Calculator

User Guide

Jorge Simões
jorgesimoes@tecnico.ulisboa.pt
José Carolino
jose.carolino@tecnico.ulisboa.pt

January 29, 2020

Contents

List of Tables

List of Figures

1 Introduction

Calculator is a simple 2 operands calculator with a small set of operations. It is capable of doing sums, subtractions, multiplications, exact divisions, remainder and powers. The numbers and the operations are both chosen using the buttons and displayed on the 7 segment display. The result of the operation and its operands are displayed on a VGA display.

This program uses picoVersat hardware controller and has peripherals to control the buttons, 7 segment display and VGA display. It has a multiplier as peripheral because it uses the built in multiplier of FPGA. Calculator is design to be implemented on a Basys2 board but can easily be ported to another board.

2 Block Diagram

The Calculator block diagram is show in Fig. 1. This system on chip named Calculator contains picoVersat hardware controller and several peripherals. The peripherals are a multiplier, divider, a general purpose register file, a 7-Segment display controller, a button controller, a VGA controller and a Video Encoder.

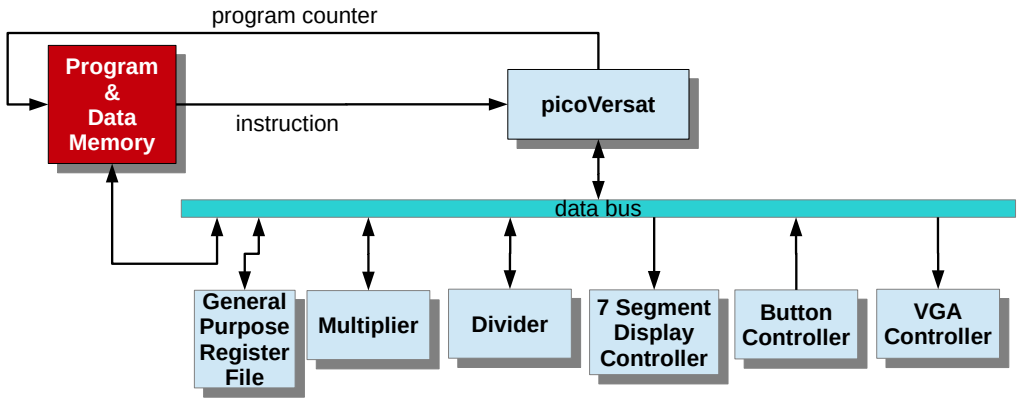

Figure 1: Calculator block diagram

2.1 picoVersat

The hardware controller used by this SoC is picoVersat. PicoVersat is designed to act like a hardware controller but in this case it is also used for calculations. All operations that are available are computed with picoVersat. The software that runs on picoVersat is capable of doing this operations on signed integers: sums, subtractions, multiplications, divisions and remainder. It can also do the following operations on unsigned integers: powers.

2.2 General Purpose Register File

The general purpose register file consists of 16 registers with 32 bits each. They are used to store intermediate values and the result of the last operation.

2.3 Multiplier

The multiplier is a peripheral designed to do all of the multiplications required by the operations, besides the multiplication operation.

2.4 Divider

The divider is a peripheral designed to do all of the divisions required. Computes the quotient and the remainder.

2.5 VGA Controller

The VGA controller is used so the program can output to the display screen the numbers chosen by the user, the operation symbol and the result of that operation. The controller passes through all pixels of the screen and for each one of them displays the desired colour using an 8-bit pallet.

2.6 7-Segment Display Controller

The 7-Segment display controller displays, when choosing the input values, the represented number in hexadecimal form and, when choosing which operation to do, the operation abbreviation.

2.7 Button Controller

The button controller purpose is so the user can make use of the 3 buttons, using them to choose which operation to do, to use the last operation result as an input number for the current operation and to increase the value of the input number.

3 Interface Signals

The symbol of this system is shown in Fig- 2 and the interface signals of the Calculator are described in Table 1.

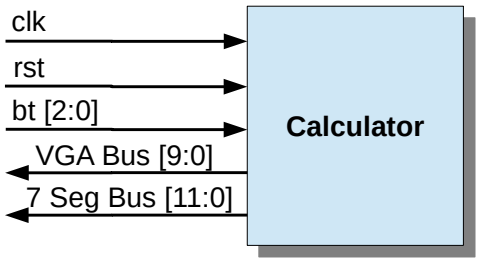

Figure 2: Calculator's symbol

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
Inputs Bus Interface		
bt[2:0]	IN	Buttons.
VGA Bus Interface		
HS	OUT	Horizontal sync for VGA.
VS	OUT	Vertical sync for VGA.
red[2:0]	OUT	VGA Red.
green[2:0]	OUT	VGA Green.
blue[1:0]	OUT	VGA Blue.
7 Seg Bus Interface		
enable[3:0]	OUT	enable the digit.
seg[7:0]	OUT	7 segments and the dot.

Table 1: Interface signals.

4 Program

This system is capable of doing two operands operations. Each operand has 16 bits and the result can have 32 bits. Each operand is chosen with the help of the buttons ”Enter” and ”Rotate”. For each operand 4 bits are chosen at a time. The ”Rotate” button rotates the number between 0 and 15, in hexadecimal 0 to F, and the ”Enter” button chooses these 4 bits. It starts with the 4 least significant bits and in four times one operand is picked. The operand is printed on the 7-Segment display while being chosen. The flowchart for the operation of choosing an operand is shown in Fig 3.

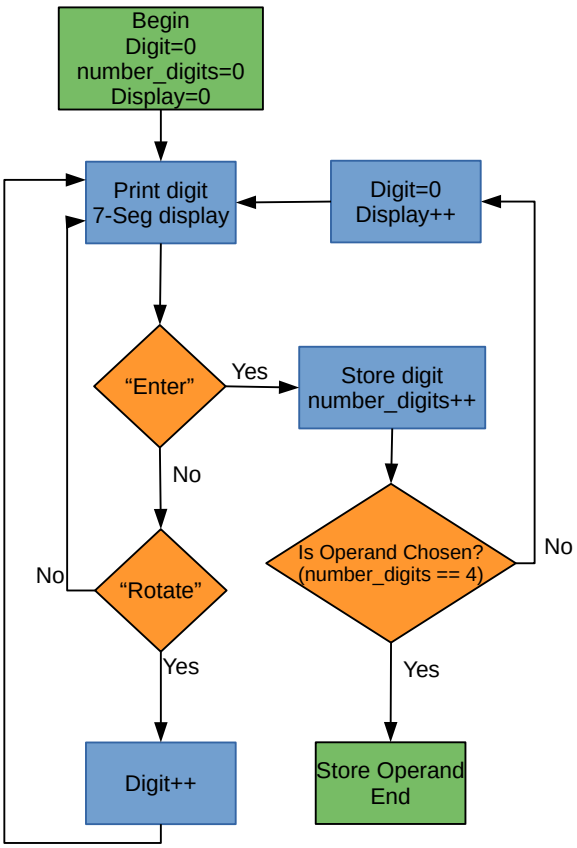

Figure 3: Flowchart of the operand selection

Choosing the operation is like choosing the operand. The ”Rotate” button rotates between the operations, while they are being printed in the 7-Segment display, and the ”Enter” button selects the operation. The flowchart for the operator’s selection is shown in Fig 4.

The order that the operands and operation are chosen is 1^{st} operand, operator and the last 2^{nd} operand. When the operand or operator is select they are printed on VGA display. When all 3 are selected,the result is computed and printed on the VGA display. At this point the result can be used as the 1^{st} operand in the next calculation using the ”Enter” button or can it be discarded using the ”Clear” button. The top-level routine is shown in Fig 5.

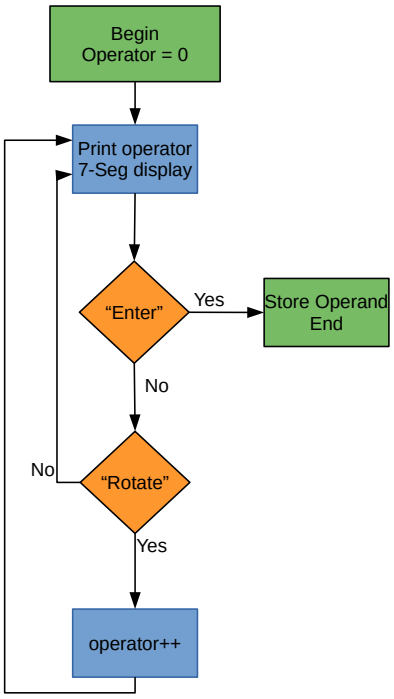

Figure 4: Flowchart of the operator’s selection

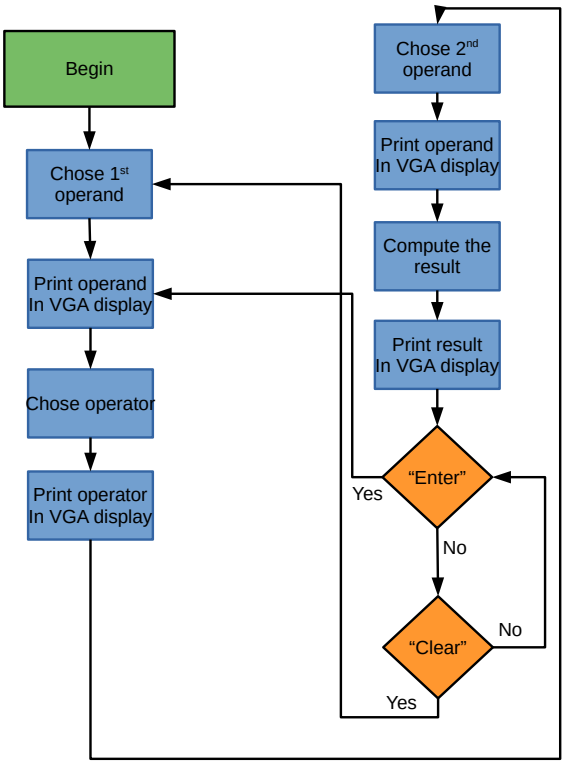

Figure 5: Flowchart of program

The computation of the result depends on the operator. For sums and subtractions the method used is the picoVersat instruction for that purpose. In case of multiplications it is used the multiplier peripheral. In case of exact divisions the method used is an algorithm with shifts and subtractions done in hardware. In case the operator is powers the algorithm used is described in Algorithm 1.

Algorithm 1: Powers

```
Powers(pow, exp)
accumPow:= 1
while exp ≥ 1 do
    if exp least significant bit is 1 then
        accumPow:= accumPow*pow
    end
    exp:= exp<<1
    pow:= pow*pow
end
return accumPow
```

The conversion of a digit to the VGA is done with consecutive splits for 10 sending the position and the remainder to the VGA controller.

5 Memory Map

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

Mnemonic	Address	Read/Write	Read Latency	Description
REGF_BASE	1024	Read+Write	0	Register file peripheral
MULT_BASE	1240	Read+Write	0	Multiplier peripheral
DIV_BASE	1264	Read+Write	0	Divider peripheral
7SEG_BASE	1480	Write only	0	7-Seg display peripheral
BUTT_BASE	1600	Read only	0	Button controller peripheral
VGA_BASE	1640	Write only	0	VGA controller peripheral

Table 2: Memory map base addresses

6 Peripherals

The SoC Calculator has picoVersat as processor/micro-controller and several peripherals as seen before. In this section for each peripheral there is a small description of how is works and information on inputs, outputs and registers. The picoVersat has a address decoder that enables each peripheral.

6.1 Multiplier

The purpose of this peripheral is to do multiplications of 16 bits. There are 2 registers for the two operators and a third register to store the result. The multiplications could be done in software but it is in hardware. This

happens because the fpga's built-in multiplier is used. The symbol of this peripheral can be seen on Fig 17 and a table with descriptions of inputs and ouput in Table 3.

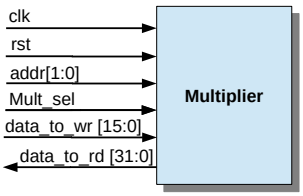

Figure 6: Symbol of multiplier

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
mult_sel	IN	select for the peripheral.
addr[1:0]	IN	select for the register to write/read.
data_to_wr[31:0]	IN	data do write.
data_to_rd[31:0]	OUT	data to read from register.

Table 3: Multiplier interface signals.

The multipliers peripheral block diagram is shown in Fig 7. The register RegMultA and RegMultB are the operands and RegMultC stores the result of multiplication. In Table 4 is shown the memory map for this peripheral.

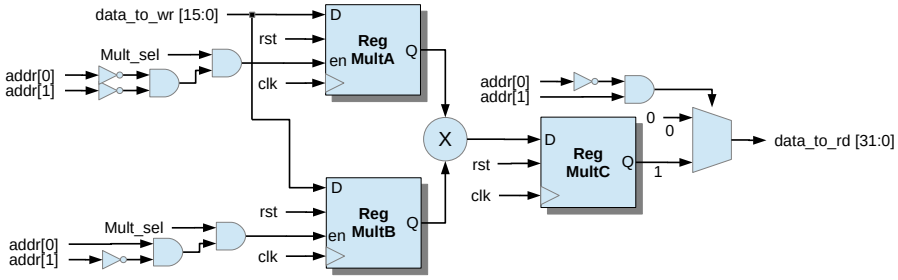

Figure 7: Block Diagram of multiplier

Mnemonic	Address	Read/Write	Read Latency	Description
RegMultA	0	Write only	NA	Register for operand
RegMultB	1	Write only	NA	Register for operand
RegMultC	2	Read	0	Register for result

Table 4: Memory map of Multiplier

6.2 General purpose register

General purpose register is a peripheral that picoVersat already contains. It has 16 registers that can be read and written for the software that runs on the processor.

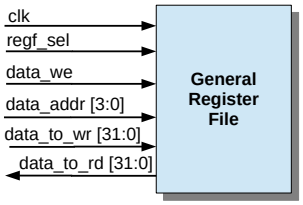

Figure 8: Symbol of general purpose register

6.3 7-Segment display controller

The purpose of this peripheral is to show the operands and the operator when they are being chosen. The operators are 16bit and they are displayed in hexadecimal so one operand will fill the whole 7-segment display. The fpga just has 7 outputs for this display, so in each clock cycle just one of the digits in the display will be illuminated. This will not be visible to human eye because clock is too high. The signal that chooses which digit will be activated is the enable, 1 bit for each digit. The symbol for this peripheral is shown in Fig 9. A table with descriptions of inputs and output in Table 5.

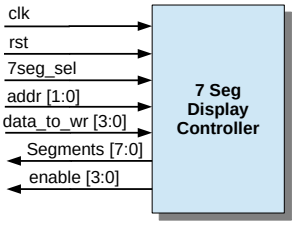

Figure 9: Symbol of 7-Segment display controller

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
addr[1:0]	IN	select for register to write.
7seg_sel	IN	Peripheral select.
data_to_wr[31:0]	IN	data do write.
segments[7:0]	OUT	signal to the 7segments and the dot of 1 digit.
enable[3:0]	OUT	digit's enbale.

Table 5: 7-Segment diplay interface signals.

This peripheral has 4 registers that correspond to the 4 digits.The enable is the signal that chooses the digit, so must only have one bit '0' and the remaining bits '1'. Every clock cycle enable does a left circular shift and the register used for output is the one corresponding to that digit. The controller has an encoder that translates a number, 0 to 15, to the bits for each segment in a digit. The 7-seg display has a dot for each digit. This dot is used to show what digit is being chosen. The block diagram is show in Fig 10 and the memory map for this peripheral in Table 6.

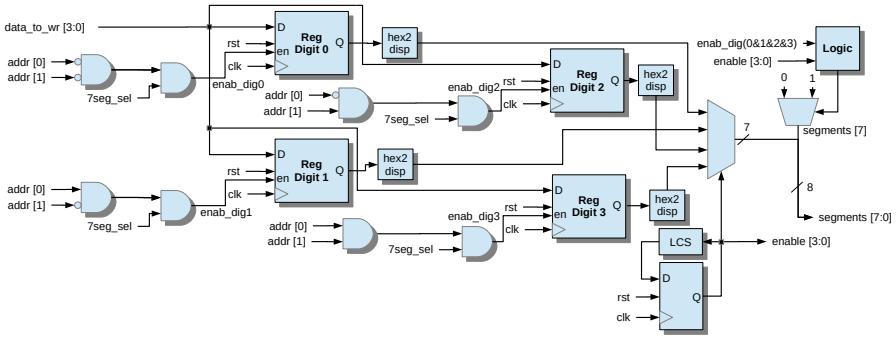

Figure 10: Block Diagram of 7-Segment display controller

Mnemonic	Address	Read/Write	Read Latency	Description
RegDigit0	0	Write only	NA	Register for digit
RegDigit1	1	Write only	NA	Register for digit
RegDigit2	2	Write only	NA	Register for digit
RegDigit3	3	Write only	NA	Register for digit

Table 6: Memory map of 7-Segment display controller

6.4 Button Controller

The buttons are used to select the operand's digits and the operations. The button controller symbol is shown in Fig 11 and a table with the descriptions of inputs and outputs in Table 7.

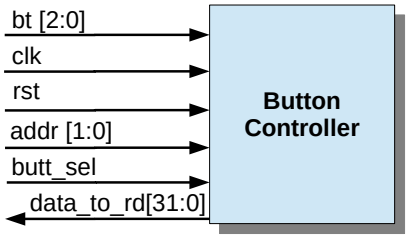

Figure 11: Symbol of button controller

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
butt_sel	IN	Peripheral's enable.
addr[1:0]	IN	Select register to read.
bt[2:0]	IN	signal from 3 buttons.
data_to_rd[31:0]	OUT	data do read.

Table 7: Button controller interface signals.

This peripheral receives the signal of the buttons and stores the state on a register. Each button has a register so in total there are 3 registers. When the buttons are toggled they cause a unpredictable bounce in the signal. This is not desired because it affects how the system works. So the button signals are connected to a debouncer to suppress this. The block diagram of this peripheral is shown in Fig 12 and the memory map in Table 8.

The debouncer consists of a register, preseted with a number high. The signal from the button is the one that presets the register. In each clock cycle the register decreases it's stored value by one and when it reaches 0 the output signal of the debouncer is '1'.

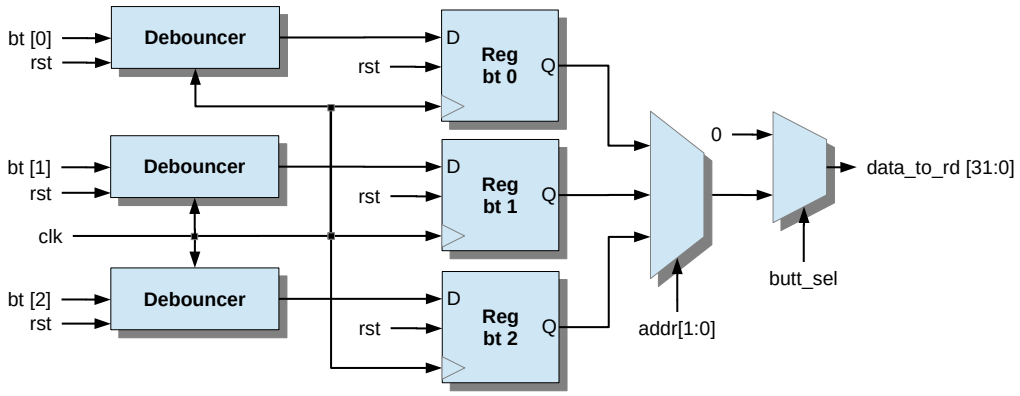

Figure 12: Block Diagram of button controller

Mnemonic	Address	Read/Write	Read Latency	Description
RegBT0	0	Read only	0	Register for button
RegBT1	1	Read only	0	Register for button
RegBT2	2	Red only	0	Register for button

Table 8: Memory map of button controller

6.5 Divider

The purpose of this peripheral is to do divisions. This peripheral compute the quotient and the remain. There are 3 inputs register one for dividend, other for divisor and the last one is the start register. When a '1' is write to start register the divider start the computation. There are 3 output registers, one for quotient, other for the remain and the last one is the done register. When the computation is over the done register will have '1' stored. This is a sub shift divider, that computer the division in several clock cycles using subtractions and shifts. The symbol for this peripheral is shown in Fig 13. The inputs and outputs description can be consulted in Table 9.

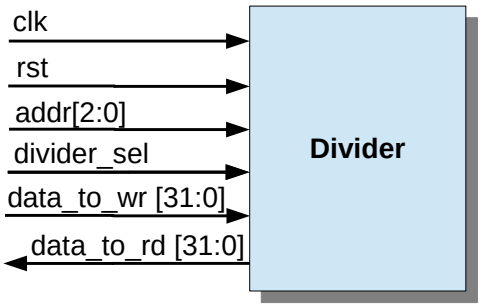

Figure 13: Symbol of divider

The memory map of this peripheral can be consulted in Table 10 and block diagram in Fig 14.

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
divider_sel	IN	enable for this peripheral.
addr[2:0]	IN	address of register in this peripheral.
data_to_wr[31:0]	IN	data to write.
data_to_rd[31:0]	OUT	data to read.

Table 9: Divider interface signals.

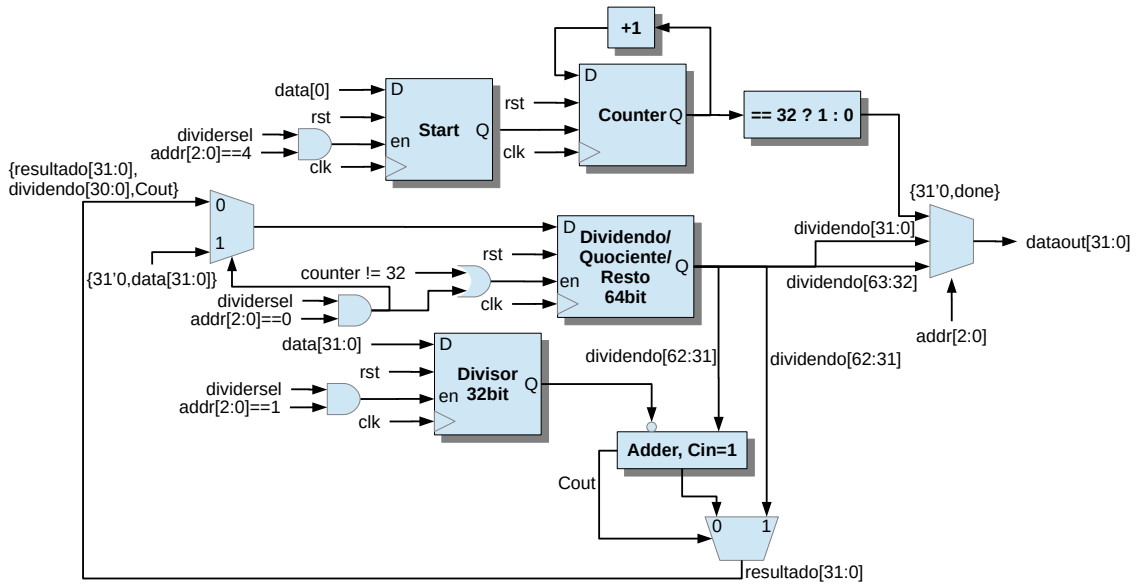

Figure 14: Block Diagram of VGA controller

Mnemonic	Address	Read/Write	Read Latency	Description
RegDiv0	0	Read only	0	Register for dividend
RegDiv1	0	Read only	0	Register for divisor
RegDiv2	0	Write only	0	Register for remainder
RegDiv3	0	Write only	0	Register for quotient
RegDiv4	0	Read only	0	Register for start
RegDiv5	0	Write only	0	Register for done

Table 10: Memory map of divider

6.6 VGA Controller

VGA controller is the peripheral that controls the signals entering VGA display. This peripheral generates a horizontal sync signal, vertical sync signal and the red, green and blue signal for each pixel. There is a video memory that contains, for each pixel on the used display area, a '1' if the pixel is too be drawn white and '0' for it to be drawn black. Because this system only uses some space in the display just a little area of display is stored in the memory. The symbol for this peripheral is shown in Fig 15. The inputs and outputs description can be consulted in Table 11.

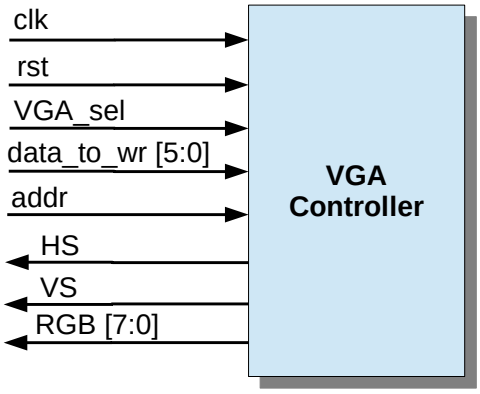

Figure 15: Symbol of VGA controller

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
enable_VGA	IN	enable for this peripheral.
addr	IN	address of register in this peripheral.
data_to_wr[5:0]	IN	data or location to write on the display.
HS	OUT	horizontal sync.
VS	OUT	vertical sync.
red[2:0]	OUT	red for vga
green[2:0]	OUT	green for vga
blue[1:0]	OUT	blue for vga

Table 11: VGA controller interface signals.

The block diagram is shown in Fig 16.

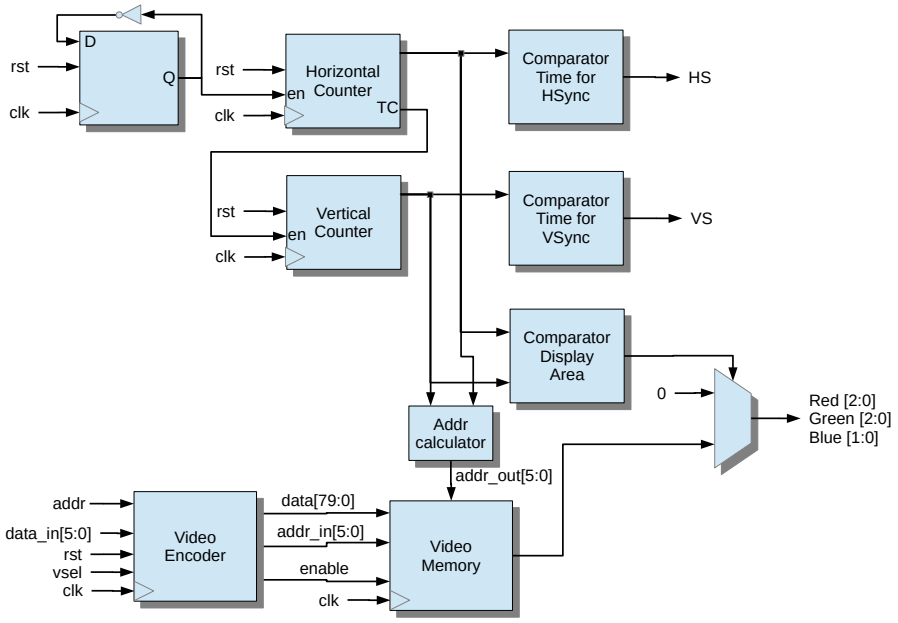

Figure 16: Block Diagram of VGA controller

6.6.1 Video Memory

This sub module is a simple memory with 27 blocks each for a single number/symbol. It receives an output address, so it can output the desired block of pixels to then be sent to the display, and the enable signal, input address and an input block of pixels that are sent from the video encoder sub module to save the number/symbol in the address received if only the enable is active. It currently has 27 blocks which is a bit more than needed but it can be used in the future to allow for bigger numbers, negative numbers or more complex calculations with more numbers at the same time. It can also be easily expanded if the number of blocks is not enough for the desired calculations.

6.6.2 Video Encoder

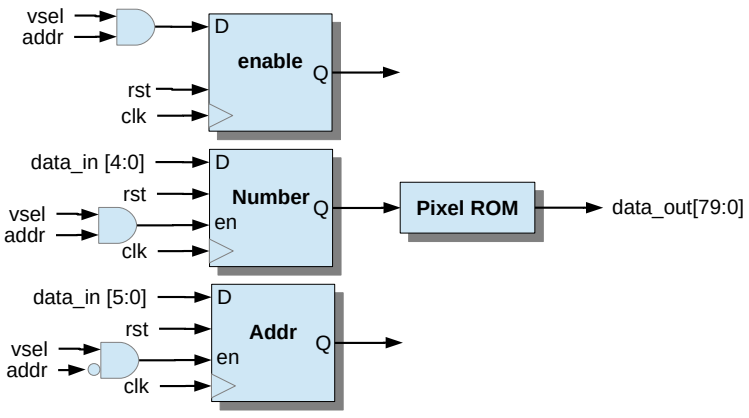

Figure 17: Block diagram of the Video Encoder

Name	Direction	Description
clk	IN	clock signal.
rst	IN	reset signal.
vsel	IN	peripherall signal.
data_in[5:0]	IN	character to convert to pixels.
addr	IN	address of register inside peripheral.
data[79:0]	OUT	converted pixels.
addr_in[5:0]	OUT	positon of the digit.
enable	OUT	enable of the memory.

Table 12: Video encoder interface signals.

This sub module is responsible for receiving a position on the display are and the number/operation symbol to be represented on the position received previously. After receiving the number/operation symbol it is translated into the pixel format and then sent, with the corresponding desired position and enable signal to the video memory so it can be saved there. The pixel format is a block of 8x10 pixels/bits and each bit corresponds to a lit up pixel or not.

Mnemonic	Address	Read/Write	Read Latency	Description
RegVGA0	0	Write only	0	Register for position
RegVGA1	1	Write only	0	Register for character

Table 13: Memory map of video encoder

6.6.3 Pixel ROM

The pixelrom sub module receives the desired number/symbol and converts it into the pixel format. Since there are only 10 digits and 8 symbols it was implemented using a ROM.

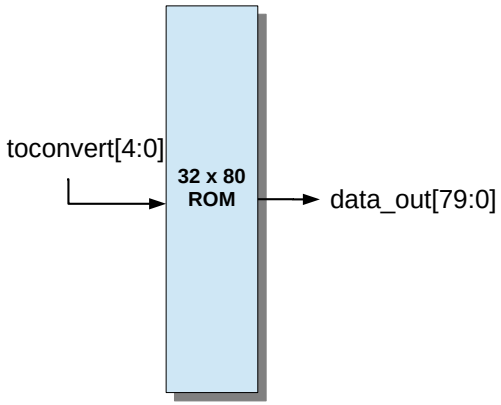

Figure 18: Block Diagram of Pixel ROM

Name	Direction	Description
toconvert[4:0]	IN	character to convert to pixels.
data_out[79:0]	OUT	pixels for one digit.

Table 14: Pixel ROM interface signals.

7 Implementation Results

The implementation was done on Spartan-3 XC3S100E FPGA. This FPGA has 1920 LUTs with 4 input and 2 block rams. The system occupies 81% of the LUTs and 50% of the block ram.

It was used a global clock of 25 MHz because this is the frequency that VGA timings are working with. A higher clock isn't need for anything else so it was defined as the global clock.

The system works as intended. One problem is the stabilization of VGA image. The image isn't stable and the pixels keep going back and forth. This happens because the clock source is not good. Using a resistance of 150 Ω in parallel with the circuit helped but the real solution is to have a better clock source.

This video shows the system working: `https://youtu.be/DqXdNN2pLhk`

As explained before there are three states to do a calculation, the first is choose the first operand, the second is to choose the operator a the last state is to choose the second operand. On the first and third state the second button rotates the digit of the 7seg and the third button locks that digit, when the 4 digits are locked the operand is chosen. In the second state the second button rotates through the operator (0-sum,1-subtraction,2-multiplication,3-division,4-remain,5-power) and the third button chooses the operator.

After the conclusion of a calculation a new one will begin and a new first operand must be chosen. If the third button is pressed the user will need to select the operand using the previous described method, however if the user wishes to use the result of the current calculation as the first operand of the next one the first button needs to be pressed, which also skips the first state.

8 Conclusions

One of the difficulties found is the memory space of the FPGA used. It has 4 memories of 16K bits and that wasn't enough to use a compiler. So all the code was written in assembly. If the compiler was an option the software development for the VGA would be a lot easier and would have many more features, like 1 operand calculations, history of previous calculations, support for negative values and instead of using integers it could use floating point or fixed point.

This system occupies a lot of LUTs. This happens because the video memory is not being mapped into a BRAM. The VGA controller alone needs 20% of all LUTs to be implemented. The video memory is not being mapped into a BRAM because of the word size being to big. The largest word in memory organization for this FPGA is 72 bits (Table 3 in `https://www.xilinx.com/support/documentation/application_notes/xapp463.pdf`) and the word size of video memory is 80 bits (number of bits/pixels needed to represent a digit in the display). The last 2 BRAMs could be used in parallel each storing 40 bits but the implementation doesn't use them. This could be due to timing constrains.

Overall this system/project was considered a success. We proved that picoVersat can handle calculations and VGA output at same time.