|  |
| --- |
| AGSTU |
| VGA COMMUNICATION COMPONENT |
| Designing both hard – and software for a VGA intellectual property core with VHDL and low level C programing |
|  |
| **Saif Saadaldin** |
| **2021-01-11** |

|  |
| --- |
| The constructed VGA IP component can be reused in other projects. It was built by first writing the VGA hardware files in VHDL and testing the code. Secondly an hw.tcl file and bus interface was constructed in Qsys (Quartus). The IP components was tested together with a CPU system that was downloaded to a DE10-Lite FPGA board with Altera MAX10 chip (10M50DAF484C7G). C code functions was written to draw different shapes on a VGA monitor. The delivered system contains all source files, documentation, and example code. |

[1 INTRODUCTION 2](#_Toc61650292)

[2 REQUIREMENT SPECIFICATION 3](#_Toc61650293)

[3 TIMETABLE 4](#_Toc61650294)

[4 SYSTEM ARCHITECTURE 5](#_Toc61650295)

[4.1 Hardware part (VHDL) 6](#_Toc61650296)

[4.1.1 VGA\_IP 7](#_Toc61650297)

[4.1.2 VGA\_RAM 8](#_Toc61650298)

[4.1.3 vga\_sync\_signals 9](#_Toc61650299)

[4.2 Software part (C programming) 10](#_Toc61650300)

[5 VERIFICATION 11](#_Toc61650301)

[5.1 Test Protocol 11](#_Toc61650302)

[5.2 Result from simulation 12](#_Toc61650303)

[6 VALIDATION 13](#_Toc61650304)

[6.1 Result from validation 14](#_Toc61650305)

[7 CONCLUSIONS 16](#_Toc61650306)

[8 APPENDIX. 17](#_Toc61650307)

# INTRODUCTION

The aim of this project is designing a VGA IP-core that can be used in other projects. The hardware programing language is VHDL and the software programing is done by C programing. Both hard – and software files were already constructed in previous projects and in this project everything will be gathered and saved in one IP directory.

# REQUIREMENT SPECIFICATION

In table 1 below the customer requirements for the constructed system are listed.

Table 1: System requirements

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Description** | **Status** |
| 1 | Build IP core for a VGA component by reusing previous VHDL and C code from previous projects in the area. | Done |
| 2 | Using C programing design these function that will use the constructed VGA-IP core: print\_pix(), print\_hline(), print\_vline(), print\_char(), read\_pixel\_ram\_int(), clear\_screen(), print\_circle() | Done |

# TIMETABLE

Planed schedule:

* Week 1 – Learning the tool Qsys and gathering information about how the hardware and software files will be grouped together in one construction.
* Week 2 – implementation, simulation and validation of the construction.
* Week 3 – implementation, simulation and validation of the construction.

It was planned that the total time for this project will be 80 hours. After the project was done, the total time was 100 hours. That means the cost of the projects total cost is 50k Swedish krona

# SYSTEM ARCHITECTURE

The constructed VGA component is the red box and the blocks inside it, se figure 1. The figure shows an example of how the VGA component can be used together with a CPU system and VGA monitor. The vhdl files VGA\_SYNC\_SIGNALS.vhd and VGA\_RAM.vhd were constructed in previous projects and was reused in this project. The file VGA\_IP.vhd was built during this project to act as a file wrapper and handles the read and write logic from and to a CPU system (the green box), as shown in figure 1.

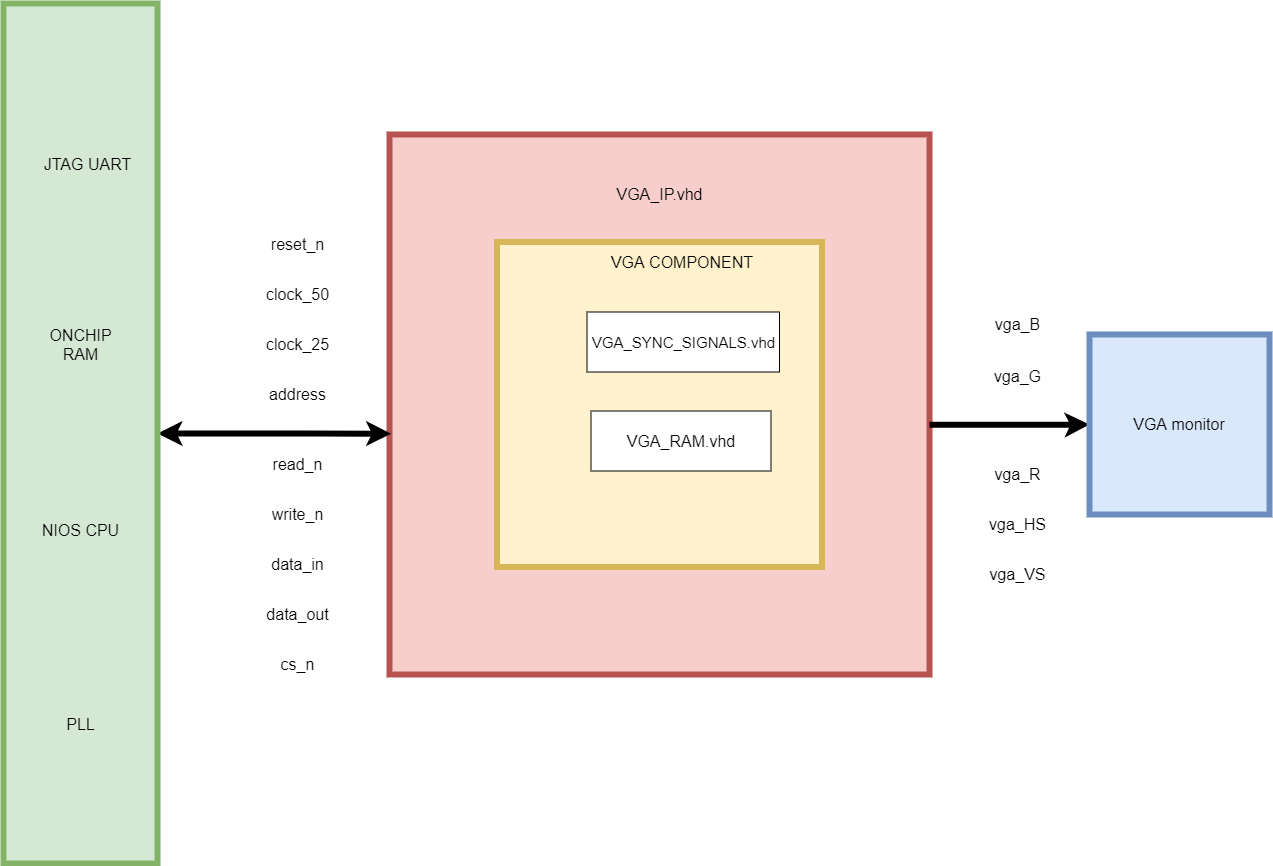
This chapter will focus on the VGA components, its parts and in – and outputs.

Figure 1: Block diagram of the system

## Hardware part (VHDL)

The project used the tool Quartus Prime Lite 18.1 to write the VHDL code. Figure 2 shows the RTL picture of the entire VGA IP component. The task of module VGA\_RAM and VGA\_SYNC\_SIGNALS is to communicate with the VGA hardware (VGA monitor). The top file VGA\_IP as shown in figure 1 is to make it possible for the IP core to communicate with a bus system. In this case its Avalon bus system. The VGA\_IP file is the part within the red lines in figure 2 and it consist of the flip-flops, the multiplexer and the and-gates shown in figure 2.

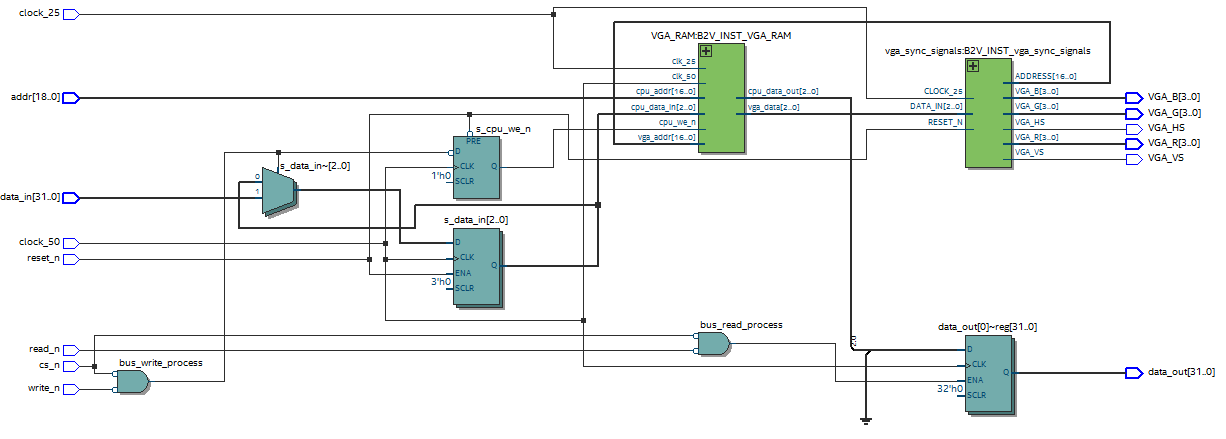


Figure 2: RTL schematic over the IP core

### VGA\_IP

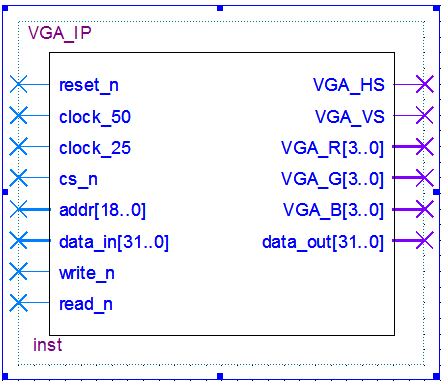
The top\_vhdl component has the following in and outputs in figure 3. Table 2 shows the type and function of each in and output pin of this top component.

Figure 3: Component’s symbol

Table 2: in and outputs for VGA\_IP

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| reset\_n | In std\_logic | System reset |
| clock\_50 | In std\_logic | System clock 50MHz |
| clock\_25 | In std\_logic | Clock to drive VGA signals 25MHz |
| cs\_n | In std\_logic | Chip select |
| addr | In std\_logic\_vector(18 downto 0) | Address to write to or read from |
| data\_in | In std\_logic\_vector(31 downto 0) | Incoming data from CPU |
| write\_n | In std\_logic | Write enable |
| read\_n | In std\_logic | Read enable |
| VGA\_HS | Out std\_logic | Horizontal sync signal needed to drive the VGA |
| VGA\_VS | Out std\_logic | Vertical sync signal needed to drive the VGA |
| VGA\_R | Out std\_logic\_vector(3 downto 0) | VGA signals to activate red color on screen |
| VGA\_G | Out std\_logic\_vector(3 downto 0) | VGA signals to activate green color on screen |
| VGA\_B | Out std\_logic\_vector(3 downto 0) | VGA signals to activate blue color on screen |
| data\_out | Out std\_logic\_vector(31 downto 0) | data out to CPU when requested |

### VGA\_RAM

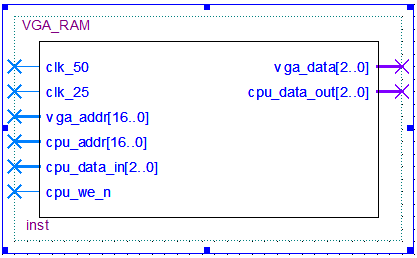
This dual-port ram is used to store the written data from the CPU and which will be drawn on the VGA monitor. The component also respond for CPU requests when it asks for the stored VGA color data at specific memory address. The component’s in and outputs can be seen in figure 4 and table 3. The ram memory consist of 320\*240 address where each address represents a pixel on the VGA monitor.

Figure 4: component’s symbol

Table 3: in and outputs for VGA\_RAM

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| clk\_50 | In std\_logic | System clock |
| clk\_25 | In std\_logic | Clock to drive VGA signals 25MHz |
| vga\_addr | In std\_logic\_vector(16 downto 0) | Requested address to draw form VGA |
| cpu\_addr | In std\_logic\_vector(16 downto 0) | Address to write to or read from |
| cpu\_data\_in | In std\_logic\_vector(2 downto 0) | The data the CPU saving to the RAM |
| cpu\_we\_n | In std\_logic | Write enable |
| vga\_data | Out std\_logic\_vector(2 downto 0) | Color data to the VGA |
| cpu\_data\_out | Out std\_logic\_vector(2 downto 0) | Requested data from CPU |

### vga\_sync\_signals

This component is responsible to synchronize the VGA signals in order to get proper resolution and color on the VGA monitor. The components counts which pixel to draw for the moment and asks the VGA\_RAM for the VGA color for counted pixel number.

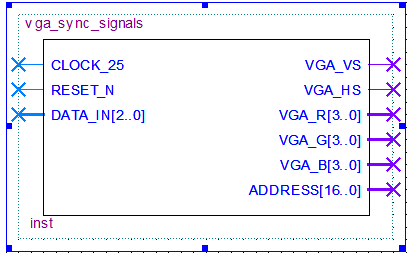
Figure 5: component’s symbol

Table 4: in and outputs for vga\_sync\_signals

|  |  |  |
| --- | --- | --- |
| **Signal** | **Type** | **Info** |
| CLOCK\_25 | In std\_logic | System clock |
| RESET\_N | In std\_logic | system reset |
| DATA\_IN | In std\_logic\_vector(2 downto 0) | data in from VGA\_RAM |
| VGA\_VS | Out std\_logic | Vertical sync signal needed to drive the VGA |
| VGA\_HS | Out std\_logic | Horizontal sync signal needed to drive the VGA |
| VGA\_R | Out std\_logic\_vector(3 downto 0) | VGA signals to activate red color on screen |
| VGA\_G | Out std\_logic\_vector(3 downto 0) | VGA signals to activate green color on screen |
| VGA\_B | Out std\_logic\_vector(3 downto 0) | VGA signals to activate blue color on screen |

## Software part (C programming)

When the VHDL files in section 4.1 were done and verified it was time to create the new VGA IP with the help of the tool Platform Designer in Quartus. When that is done we can then add the VGA IP to a complete CPU system as the one shown in figure 1. The CPU system is then downloaded to the DE10-Lite FPGA board.

By using C programing language it is possible for the software designer to communicate with VGA hardware design and write C code that draws different things on the VGA monitor. The communication with the hardware is done by HAL commands (Hardware abstraction layer) to write and read data from the VGA IP core.

All the functions listed in table 1 are done in C and can be found in GitHub as mentioned in section 8 of this report.

# VERIFICATION

## Test Protocol

Only the write and read cycles to the VGA IP will be simulated in this projects. The verification is done by building a simple testbench and simulate the results on ModelSim. See table 5 for the test cases. This simulation is for the VHDL files.

Table 5: test protocol for verification

|  |  |  |  |
| --- | --- | --- | --- |
| **Case ID** | **Description** | **Expected behavior** | **Result** |
| A | Write cycle to the VGA IP core | write\_n = ‘0’, read\_n = ‘1’, cs\_n = ‘0’,  address and data\_in should be similar to data in testbench | Ok |
| B | Read cycle from the VGA IP core | write\_n = ‘1’, read\_n = ‘0’,  cs\_n = ‘0’, address and data in should be similar to data\_in testbench | Ok |

## Result from simulation

**Case ID: A**

The address in the test bench for this test is 0x00000 and the data to write is 0x00000001. As figure 6 shows that the signals for the write cycle are correct and the data and address are coming in right positions.

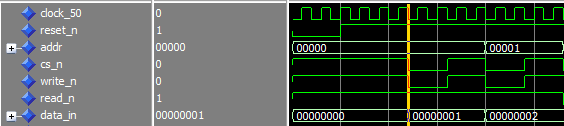


Figure 6: RTL simulation of a CPU write cycle to the VGA IP core

**Case ID: B**

The address to read from for this case is 0x00000 and should now have the data 0x00000001 in it. Figure 7 below shows that signals for this test are right and according to table 5. Its good to mention that the right value of the data\_out is available two clock cycle after an active low read\_n signal. When designing the VGA component in Qsys the designer should specifies after how many clock the data\_out is stabile/available.

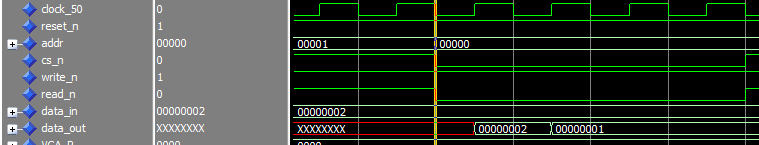


Figure 7: RTL simulation of CPU read cycle from RAM memory on VGA IP core

# VALIDATION

The validation is done by loading the hardware design to the DE-10 Lite FPGA board and drawing two different figures on the VGA monitor. Table 6 shows the testes to validate. The C code functions will be used here to draw two different shapes.

Table 6: validation test cases

|  |  |  |
| --- | --- | --- |
| **Case ID** | **Description** | **Result** |
| A | Draw horizontal line that is 50 pixel long, starts at coordinates (10, 10) and is red colored. See figure 8 | Ok |
| B | Draw a circle with a radius of 10 pixel, has center coordinates at (100, 200) and is blue colored. See figure 9 | Ok |
| C | Write my name “Saif Saadaldin” on a white VGA monitor with black letter background. See figure 10 | Ok |

## Result from validation

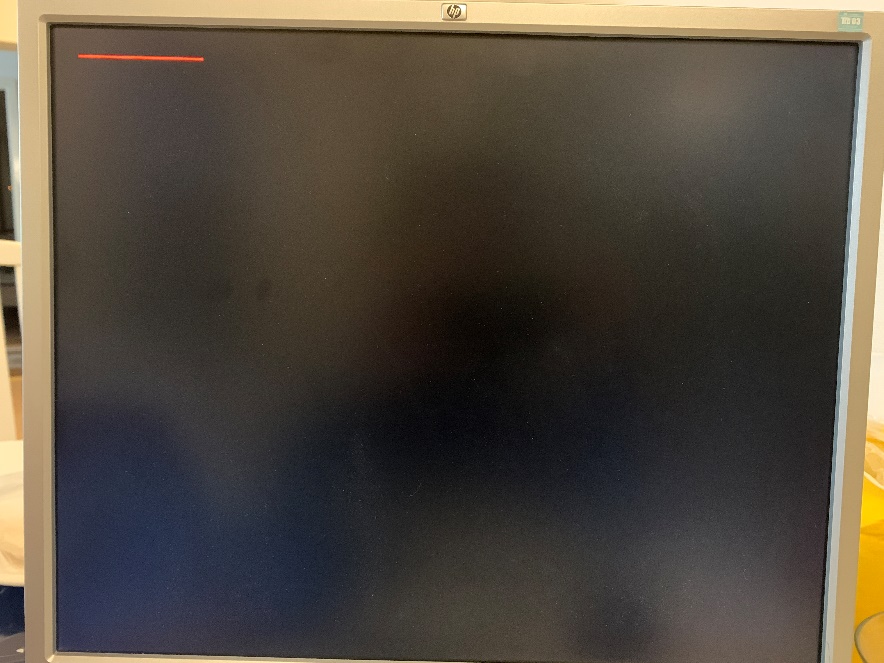
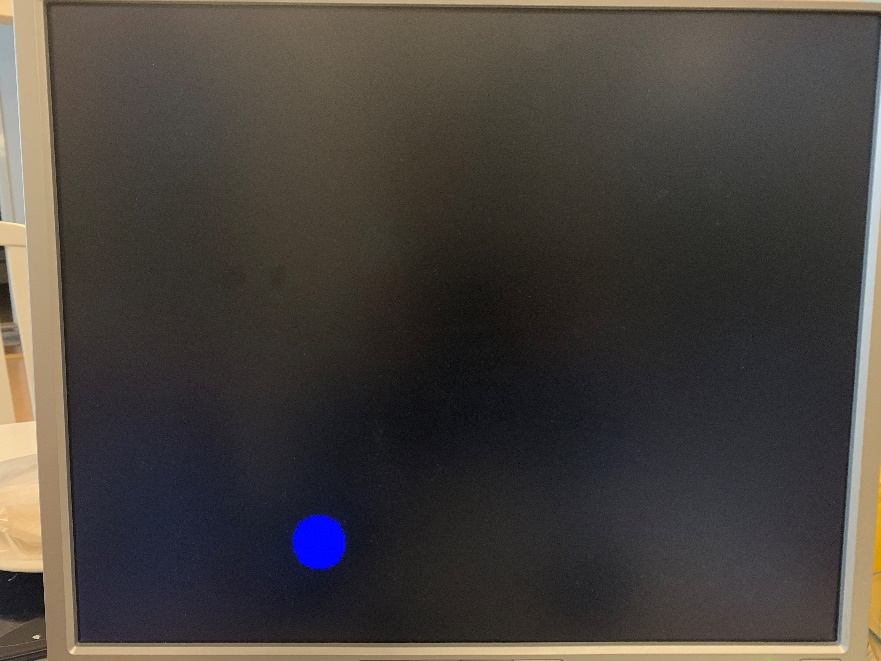
**Case ID: A**

Figure 8: result for case A

**Case ID: B**

Figure 9: result for case B

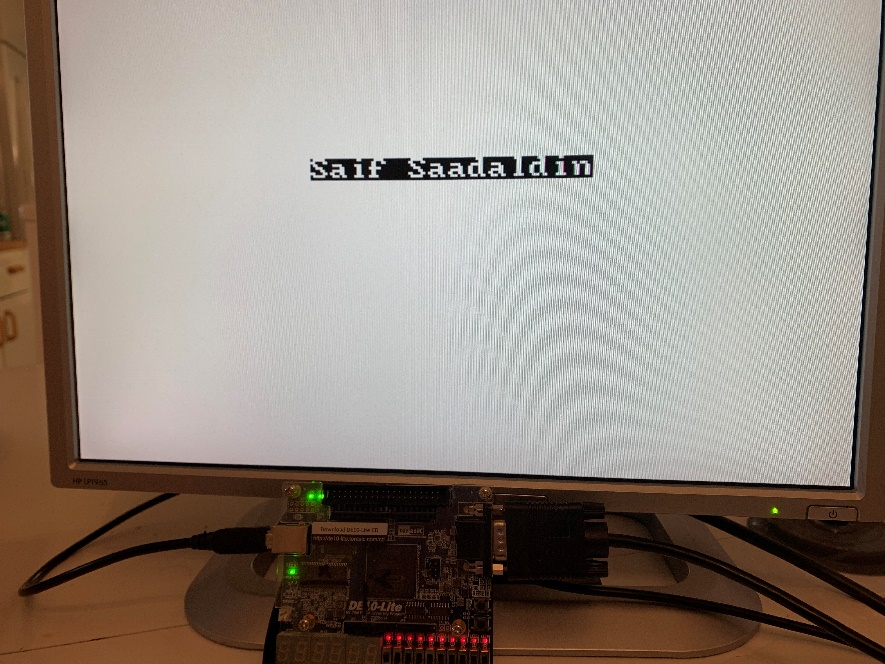
**Case ID: C**

Figure 10: result for case C

# CONCLUSIONS

The idea of building an IP component make it very flexible to reuse the component in other projects in the future.

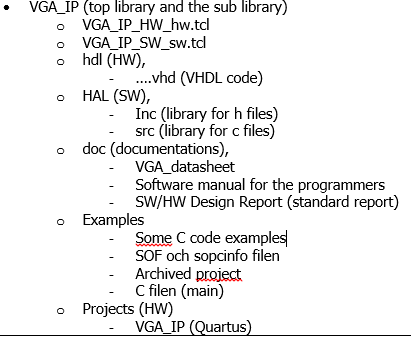
It’s very important to have a file structure as figure 11 below in order to easily implement the IP component in other projects.

Figure 11: general file structure of an IP component

# APPENDIX.

The documentation for the VGA IP files can be found in GitHub via this link: <https://github.com/S81f/VGA_intellectual_property>