DSD Lab1: Calculator

Tommaso Fava & Jean-Baptiste Denoël

September 22, 2023

1 Introduction

This document presents the work done for the L1 activity of the Digital System Design course. The assignment goal was to create a simple calculator, capable of doing both basic arithmetic (ADD, SUB, MUL) and logic (NAND, NOR, SRA) operations between 2 registers, writable as desired.

Vivado 2023.1 was used for every step of the design and the final implementation was done on the Basys 3 development board, which allows interfacing with the user through 16 switches and LEDs, 5 push-buttons and a 4-digit 7-segment display. As showed in the Figure 1, our application only use some of them.

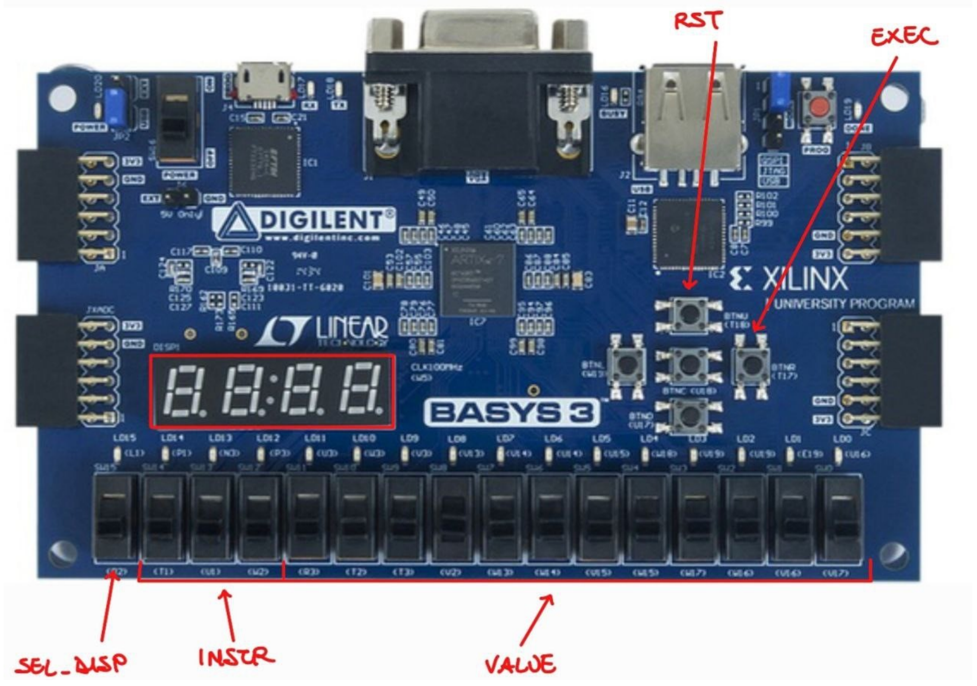


Figure 1: User I/O on the Basys3 development board.

2 Design

Starting by carefully reading the assignment and following the indications provided in class by the professor, the idea was to start defining the signals, the registers and the blocks needed to implement all the operations. Understood this, implementing the control logic and the user interface would simply follow.

2.1 Datapath unit

The Datapath is the entity responsible for the operations and its block diagram (Figure 2) was designed by simply reproducing in blocks and arrows how our application should work. We based our work on the circuit from the Lab0, except that here the register R1 and R2 have different size and instead of 4 possible operations we now have 6, which require a larger MUX (3 bits instead of 1 for the *sel_mux*).

For the registers, the size was chosen in order to allow the storing of the maximum values of the relative intervals, written in the document. In particular, 2^{11} (2047) and 2^{16} (65535) were the highest numbers possible for R1 and R2, but 1-bit more was needed for the sign-magnitude representation.

Moreover, we had to add a last MUX (*sel_disp* of 1 bit) in order to display the register we want on the screen.

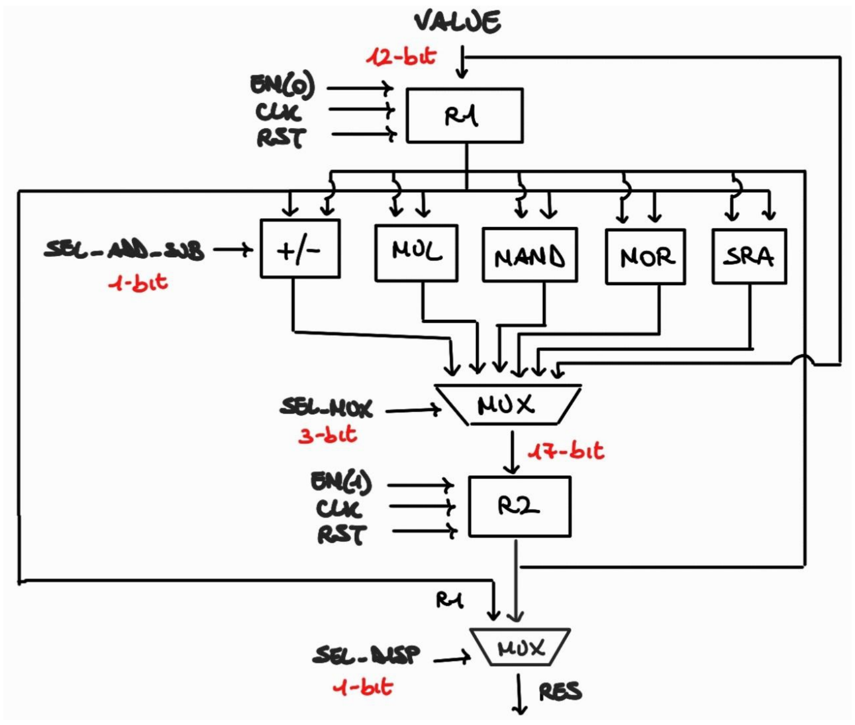


Figure 2: Block diagram of the Datapath entity.

More effort was required to handle the proper numeric formats. Since the indication was to internally work with 2's complement, the input value is immediately converted ($S' = not(S) + 1$) from sign-magnitude and it maintains this format along the entire circuit, before being converted back ($S = not(S' - 1)$) at the output stage.

Subsequently, a conversion between *std_logic_vector* and *signed* is needed by the arithmetic blocks.

Furthermore, there is also the padding of the input value (when loading it into R2) and R1 (either when performing NAND/NOR or showing it to the display) to the size of R2. For the two logic operations, their neutral element is added at the beginning of the vector, while in the other cases 5 bits with the same value of the sign are inserted.

2.2 Control unit

Once designed the Datapath, the Control entity, responsible for the selection and order of the operations to be executed, was defined as a FSM with 1 initial, 1 idle and 8 custom states, as showed in Figure 3.

Like the FSM showed in the class, when the execution is not launched the state is always *wait*, regardless of the instruction selected. Otherwise, the particular operation is chosen by initializing the enables and the selectors for the Datapath.

Here, the only interesting thing is the choice of 'X' (*don't care*) for the bits that are not checked by the Datapath in the selected operation. This can cause a little confusion in the simulation step, but we are convinced that it increases the clarity of the diagram and the code.

2.3 Circuit and Overall unit

Circuit and then Overall (*fpga_basicIO.vhd*) units are needed only to instantiate the components (Datapath, Control but also the display and debouncing manager, not modified in this design) and thus map I/O. They do not deserve special focus, but for the sake of completeness we provide in Figure 4 the schematic generated from Vivado. Moreover, we can see the interaction between the Control and the Datapath units as well as the the signals used for the application.

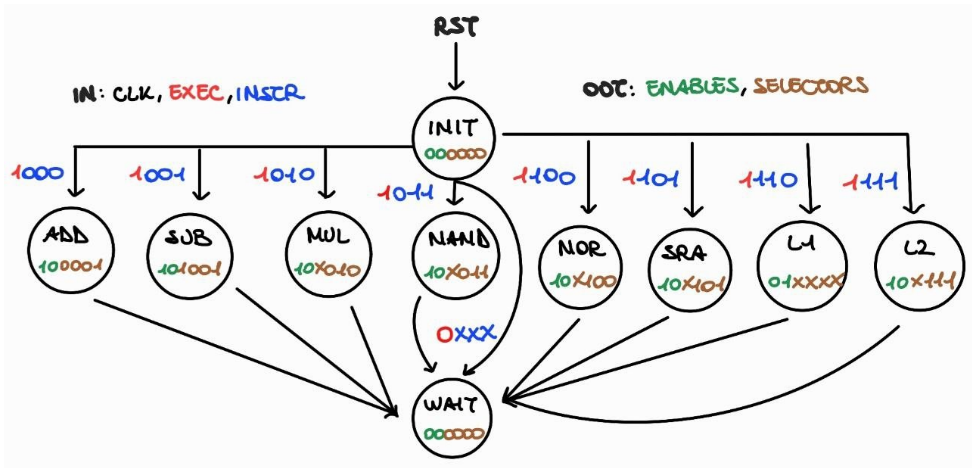


Figure 3: Block diagram of the Control entity.

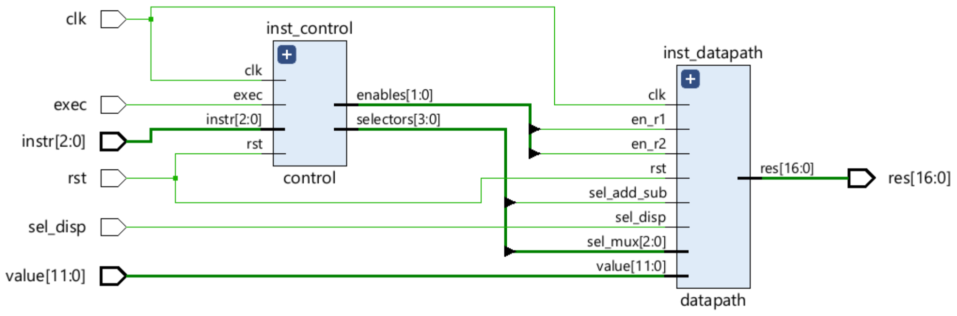
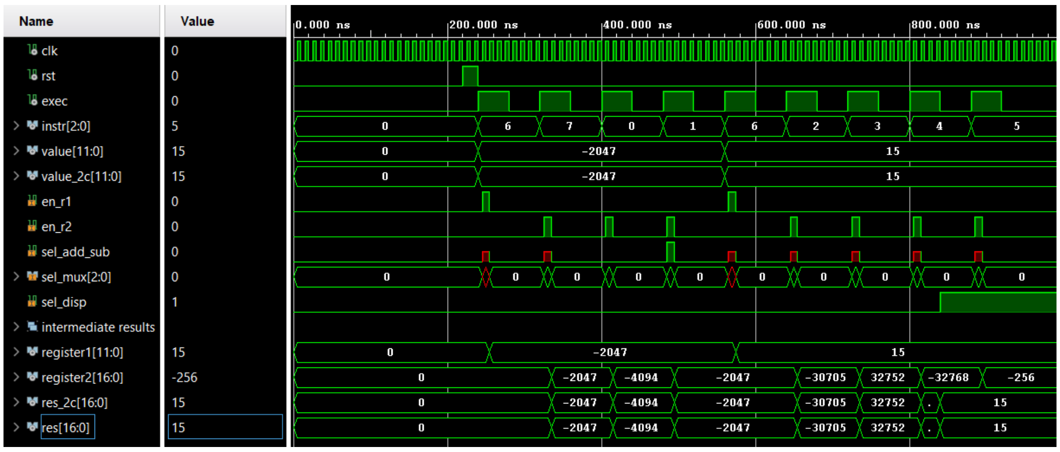


Figure 4: Schematic of the circuit.

3 Simulation

Before proceeding with Synthesis and Implementation, the behaviour simulation of the design was checked in order to validate it.

The testbench file was based on the code made available for the introduction lab. We only modified the ports and the operation to be performed, inputting both negative and positive values to check the correctness of the conversions discussed above. The result of a possible use is shown in Figure 5[1].



Name	Value
clk	0
rst	0
exec	0
instr[2:0]	5
value[11:0]	15
value_2c[11:0]	15
en_r1	0
en_r2	0
sel_add_sub	0
sel_mux[2:0]	0
sel_disp	1
intermediate results	
register1[11:0]	15
register2[16:0]	-256
res_2c[16:0]	15
res[16:0]	15

Figure 5: Waveforms generated by the testbench.

[1]The red signals come from the "X" bits coded in the application.

4 Synthesis and Implementation

In this section we present the achievements of our implementation, starting with the FPGA utilization and following with the timing summary. We also provide an example of the interface with the user.

From the resource utilization results provided in Table 1, we can affirm that our design demands few LUTs, FFs and only 1 DSP (probably for the multiplication). This is normal for such a simple application, however, due to the high interaction with the user, a much more consistent utilization of the input/output pins is required.

Resource	**Utilization**	**Utilization (%)**
LUT	162	0.78
FF	113	0.27
DSP	1	1.11
IO	47	44.34

Table 1: FPGA utilization.

The following Table 2 shows the time summary of the implementation.

A positive time for *Worst Negative Slack* (WNS) means that the signals are arriving slightly later than desired. For the *Worst Hold Slack* (WHS) it means that the design is comfortably meeting hold time requirements. Finally, the *Worst Pulse Width Slack* (WPWS) indicates that the design comfortably meets pulse width requirements. Globally, our implementation respects the time requirements of the device.

Slack	**Time (ns)**
WNS	2.173
WHS	0.145
WPWS	4.500

Table 2: Timing summary.

At the end, to explain better what was already introduced in Figure 1, Table 3 guides you in a possible use of the board.

Instruction			**Inputs**				
Operation	Value	Display	button_up	button_right	sw[15]	sw[14-12]	sw[11-0]
LOAD R1	3	R1	0	1	1	110	000000000011
LOAD R2	-3	R2	0	1	0	111	100000000011
ADD	X	R2	0	1	0	000	X
RST	X	X	1	X	X	X	X

Table 3: Example of the interface between the user and board.

Following this guide, here it's what happens internally:

$$\begin{aligned}
&\text{LOAD R1:} && R1 \longleftarrow 3\\
&\text{LOAD R2:} && R2 \longleftarrow -3\\
&\text{ADD:} && R2 \longleftarrow R2 + R1\\
&\text{RST:} && R2 \longleftarrow 0, R1 \longleftarrow 0
\end{aligned}$$

5 Conclusion

To conclude the report, we must highlight that despite this project was our very first work with VHDL, we were able to design a circuit which correctly performs all the requested operation within the assigned time.

Through this calculator, we understood how to deal with FSM, control signals and datapath entity, creating a scalable code that represents a good basis for possible further works.

Furthermore, we managed to handle the proper conversions between sign-magnitude and 2's complement numeric formats. Since this was one of the biggest challenges of the design, this achievement will certainly speed up our future designs.