1. EXERCISES

1) Modify the SoC to include the fadd, fmul and fdiv instructions, as explained in Section C. Generate the bitstream in Vivado and RVfpgaEL2-Trace, RVfpgaEL2-ViDBo and RVfpgaEL2-Pipeline binaries with Verilator.

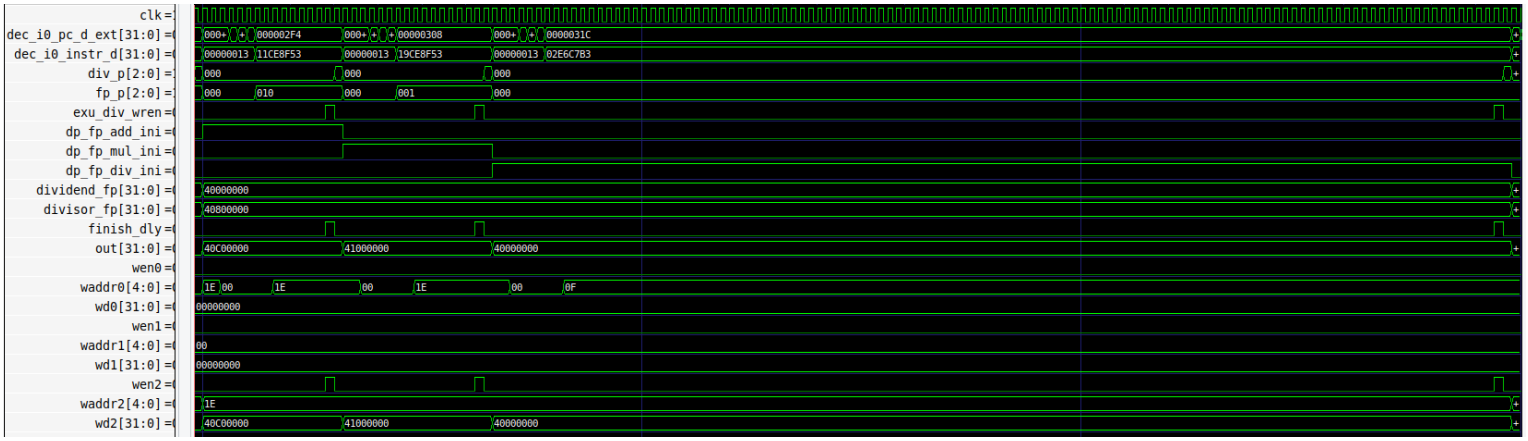
   Solution at: *[RVfpgaBasysPath]/Labs/RVfpgaLabsSolutions/Lab18/Exercise1*

2) Test the program from Figure 1, both in RVfpga-Trace and on the board. Analyse the `fmul` and `fdiv` instructions.

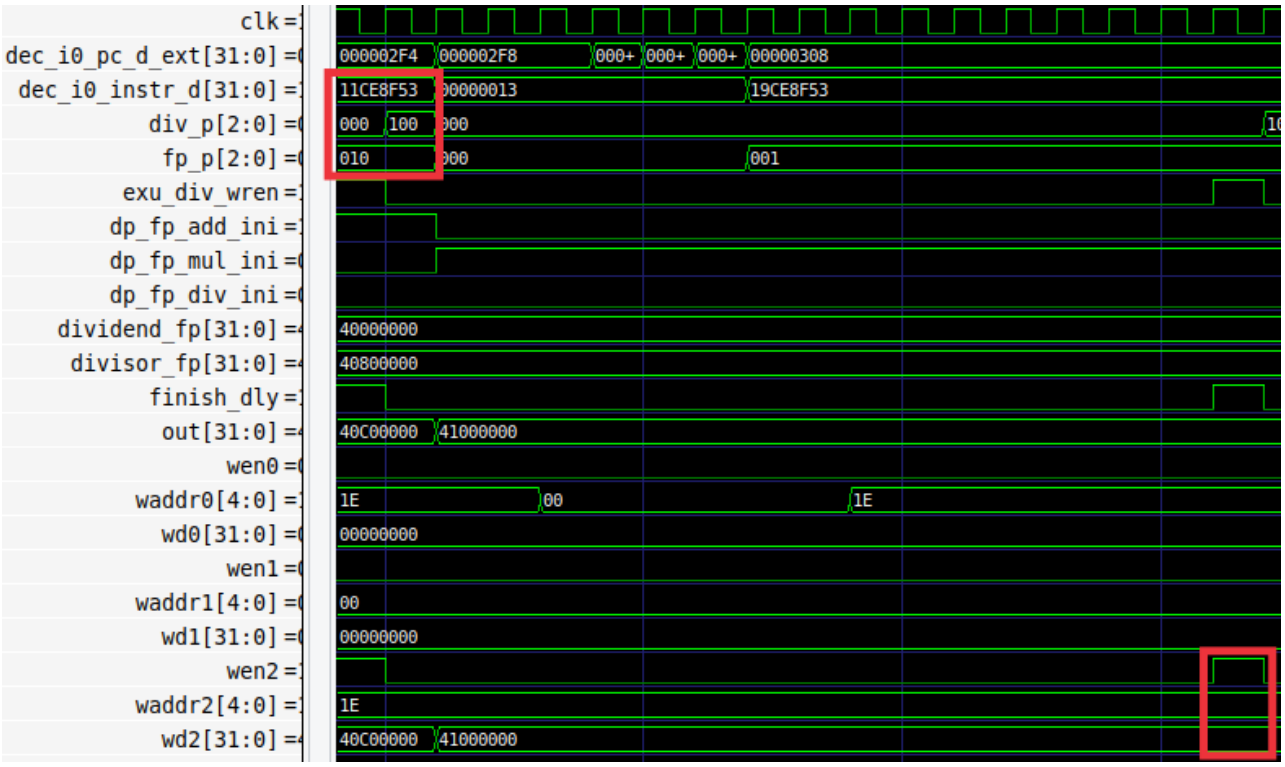
   Catapult project at: *[RVfpgaBasysPath]/Labs/RVfpgaLabsSolutions/Lab18/Exercise2*

RVfpga-Trace:

One whole iteration – fadd fmul fdiv:

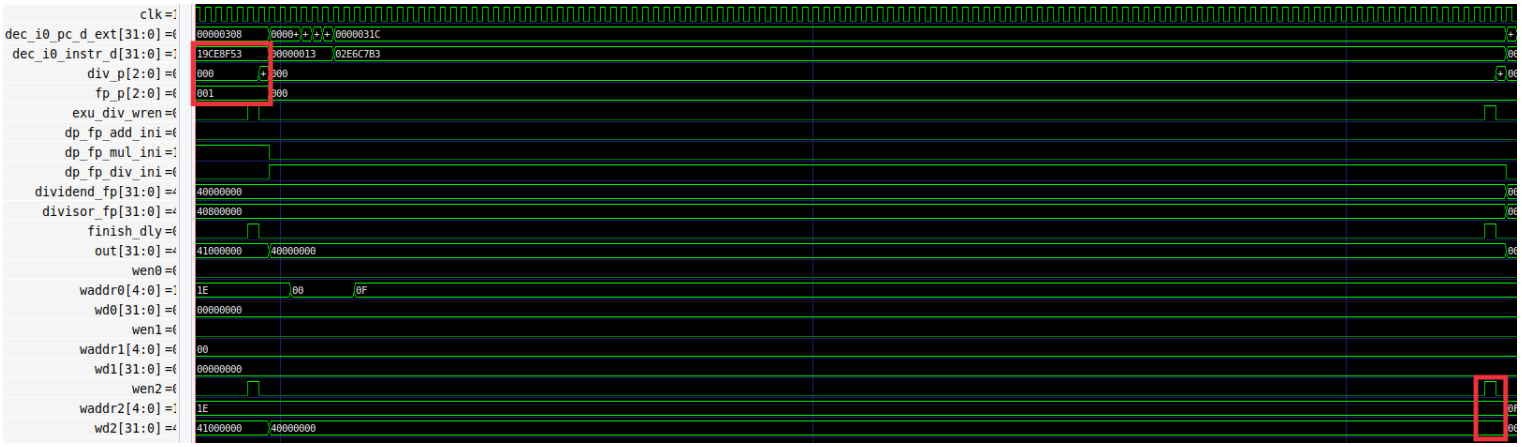


`fmul` instruction:



The fmul is executed correctly and obtains the expected result: 0x41000000 which is 8 (4*2).

`fdiv` instruction:



The fdiv is executed correctly and obtains the expected result: 0x40000000 which is 2 (4/2).

3) Modify the provided program to test other cases and test if the instructions work correctly. For example, test negative numbers, data dependencies with previous/subsequent instructions, etc.

Based in the Test Program used in the lab, we insert two dependent instructions, one before the fadd and another after the fadd.

```
main:

li t3, 0x40800000
li t4, 0x40000000
li t6, 0x800000

li a3, 20
li a4, 4

REPEAT:

INSERT_NOPS_4
add t3, t3, t6
.word 0x01ce8f53 # fadd.s t5, t3, t4
sub t3, t3, t6
```

add: 0x40800000 + 0x00800000 = 0x41000000 (8)
fadd: 0x41000000 (8) + 0x40000000 (2) = 0x41200000 (10)
sub: 0x41200000 - 0x00800000 = 0x41000000 (8)

- The integer add and sub are computed correctly.
- The values are also correctly forwarded.
- The result of the fadd is correct (0x41200000).

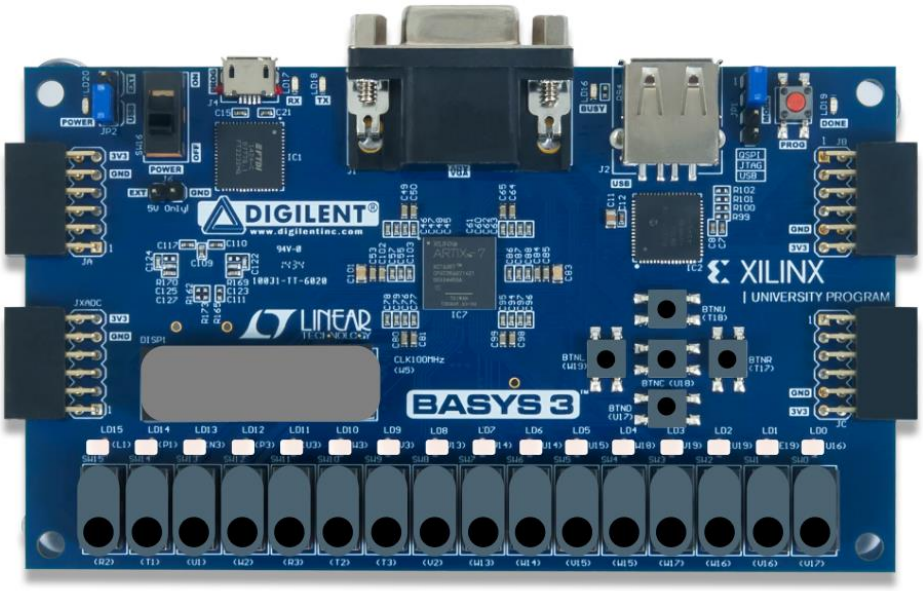
4) Implement the example *DotProduct_C-Lang* provided in the GSG, using the new `fmul` and `fadd` instructions for performing the floating-point computations. Compare the execution of this algorithm when floating-point instructions are emulated vs. when these instructions are implemented in hardware.

Catapult project at: *[RVfpgaBasysPath]/Labs/RVfpgaLabsSolutions/Lab18/Exercise4*

```
                Cycles SwEmul= 1270
Cycles HwImpl= 447
```

Disconnect | Clear UART output

7 SEGMENT DISPLAYS: 0 0 0 0



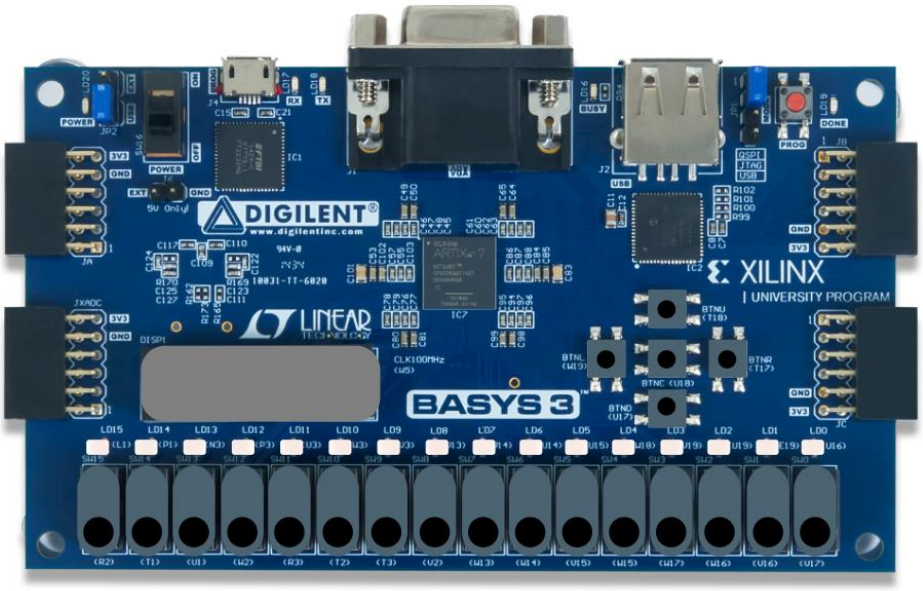
5) Implement the Bisection Method. You can find a lot of information about this root-finding algorithm on the internet, for example, at: https://en.wikipedia.org/wiki/Bisection_method. Compare the execution of this algorithm when floating-point instructions are emulated vs. when these instructions are implemented in hardware.

Catapult project at: *[RVfpgaBasysPath]/Labs/RVfpgaLabsSolutions/Lab18/Exercise5*

```
                Cycles SwEmul= 9069
Cycles HwImpl= 1994
```

Disconnect | Clear UART output

7 SEGMENT DISPLAYS: 0 0 0 0



6) Replace the FP Unit for the following one: https://github.com/openhwgroup/cvfpu. The Final Degree Project "Extensiones de punto flotante para el core SweRV EH1" should be helful, as it performs the same extension on SweRV EH1. You will find the project on the Internet and the sources at: https://github.com/aperea01/TFG-SweRV-EH1-FP.

Solution not provided for this exercise.

7) Add more functionality, such as providing support for: other floating-point formats (such as *double precision*), other floating-point rounding modes, a new register file for the floating-point values (note that floating point instructions that use a Floating Point Register File are described in the RISC-V F extension), your own FP unit implementation, etc.

Solution not provided for this exercise.

8) Add instructions from other RISC-V Extensions that are not available in the SweRV EL2 processor.

Solution not provided for this exercise.

9) Verify the processor including the new instructions. The Final Degree Project “Extensiones de punto flotante para el core SweRV EH1” should be helful, as it performs the same extension on SweRV EH1. You will find the project on the Internet and the sources at: https://github.com/aperea01/TFG-SweRV-EH1-FP.

Solution not provided for this exercise.