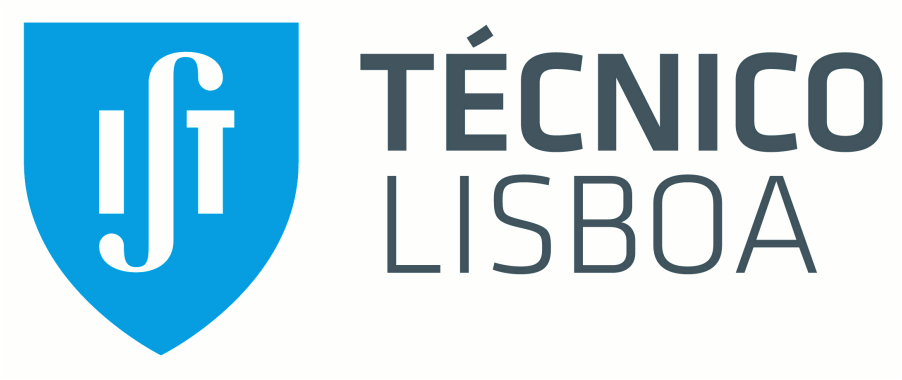

Computer Electronics

RISCV System on a Chip

Grupo 3

Francisco Mendes - Nº 84055

Gaspar Ribeiro - Nº 84059

October 21, 2019

Contents

1 Introduction

The RISCV - SoC project intents to develop a simple complete system of a chip - SoC - on a Basys2 FPGA. On this project it is going to be used the PicoRV32I CPU core, an open-source implementation of the RISC-V RV32IMC Instruction Set. This implementation consists of a simple CPU with only integer support where peripherals such as Display throw VGA, Keyboard with PS/2, LEDs and Slide Switchs and 7 segments display are going to be connected to the CPU.

The objective is to construct a simple operating system where the user can input text that is displayed on the display and the operating system can process the text input and output not only responses on the display, but also, change the state of the other peripherals connected to the FPGA.

The option for the this CPU Core brings added benefits and challenges. First, it is a Core based on the new and trendy RISC-V ISA, this allows for the exploration of this new technology that intents to quickly become one of the industry standards. Being based on the RiscV ISA brings the benefit that a complete development tool-chain is widely available (*gcc*, binutils, etc...). However, the Basys 2 FPGA is a small, education level FPGA, so extra care and tinkering has to be done the ensure that the desired circuit fits into the FPGA.

2 Block Diagram

The RISCV-SoC block diagram is shown in Fig. 7. RISCV-SoC contains two main modules, the CPU and the Peripherals. The CPU module is responsible for controlling the system, actuating on the peripherals modules and run the main program.

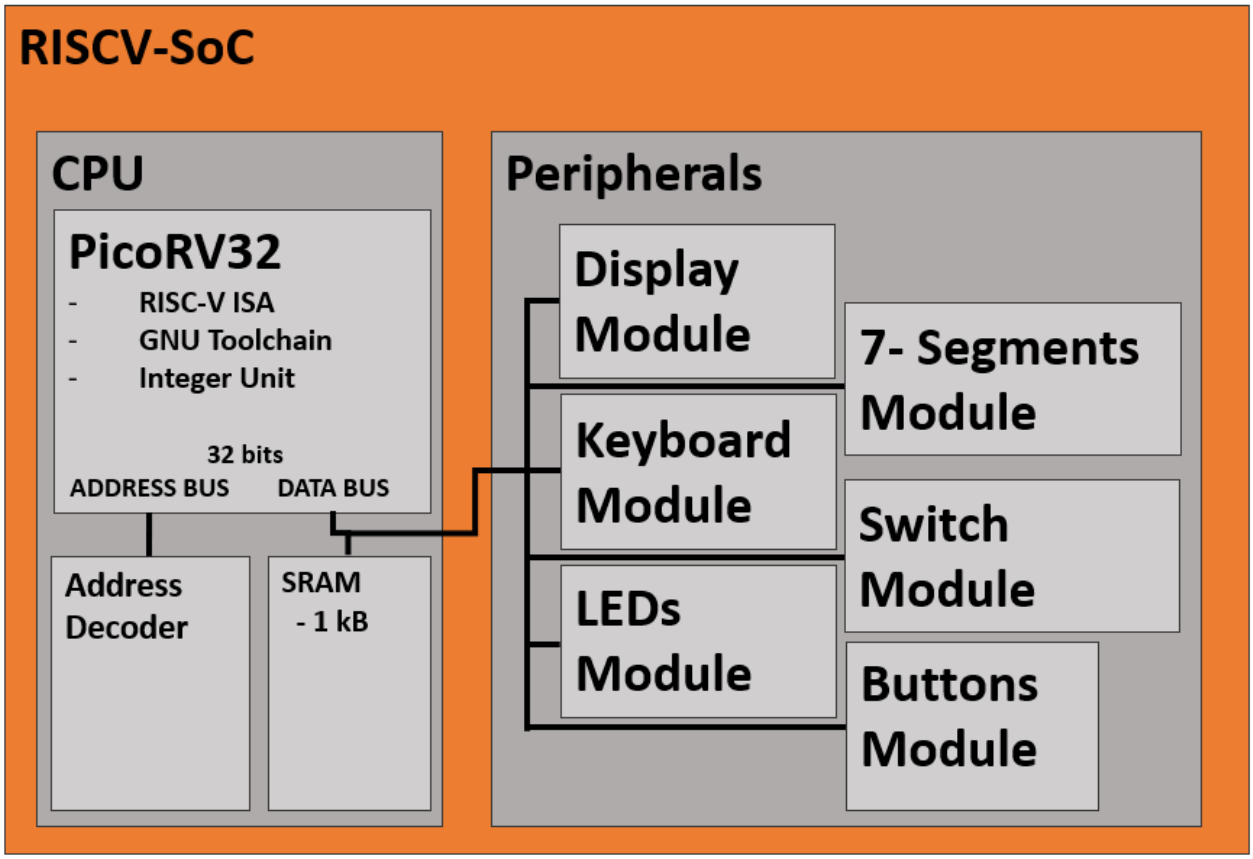

Figure 1: RISCV -SoC - Block Diagram.

2.1 CPU

The CPU Core used in this project is a RicoRV32i, open-source processor (https://github.com/cliffordwolf/picorv32). PicoRV32 is a integer only CPU core that implements the RISC-V RV32IMC Instruction Set and has an attached address decoder, in order to communicate with the different peripherals, and an SRAM module with 1kB of available memory.

2.2 Peripherals

The peripherals module is composed of a set of sub-modules each responsible to control - read or write - the corresponding physical peripheral that the system can connect to.

3 Program

The objective of the RiscV - System on a Chip is to create a simple terminal that enables the user to input words and numbers, with the keyboard, and the processor should be able to interpret the message and actuate in accordance to it. Figure 2 demonstrates a example of the system working.

Figure 2: System Objective.

3.1 Main Routine

The main loop of the program flowchart is present figure 3. The program is always running when the system is connected to power and it consists in three different stages, reading, interpretation of the input and output/actuation.

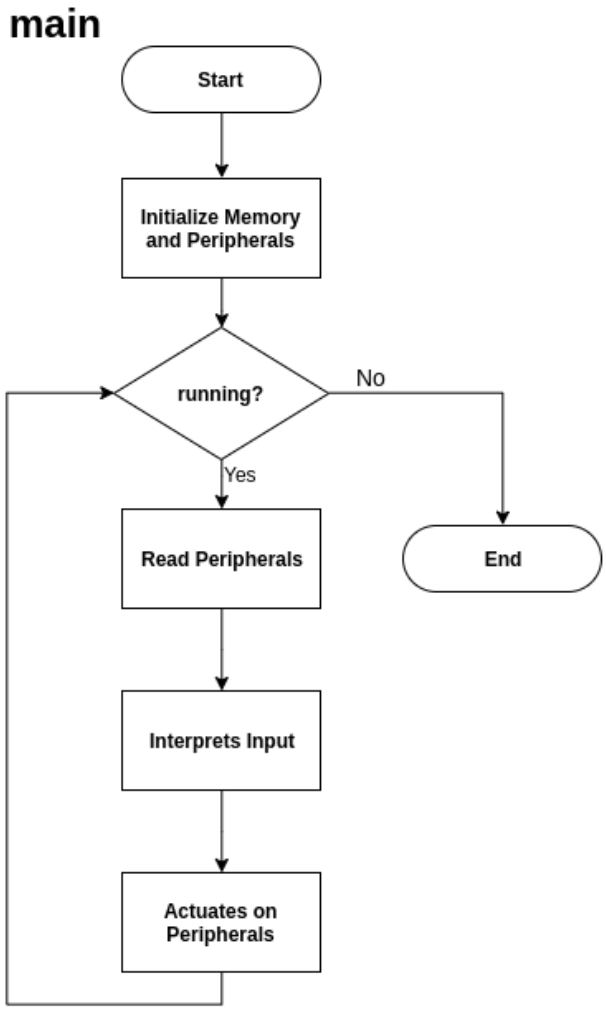

Figure 3: Main Routine Flowchart.

3.1.1 Read Peripherals Routine

On the read peripherals routine, the three types of inputs are scanned (buttons, slides and keyboard) and the interaction is saved on memory to post analysis.

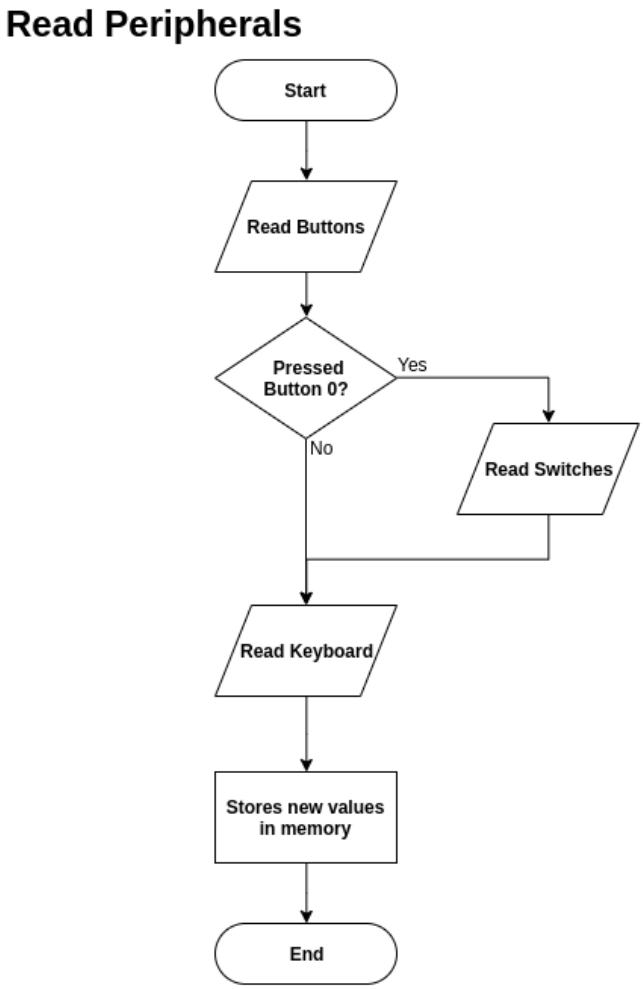

Figure 4: Read Peripherals Routine Flowchart.

3.1.2 Interpret Peripherals Routine

On this routine, the memory is read and its content is analysed, storing the result and the intended action for the next routine.

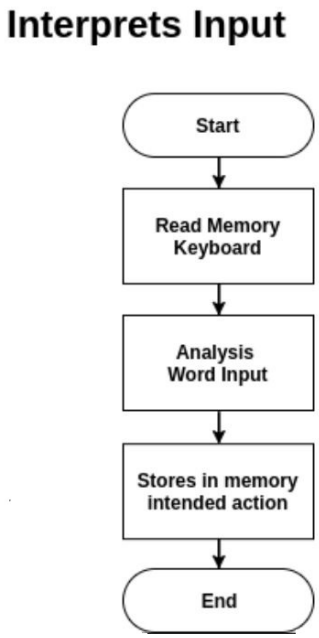

Figure 5: Interpret Routine Flowchart.

3.1.3 Actuate on Peripherals Routine

On the actuate routine, content to be written on display is sent to the VGA interface and the corresponding LEDs and 7 segments display are turn on accordingly.

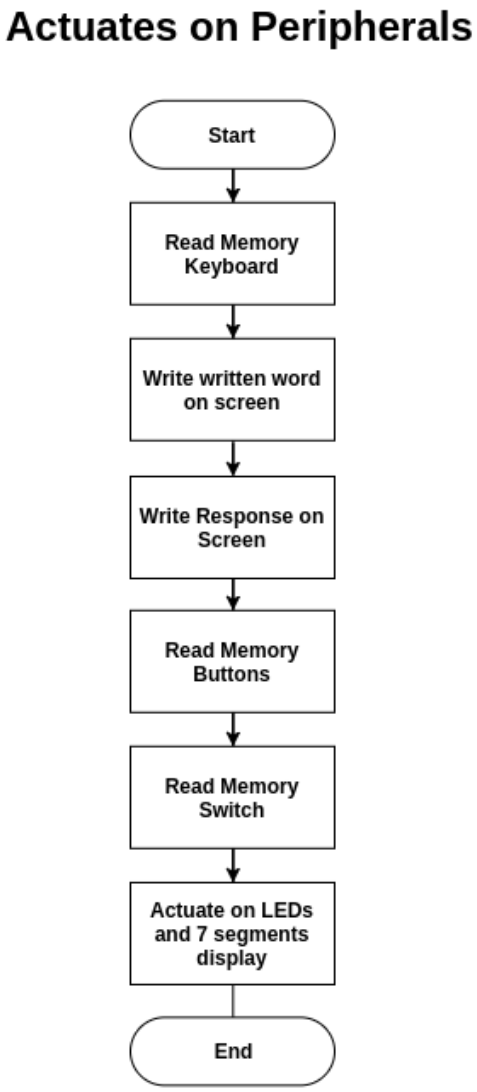

Figure 6: Actuate Routine Flowchart.

4 Interface Signals

The interface signals of the RISCV-SoC is described in the Table 1.

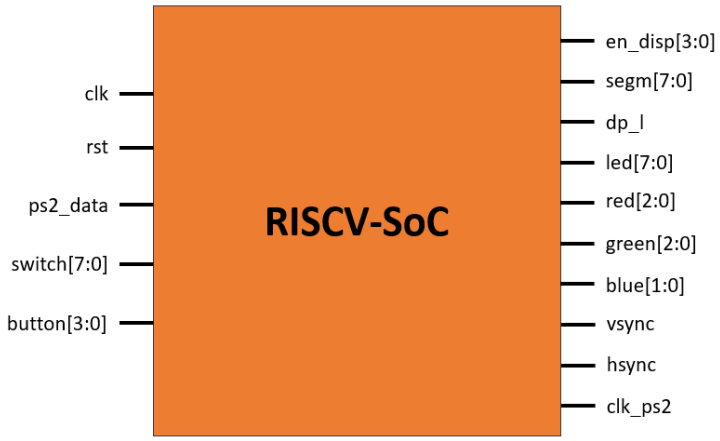

Figure 7: RISCV - SoC - Block Diagram.

Name	Direction	Description
clk	Input	Clock signal
rst	Input	Reset signal, asynchronous, on high
ps2_data [7:0]	Input	Data from the keyboard
switch [7:0]	Input	Data from the switches
button[3:0]	Input	Data from the buttons
en_disp [3:0]	Output	On/off each of the 4 seven segments display
segm [7:0]	Output	Activated leds on the 7 segment display
dp_l	Output	Activates 7 segment display
led [7:0]	Output	LEDs to bue turned on
red [2:0]	Output	Value of red
green [2:0]	Output	Value of green
blue [1:0]	Output	Value of blue
vsync	Output	VGA Vertical Sync
hsync	Output	VGA Horizontal Sync
clk_ps2	Output	Clock for the PS/2 keyboard

Table 1: Interface signals.

5 Memory Map

The memory map of the system, as seen by RISCV-SoC programs, is given in Table 2.

Mnemonic	Address	Read/Write	Read Latency	Description
DISPLAY_BASE	0	Write	NA	VGA Module Peripheral
KEYBOARD_BASE	1	Read	1	PS2 Peripheral
LED_BASE	2	Write	NA	LED Peripheral
SEVEN_BASE	3	Write	NA	Seven Segments Display Peripheral
SWITCH_BASE	4	Read	1	Switch Peripheral
BUTTON_BASE	5	Read	1	Button Peripheral
SRAM_BASE	6	Read+Write	0	SRAM peripheral

Table 2: Memory Base base

6 Peripherals

The Basys 2 development board contains peripherals responsible for the input/output of data to and from the board, that includes an array of LEDs, a 7 segment display, push buttons and switches which can be used to interact with the Risc V core implemented on the board. In addiction, the board also has a VGA output port that will be used to connect a monitor and a PS/2 input port to be used with a keyboard

6.1 CPU

The CPU responsibility is to enable the modules that interact with the physical peripherals and write/read to/from the common data-bus that connects the two sybsystems.

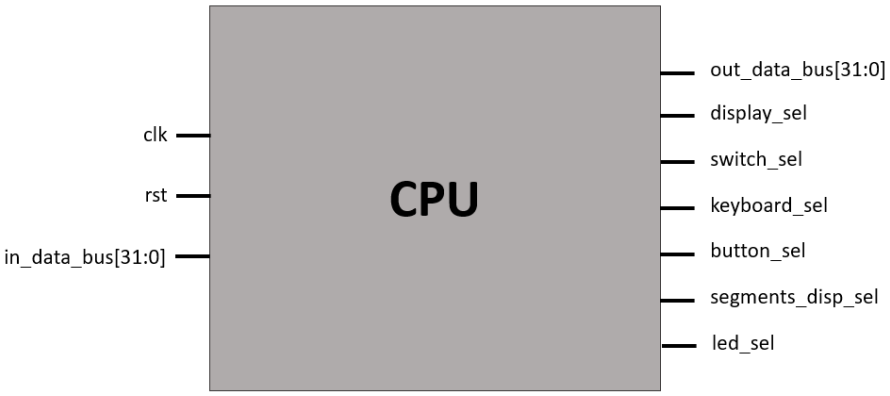

Figure 8: CPU module - symbol.

Name	Direction	Description
clk	Input	Clock signal
rst	Input	Reset signal, asynchronous, on high
in_data_bus [31:0]	Input	Data from the peripherals
out_data_bus [31:0]	Output	Data to the peripherals
display_sel	Output	Activate the display module
switch_sel	Output	Activates the Switch Module
keybaord_sel	Output	Activates the keyboard Module
button_sel	Output	Activates the button Module
segemnts_disp_sel	Output	Activates the 7 segments display Module
led_sel	Output	Activates the LEDs module

Table 3: CPU Module signals.

6.1.1 PicoRV32I

The PicoRV32i symbol is shown in figure 9. PicoRV32i is a simple processor that implements the basic RISC-V Instruction Set Architecture. It includes the complete RISC-V tool-chain, allowing for the development of C programs and the guaranteed execution of the same. It has a common data-bus where all the peripherals are connected to.

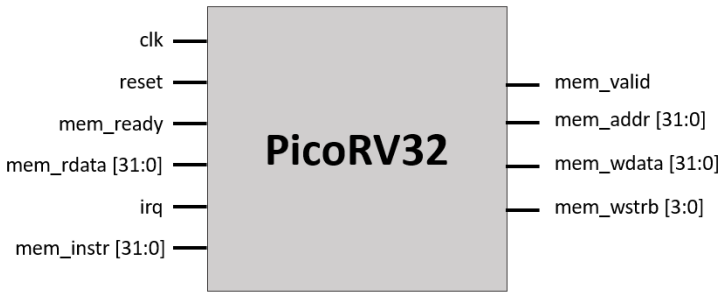

Figure 9: Picorv32i - symbol.

Name	Direction	Description
clk	Input	Clock signal
rst	Input	Reset Signal, asynchronous, on high
mem_ready	Input	Indicates that the memory is ready to be written/read
mem_rdata[[31:0]	Input	data read from the memory
irq	Input	Interruptions - Not in use
mem_instr[31:0]	Input	Current Instruction
mem_valid	Output	Enable of sram module
mem_addr[31:0]	Output	Address position to be read or write
mem_wdata[31:0]	Output	Data to be writeen on memory
mem_wstrb[3:0]	Output	Enable of the memory block

Table 4: PicoRV32 module signals.

Name	Address offset	Read/Write	Description
mem_ready_reg	0	Write	Reads if the memory is ready to be read
rdata_reg	4	Write	Content of the memory
mem_ready_reg	8	Write	Indicates that CPU is ready to write
wdata_reg	12	Read	Content to be written to the memory
wstr_reg	16	Read	Each 4 bits that wants to be written

Table 5: PicoRV32 module registers map.

6.1.2 Address Decoder

The address decoder symbol is shown in figure 10. It is responsible for decoding the address of the peripherals and create the enable signals that enable the peripherals modules.

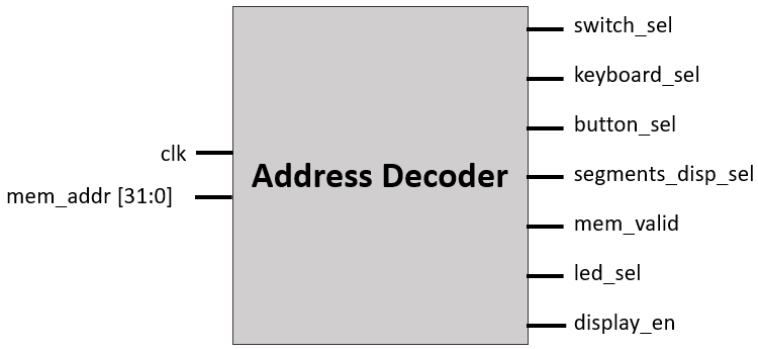

Figure 10: Address Decoder - symbol.

Name	Direction	Description
clk	Input	Clock signal
mem_addr [31:0]	Input	Address to be decoded
switch_sel	Output	Enable switch module
keyboard_sel	Output	Enable ps/2 module
button_sel	Output	Enable button module
segments_disp_sel	Output	Enable 7 segments display module
mem_valid	Output	Enable memory module
led_sel	Output	Enable LEDs module
display_en	Output	Enable VGA module

Table 6: Address decoder signals

Name	Address offset	Read/Write	Description
mem_addr_reg	0	Write	Address to be decoded

Table 7: Address Decoder module registers map.

6.1.3 SRAM

The SRAM symbol is shown in figure 11. This component has 1kB of byte addressable memory where the main program and heap and stack are stored during the execution.

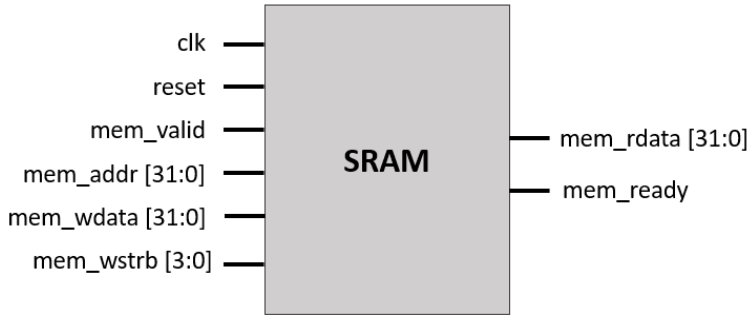

Figure 11: SRAM - symbol.

Name	Direction	Description
clk	Input	Clock signal
rst	Input	Reset Signal, asynchronous, on high
mem_valid	Input	Enable of SRAM module
mem_addr[31:0]	Input	Address position to be read or write
mem_wdata[31:0]	Input	Data to be written on memory
mem_wstrb[3:0]	Input	Enable of the memory block
mem_rdata[31:0]	Output	Data read from the memory
mem_ready	Output	Indicates that the memory is ready

Table 8: SRAM module signals.

Name	Address offset	Read/Write	Description
mem_ready_reg	0	Read	Reads if the memory is ready to be read
rdata_reg	4	Read	Content of the memory
mem_ready_reg	8	Write	Indicates that CPU is ready to write
wdata_reg	12	Write	Content to be written to the memory
wstr_reg	16	Write	Each 4 bits that wants to be written

Table 9: SRAM module registers map.

6.2 Display

The VGA module symbol is shown in figure 12, it has a memory correspondent to the value to be displayed on each of the pixels. The display is simplified to supports 40 column and 28 lines of characters, where each character is composed by 6*4=24 pixels.

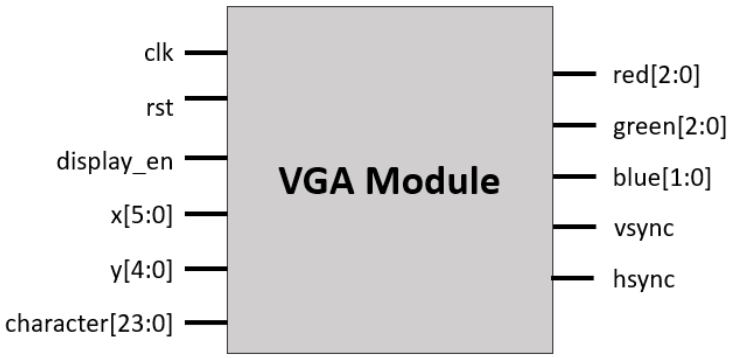

Figure 12: VGA module - Block Diagram.

6.3 Keyboard

The Keyboard module symbol is shown in figure 13. This module allows an external keyboard to be connected through the PS/2 port and for the system to receive the pressed character.

Name	Direction	Description
clk	Input	Clock
rst	Input	Reset Signal, asynchronous, on high
display_en	Input	Enables Display module
x[5:0]	Input	X Coordinate
y[4:0]	Input	Y Coordinate
character[23:0]	Input	Character bitmap
red[2:0]	output	Red color value
green[2:0]	output	Green color value
blue[1:0]	output	Blue color value
hsync	output	Horizontal sync
vsync	output	Vertical sync

Table 10: VGA module signals.

Name	Address offset	Read/Write	Description
x_reg	0	Write	X coordinate where to write the character
y_reg	4	Write	Y coordinate where to write the character
character_reg	8	Write	24 pixels of each bitmap character

Table 11: VGA module registers map.

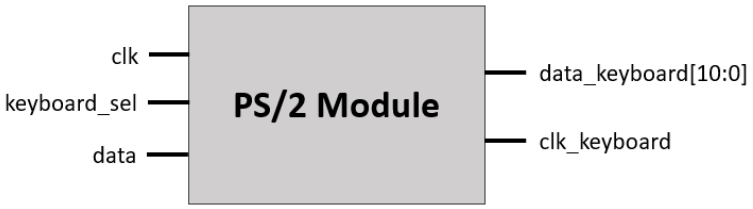

Figure 13: PS/2 module - Block Diagram.

Name	Direction	Description
clk	Input	Clock
keyboard_sel	Input	Activate PS/2 module
data	Input	Value from keyboard
data_keyboard[10:0]	Output	Value from module to CPU
clk_keyboard	Output	Clock fed to the keyboard

Table 12: PS/2 module signals.

Name	Address offset	Read/Write	Description
keyboard_reg	0	Read	Key pressed on the keyboard

Table 13: PS/2 module registers map.

6.4 LEDs

To turn on any of the 8 LEDs of the board, it's respective bit must be '1' and the led_sel signal also needs to be set to '1' by the CPU.

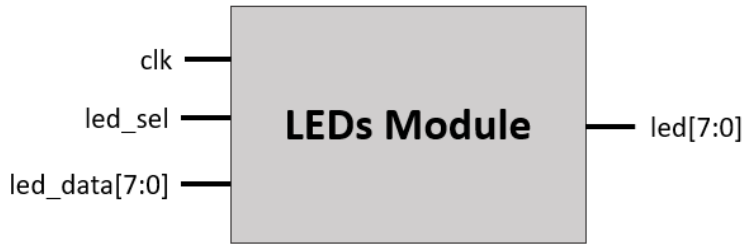

Figure 14: LEDs module - Block Diagram.

Name	Direction	Description
clk	Input	Clock
led_sel	Input	Activate the module
led_data[7:0]	Input	LEDs to be turned on if the module is active
led [7:0]	Output	LEDs to be turned on

Table 14: LEDs module signals.

Name	Address offset	Read/Write	Description
led_reg	0	Write	Leds to be turned on

Table 15: LEDs module registers map.

6.5 Push Buttons

The system has 4 push buttons. Value is 1 when the slide is on the top position and the module allows for the read of the 8 slides in parallel.

Figure 15: buttons module - Block Diagram.

6.6 Slide Switch

The system has 8 slide switches buttons. Value is 1 when the button is press and the module allows for the read of the 4 buttons in parallel.

Name	Direction	Description
clk	Input	Clock
button_sel	Input	Activates the Button Module
button[3:0]	Input	Value from the buttons
button_data[3:0]	Output	Value read from the buttons to the CPU

Table 16: Buttons Module Signals.

Name	Address offset	Read/Write	Description
button_reg	0	Read	Buttons pressed

Table 17: Button module registers map.

Figure 16: Switches module - Block Diagram.

Name	Direction	Description
clk	Input	Clock
switch_sel	Input	Activates the Switch Module
switch[7:0]	Input	Value read from the switches
switch_data[7:0]	Output	Value read from the buttons to the CPU

Table 18: Switch Module Signals.

Name	Address offset	Read/Write	Description
switch_reg	0	Read	Position of the 8 switches

Table 19: Switch module registers map.

6.7 7 segment display

The module receives a number between 0-9999 and activates the corresponding 7 segments display needed to display the integer value.

Figure 17: 7 segment display module - Block Diagram.

Name	Direction	Description
clk	Input	Clock
segments_disp_sel	Input	Activates the 7 Segments Display Module
segments_disp[12:0]	Input	Integer value between 0-9999 to be displayed
en_disp[3:0]	Output	Active 7 segment displays
segm[7:0]	Output	Active leds on segment
dp_l	Output	Activates electronic circuit for segments

Table 20: 7 Segment Display Module Signals.

Name	Address offset	Read/Write	Description
7seg_reg	0	Write	Number to be writen on the segmetns display

Table 21: 7 segments module registers map.

7 Implementation Results

Resource	*DSP*	*FF*	*LUT*	LUTRAM	*BRAM*	**Timing - Setup Worst Negative Slack [ns]**
# Estimation						
Utilisation [%]						

Table 22: Implementation of RISCV-SoC.

8 Conclusions