Instituto Superior Técnico

Computer Electronics

MEEC 2018-2019

Project Report

PS/2 Calculator

Students:
João Pedro Cardoso, 84096
Pedro Ferreira, 84166
Faculty:
José Teixeira de Sousa

Contents

1 Introduction

The objective of this work is to implement what was proposed. The project is a Calculator which inputs come from a PS/2 QWERTY PT layout Keyboard. The Project proposed was met successfully with some caveats in it's implementation which will be mentioned in another section.

The source code can be found on Gitlab[1].

2 Project Diagram

Like it was explored in the project proposal, the project symbol was overall not changed, shown in figure 1.

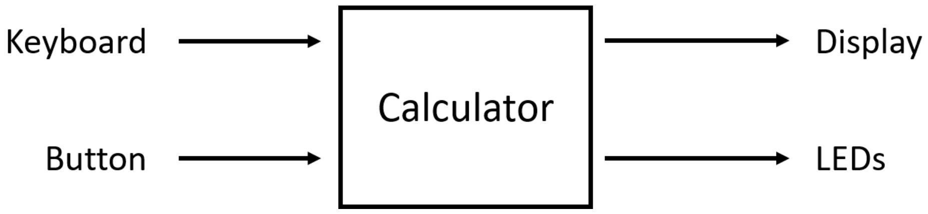

Figure 1: Project Symbol

However, the way the system works on the inside has changed due to difficulties in the implementation of the PS/2 interface and several software bugs that took more time to find than predicted. So, its shown in figure 2, the block diagram of the project.

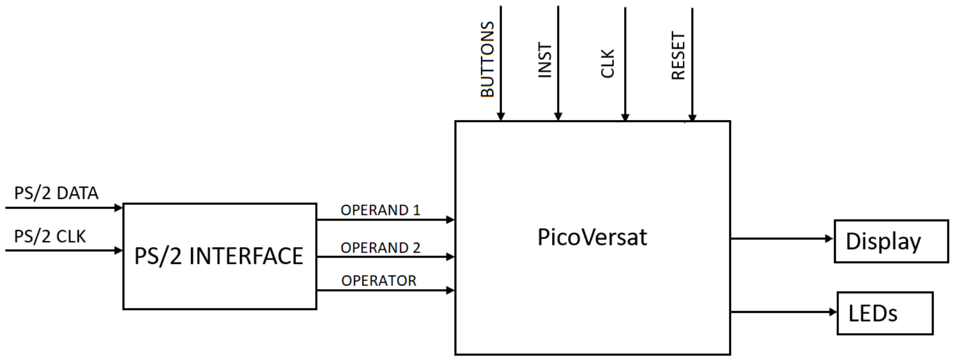

Figure 2: Project Diagram

[1]https://gitlab.com/jcardoso13/ecomp-project

3 Interface Signals

The interface signal of the project are described in table 1. From the 8 LED's, only 1 was used for the overall functionality of the calculator, the rest was used for testing purposes.

Name	**Direction**	#**bits**	**Description**
LED	OUT	8	Light up the 8 FPGA Board LED's
7-Display	OUT	16	FPGA Board 7 Segment Display
CLK	IN	1	Clock signal
RST	IN	1	Reset Signal
PS/2 Data	IN	1	Data signal from the PS/2 Port
PS/2 Clk	IN	1	PS/2 Port Clock
Btn	IN	2	Button from FPGA board signal

Table 1: Project Interface Signals

4 Peripherals

This calculator will use four peripherals. The PS/2 keyboard and one button as inputs while the LEDs and the seven segments display will be used as outputs.

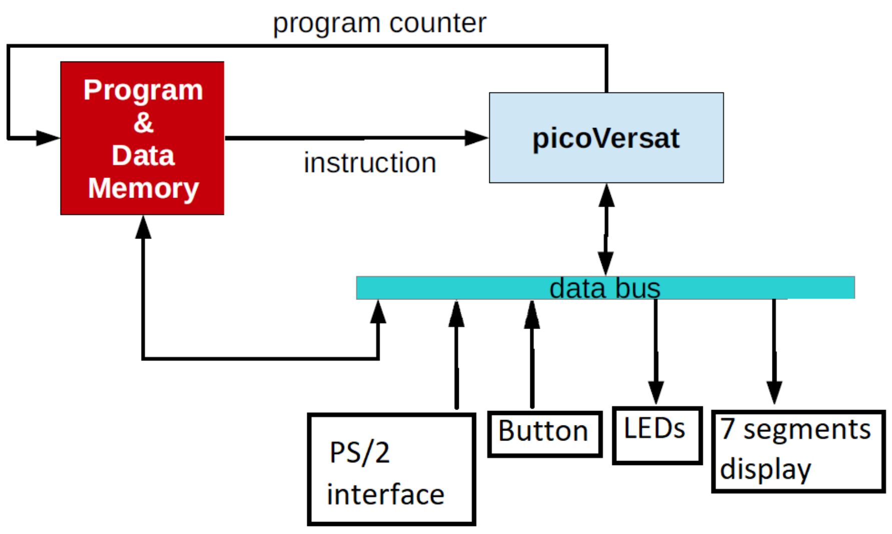

Figure 3: Peripherals

The PS/2 keyboard is the input method where the user can chose the operands and the operations to be done.

The LEDs and the seven segment display will be used as outputs. The final result of the operation will be displayed while the lower bit LED will be used to know when picoversat is reading from RAM (Program).

The buttons are used to reset the system, to show the upper 16 bits of the 32 bit result and to jump to the program.

5 PS/2 Interface

The PS/2 Port is a 6 pin mini-Din used for keyboards and mice. Although it's becoming more and more obsolete, the PS/2 port is the one present in the FPGA Board that will be used to communicate with the Keyboard for inputs for the calculator. One of the changes made compared to the proposal is the PS/2 interface. The proposed implementation didn't work as intended, so alterations were made and more functionality was built in.

5.1 PS/2 Protocol

The PS/2 Port has 2 signals, PS/2 clock and PS/2 Data. The clock is high when its idle, that is, no key is being pressed. when there is a need to send a key that was pressed, the clock goes down (start) and then starts generating pulses.

The Data is 8 bits long and its transferred alongside a start, parity and end bit. More, when the key is pressed, a code is sent and when its released the code F0 is sent alongside the code for the original key.

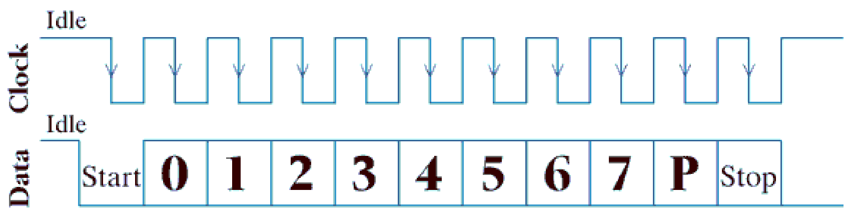

Figure 4: PS/2 Data Protocol

5.2 Code Receiver

The clock of the PS/2 protocol is in the kHz range while picoversat uses a 50Mhz clock. So to capture the data being received,a 3 state machine was devised using picoversat's clk. When the PS/2 clock goes from 1 (idle) to 0 ,It jumps to state Receive and it's ready to receive a code. After receiving it fully, it jumps to the final state that warns the Code Translator that it finished receiving 1 code.

5.3 Code Translator

The Code Translator compares the code received to a database of values found in table 2. The output is the desired translated value.

PS/2 Code	Corresponding Value
0x70	0
0x69	1
0x72	2
0x7A	3
0x6B	4
0x73	5
0x74	6
0x6C	7
0x75	8
0x7D	9
0x5A	Enter

PS/2 Code	Key	Instruction
0x15/0x79	Q +	ADD
0x1D/0x7B	W -	SUB
0x24/0x36	E &	AND
0x2D	R	XOR
0x2C	T	NOT
0x35	Y	OR
0x3C	U	NOR
0x43	I	NAND
0x44	O	XNOR
0x4D/0x3D	P /	DIV & MOD
0x22/0x7C	x *	MULT

Table 2: PS/2 Instruction Codes

5.4 Digit Accumulator

Due to bugs in the software and to meet the deadline, the Digit accumulator was changed from Software to Hardware. The Digit Accumulator discards values coming from the translator until the F0 code is received, then it stores the next value. This is to stop the interface from storing the same value multiple times. It's able to store 4 different values then it multiplies them by 1000,100 or 10 to compile a 4 decimal digit number (13 bits). This number is stored and is ready now to be read by picoversat.

The diagram of this block is in figure 4.

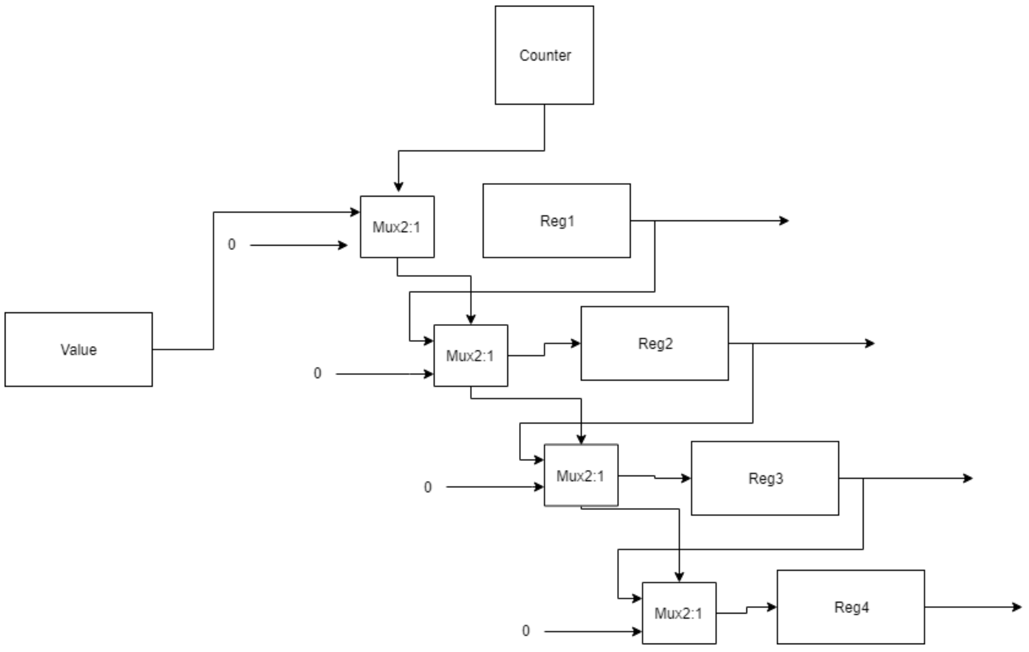

Figure 5: Accumulating Digits to form an Operand

The output of the Interface is characterized the following equation.

$$Output = Reg_4 * 1000 + Reg_3 * 100 + Reg_2 * 10 + Reg_1 \tag{1}$$

It's stored in one of 3 registers that is controlled by a Counter.The first Register is the 1st Operand, the 2nd is the Instruction and the 3rd is the 2nd Operand.
The Instruction values aren't accumulated as Counter only increments if the Value is a digit.

6 Seven Segment Display

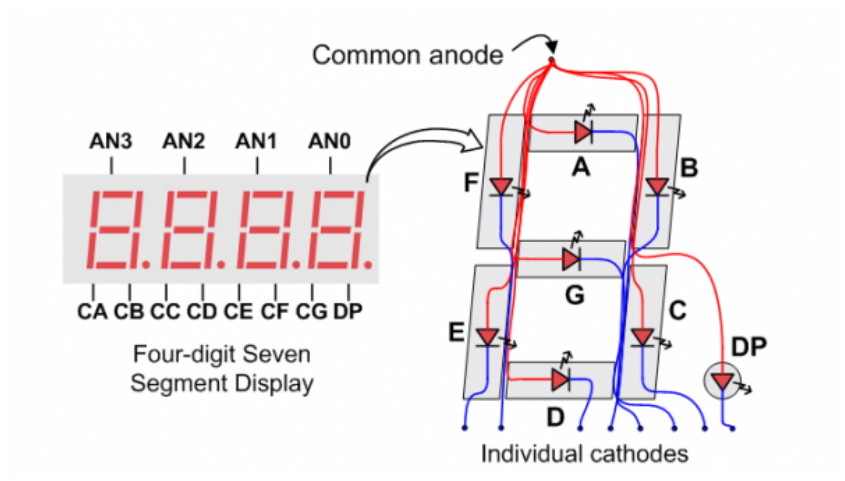

Figure 6: Seven segment display

The seven segment display refresh once every 16 ms, during this time window all the four digits of the display have to be refreshed one by one.
With that in mind, it was implemented a 20 bit counter. This counter is incremented every clock cycle and its two most significant bits will point the digit to be refresh (anode).
The value of the each digit is determined according to the table 3, where "num" is the number to be displayed.

1st digit	num[3:0]
2nd digit	num[7:4]
3rd digit	num[11:8]
4th digit	num[15:12]

Table 3: how to calculate each digit

Then, the result is stored it will be determined which cathodes have to be activated to display that number. The order of the cathodes can be seen in figure 6 and the relation between each digit and the correspondent cathode code is presented in table 4.
The cathode code set which of the segments of the display are on and off. The code form is GFEDCBA and if the value is 1, the corresponding segment is off.

number	cathode code	number	cathode code
0	1000000	8	0000000
1	1111001	9	0010000
2	0100100	A	0001000
3	0110000	B	0000011
4	0011001	C	1000110
5	0010010	D	0100001
6	0000010	E	0000110
7	1111000	F	0001110

Table 4: Cathode code for each digit

7 Memory Map

As far as the memory is concerned, we will store the operands and the operator in different memory addresses as shown in table 5. For picoversat, this memory addresses are read only as the PS/2 Interface controls it's values. Switches and LED's weren't used in software for the final design, but were used as debugging tools.

Address	Description
0x30	Operand 1
0x31	Instruction
0x32	Operand 2
0x26	Switches
0x23	LED's
0x24	Display

Table 5: Memory

8 Correct Functionality of the Program

First of all, to start the program it is needed to press the button 3 of the Basys 2.

Secondly, the user will chose the first operand and press Enter; chose the operation according to table 6 and press Enter again; finally, repeat the procedure of first operand to the second one and the result will be shown on the seven segment display[2].

To check the most significant bits of the result, the user have to press the button 2 of the Basys 2.

When the operation is chosen, the user can check if it is the right one by checking the opcode in the display. This opcode is the code of each operation in the program.

Mod and division share the same key, the program will run both operations and will print the result of division, to see the result of mod, the user have to press the button 2 of the Basys 2.

[2]The result will be shown in hexadecimal.

Operation	Key	Opcode
+	q	F0
-	w	F1
and	e	F2
xor	r	F3
not	t	F4
or	y	F5

Operation	Key	Opcode
nor	u	F6
nand	i	F7
xnor	o	F8
mod	p	F9
÷	p	F9
×	x	FA

Table 6: Instructions

9 FPGA implementation and Resource Utilization

The Resource utilization is given in table 7.

Logic Utilization	Used	Available	Utilization
Slice Flip Flops	304	1920	19%
4 input LUT's	1138	1920	59%
bonded IOB's	28	83	33%
RAMB 16s	1	4	25%
BUFGMUX's	1	24	4%
MULT 18X 18SIOs	3	4	75%

Table 7: Resource Utilization

Regarding the clock of the project, it has a data path delay of 9 levels of logic, leading to a 19.79 *ns* delay. In the light of the above, the required clock is 20.0 *ns*.

10 Software

10.1 Program overview

To get the results, the picoversat micro controller will be the one to perform the operations. While it is not the fastest way to do it, the difference in clock speed between the interface and the controller is substantial and makes performance not a priority. This program (program.va) is used to do all the mathematical and logic operations. It has an input of 3 values, two operands and one opcode, and one output, the result to be printed.

The picoversat will wait for each input different than zero. Moreover, in order to have a difference between zero and no data, each input was incremented by one and when read it is decremented; the opcode (operator) has an "F" as the value of the second byte. A simulation of this operation is shown in figure 7 where R1 is the first operand, R2 is the second operand and R3 is the operation code.

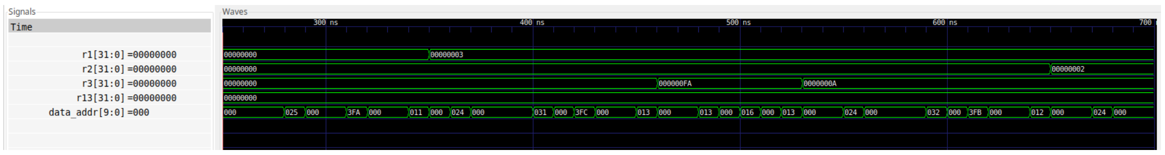

Figure 7: picoversat reading inputs

The values that are load to the registers are already decremented by the function (beqi) that check if the input is 0 or a valid number.

After reading the inputs, the program will run the operation, the simulation in figure 7 is running a multiplication (table 6), and will store the result on R13 as shown in figure 8

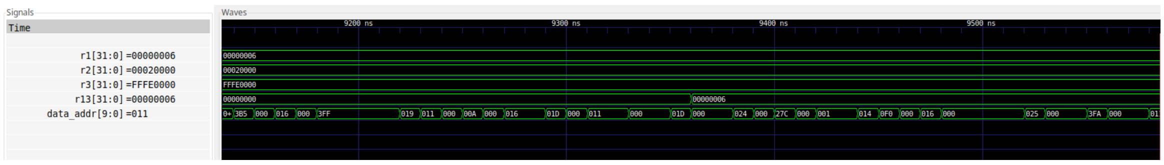

Figure 8: PicoVersat running multiplication

As seen in figure 8, after the result is ready it will be sent to the display (data_addr = 24).

10.2 Operations

The assembller of the picoversat has some of the operations (addition, subtraction, xor, and) that are trivial to implement, the other ones are made using the existing ones.

10.2.1 Not

To implement a logic not with the gates given, a xor with a logic '1' was used. In our case '0xFFFFFFFF'.

10.2.2 Or

The or gate was done with Ands and Xors resulting in the following logic equation

$$OR = \overline{\overline{X} \cdot \overline{Y}} \tag{2}$$

where the Nots are done as explained above.

10.2.3 Nor, Nand and Xnor

These operation were done as Or, And and Xor but in the end it was implemented a Not gate.

10.2.4 Division and mod

The division is done turning to the sub shift algorithm, where the result of the division is on the 16 less significant bit and the result of the mod in the most significant bits. The code of this algorithm is shown in Annex 1.

10.2.5 Multiplication

The implement the multiplication was used the algorithm Radix 2, using the operations addition, subtraction and shift. The code of this algorithm is shown in Annex 2.

10.3 Changes to the proposal

As explained, instead of receiving key by key from the keyboard, the program will receive the final operand and the final opcode.

11 Annex 1 - Division Code

```
#divide two operands
#save the result on R13
#jump back to the beginning

#R1 - 1st operand and where the result will be stores while the result is not ready
#R2 - 2nd operand
#R3 - aux register to store values that may or not be used later
#R6 - size of the operands (always 16)
#R8 - 0x100000...00 (aux value)
#R9 - last bit that will choose the next step (Pos or Neg)

#init all values
Init_D  ldi 0x0000000F
        ldi 0x0000000F
        wrw R6
        nop
        ldi 0xF8000000
        nop
        wrw R8
        shft -1
        shft -1
        shft -1
        shft -1
        wrw R8
        ldi 0x00000001
        wrw R7
#shift R2 to the right place
Shift_d rdw R6
        rdw R6
        beqi Div_i
        wrw R6
        rdw R2
        shft -1
        wrw R2
        ldi 0
        beqi Shift_d
        nop
        nop
Div_i   rdw R6
        ldi 0x0000000F
        wrw R6
Div     rdw R6
        rdw R6
        beqi Done_D
        nop
        nop
        wrw R6
        rdw R1
        wrw R1
        sub R2
        wrw R3
        and R8
#check if R1 is bigger than R2 and store it in R9
        wrw R9
        rdw R9
        beqi Pos
        nop
        nop
#R1 is not bigger than R2
#just shift
        ldi 0xFFFFFFFF
        wrw R9
        rdw R1
        shft -1
        wrw R1
        ldi 0
        beqi Div
        nop
        nop
#R1 is bigger than R2 so subtract R2 and then shift
Pos     rdw R3
        ldi 0xFFFFFFFF
        wrw R9
        rdw R3
        shft -1
        wrw R1
        add R7
        wrw R1
        ldi 0
        beqi Div
        nop
        nop
#print the value and jump to the beginning
Done_D  rdw R1
        rdw R1
        nop
        shft -1
        nop
        wrw R13
        nop
        nop
        wrw DISPLAY_BASE
```

```
ldi 0
beqi Start
nop
nop
```

12 Annex 2 - Multiplication Code

```
#multiply two operands
#save the result on R13
#jump back to the beginning

#R1 - 1st operand and where the result will be stores while the result is not ready
#R2 - 2nd operand
#R3 - -R2
#R4 - aux value 0000....011
#R6 - size of the operands (always 16)
#R7 - value of 1 to do subtractions
#R8 - 0x100000...00 (aux value)
#R9 - last bit that will choose the next step (Pos or Neg)
#R11 -



Radx    ldi 0x00000010
        ldi 0x00000010
        wrw R6
        ldi 0x00000000
        wrw R11
        ldi 0x00000001
        wrw R7
        ldi 0xFFFFFFFF
        xor R2
        add R7
        wrw R3
Shift   rdw R6
        rdw R6
        beqi Init
        nop
        nop
        wrw R6
        rdw R2
        shft -1
        wrw R2
        rdw R3
        shft -1
        wrw R3
        ldi 0
        beqi Shift
        nop
        nop
#do the first step that
```

```
Init    ldi 0x00000003
        ldi 0x00000003
        wrw R4
        ldi 0x0000000F
        wrw R6
        rdw R1
        and R7
        beqi M00
        nop
        nop
        rdw R1
        and R7
        bneqi M10
        nop
        nop

Mult    rdw R6
        ldi 0xF4000000
        shft -1
        shft -1
        shft -1
        shft -1
        wrw R9
        and R1
        nop
        beqi jump
        nop
        nop
        ldi 0xF8000000
        shft -1
        shft -1
        shft -1
        shft -1
        wrw R10
        add R1
        wrw R1

#choose next step
jump    rdw R6
        rdw R6
        beqi Done
        nop
        nop
        wrw R6
        rdw R5
        beqi M00
        nop
        nop
        ldi 0x0000001
        wrw R8
        rdw R5
        sub R8
        beqi M01
        nop
        nop
        ldi 0x00000002
        nop
        wrw R8
        rdw R5
        sub R8
        beqi M10
        nop
        nop
        ldi 0x00000003
        wrw R8
        rdw R5
        sub R8
        beqi M00
        nop
        nop

#print the value and jump to the beginning
Done    rdw R1
        rdw R1
        nop
        nop
        wrw R13
        nop
        nop
        wrw DISPLAY_BASE
        ldi 0
        beqi Start
        nop
        nop
#last tow bits are 11 or 00
M00     rdw R1
        ldi 0x00000000
        wrw R5
        rdw R1
        and R4
        wrw R5
        rdw R1
        shft 1
        wrw R1
        ldi 0
        beqi Mult
        nop
        nop
#last bits are 10
M10     rdw R1
        ldi 0x00000000
        wrw R5
        rdw R1
        and R4
        wrw R5
        rdw R1
        add R3
        shft 1
        wrw R1
        ldi 0
        beqi Mult
        nop
        nop
#last bits are 01
M01     rdw R1
        ldi 0x00000000
        wrw R5
        rdw R1
        and R4
        wrw R5
        rdw R1
        add R2
        shft 1
        wrw R1
        ldi 0
        beqi Mult
        nop
        nop
```