start by making a new directory somewhere on your computer, for example `C:/Licensedir`. Download the attachment you got from Altera in this folder. Start Quartus II you just installed. In Quartus II a window will open called **License Setup Required**. Click on **Specify valid license file** and then **OK**. See figure 3 below.

![Figure 3. Quartus II License](image)

Then click the **browse button** and point out the directory where the license file is located. See Figure 4 on the next page. Then click **OK**.
You have now completed the installation and licensing. You can start using the Quartus II and Nios II IDE.
Appendix 6: Creating a symbol file from an existing VHDL/Verilog file(s) in Quartus II Web Edition v 5.1

To be able to create a symbol file an existing project must be opened or created in Quartus II. Also the VHDL file(s) from which the symbol file will be created must be added to the project. If the VHDL file(s) already exist in the project skip step 1.

1. Importing a VHDL file into an existing Quartus II project.
Start by clicking on the Files tab in the Project Navigator window (see Figure 1 below).

![Figure 1. Files tab](image1)

Right click on the folder named Device Design Files then click on Add/Remove Files in Project. Now click the browse button and point out the directory where the VHDL file(s) are located (see Figure 2 below). When you are done click Add then click OK.

![Figure 2. Files window](image2)
2. Converting the file(s) to a symbol file.
The added VHDL file(s) can now be located in the folder named Device Design Files in the Files tab. To covert the desired VHDL file(s) to a symbol file right click on the VHDL file which defines the top entity of your design and select Create Symbol File for Current File (see Figure 3 below).

![Figure 3. Creating Symbol Files](image)

Now Quartus II creates a symbol file out of the selected VHDL file(s). This symbol file can be found in the symbol library under the project folder. The symbol library can be accessed by clicking on the button. This button can be accessed while editing a block diagram file (.bdf).