Assignment Homework 3

Due date: Check Canvas

Student Version – "Playing With DSPs"

Please team up with a fellow student and start working on the following tasks (100 pts total). In addition, you *must* consider the following:

- Vivado creates a default template including a comment block with:
  - "Company" – insert team name/number
  - "Engineer" – put team member names here
  - "Design Name" – based on task description (*do not leave empty*)
  - "Project Name" – Assignment 3
  - "Description" – based on task description (*do not leave empty*)
- To prepare your assignment for submission through Canvas, you must use (in Vivado): `File > Project > Archive`
- You work as a team. Please alternate between one student creating the testbench, the other student creating the implementation.

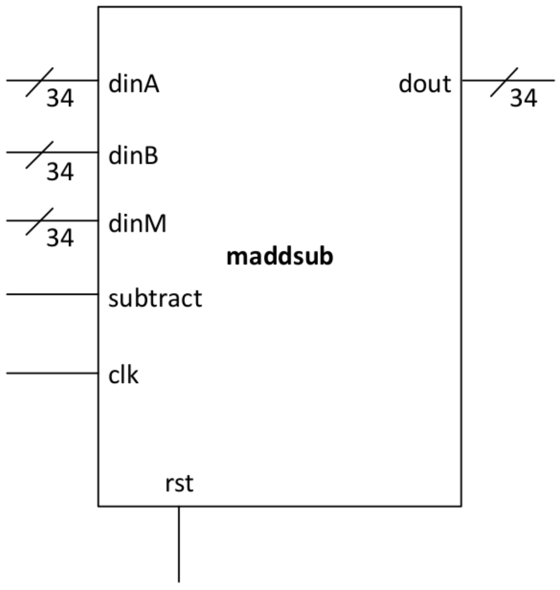
We will start this homework assignment together through Zoom as a lab. Please try to follow along and ask questions if you get stuck somewhere. Your homework assignment includes all tasks mentioned here (even those, that you manage to complete during our joint lab time).

1 Task

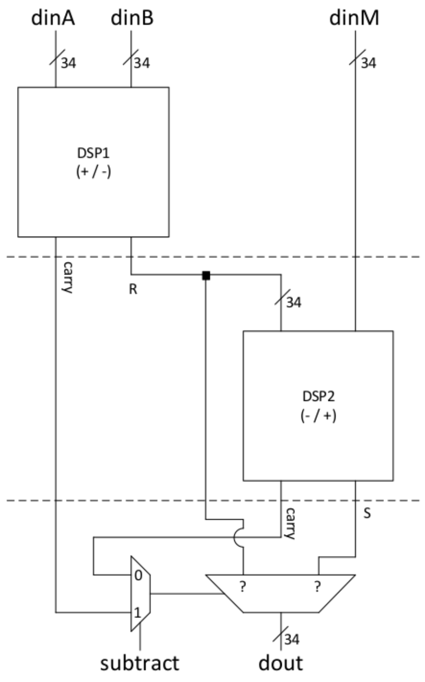
Modular arithmetic operations are very essential for a wide range of applications. To test your knowledge of FPGAs, you decide to make use of its DSPs to implement the following equation: $A \pm B \bmod M$. You grab a sheet of paper and write down the following sketch for an implementation:

1. $A, B \leq M$ (this is a good assumption)
2. If $R = A + B \geq M$, then $S = (A + B) - M$ is the result, else R
3. If $R = A - B \leq 0$, then $S = (A - B) + M$ is the result, else S
4. Only small numbers to start with.

Since DSP slices are a hard block in an FPGA, you familiarize yourself with its internals first[*]. In addition, you limit the data inputs to 34 bit. After a little bit of digging in the data sheet, and studying of the Verilog instantiation template in the HDL Libraries guide[†], you decide following your instructor's recommendation to use the IP Core generator in Vivado. To help you complete this assignment more quickly, you are also advised to use `CONCAT+C` and `C-CONCAT` as the two supported instructions in the Core generator.



(a) Module layout



(b) Basic structure (*incomplete*)

Figure 1: Module/interface description of assignment 3.

To simplify the design process, you quickly partitioned the design into three sections. Now you need to do the following steps:

1. Read the about DSP slices and understand its pipeline registers and input/output paths.
2. Look at the module layout and create the entity description. Note that all inputs are either on the left or lower side while the output is on the right side.
3. Implement the basic structure. Configure the DSP cores in a way that they do not use any pipeline registers (combinatorial logic only).
4. Save the adder which computes the modular addition or subtraction in one clock cycle as design `maddsub_direct.v`.
5. Now, modify the design to use two pipeline stages (split the design into three sections as indicated in the figure). Save the design as `maddsub_pipe.v`.

[*] `https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1`
[†] `https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/DSP48E1`

2 Output

Create two (System-)Verilog modules `maddsub_direct.v` and `maddsub_pipe.v`, as well as two test benches. Explain where your pipeline registers are (inside or outside of the DSPs) and explain your choice. Think about good test vectors and explain why you use the values and what you try to test, e.g., corner cases. Test both modules using these values and verify the correctness. Again, please try to use good habits to write your code and the test benches. This includes creating a nice wave form configuration, with groups for each DSP and multiplexer. In your submission, please include all modules you use (not everything you have written up until this point, just assignment 3 related modules), the test documentation, and ideally, the wave form configuration in the archive.

3 Hints

1. Use synchronous resets if not stated otherwise.
2. Do not worry about input carry bits, as you only have to process 34 bit numbers. This will make it easier getting used to DSPs.
3. You will quickly realize that you need to extend signals to match the port definition of the DSP. In Verilog, you can use code such as:

Listing 1: Concatenation Example

```
wire [48-1:0] extDinA;
wire [48-1:0] extDinB;
assign extDinA = {{14{1'b0}}, dinA};
assign extDinB = {{14{1'b0}}, dinB};
```

4. You specifically need to think about the control signal of the output multiplexer:
   - You need to select the output of DSP1 or DSP2
   - This depends on 3 signals: subtract, carry1, carry2
   - $2^3 = 8$ possible combinations/logic states to consider

```
When subtract=0 AND carry1=0 and carry2=0 then ...
When subtract=0 AND carry1=1 and carry2=0 then ...
When subtract=0 AND carry1=0 and carry2=1 then ...
...
are there two combinations that should never occur?
if so, please assign 'X' instead of 0/1
...
When subtract=1 AND carry1=1 and carry2=1 then ...
```

   - If in doubt please consult the Vivado Synthesis Guide: [*]

[*] https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Vivado-Synthesis