Integrated Systems Architectures

Lab 2: digital arithmetic Assignement

Read the Lab 2 description and address the following points.

1 Digital arithmetic and logic synthesys

1.1 Synthesis strategies

- Download from the *Portale della didattica* the "Floating Point Unit lite" (cvfpu_lite) project. This project is a simplified version of the *cvfpu* project developed by the OpenHW group (`https://github.com/openhwgroup/cvfpu`). It is Floating Point Unit developed in SystemVerilog and is compliant with the IEEE 754-2008 standard:
  `https://en.wikipedia.org/wiki/IEEE_754`.
- Unpack the project. You have sources (`src`) and testbench (`tb`) files. As wou can see you have two topfiles for the testbench: *tb_fpnew_top_rtl.sv* and *tb_fpnew_top_net.sv* as user defined types at the top module interface in the RTL description are translated to standard arrays in the netlist. The text file prj.txt gives you the hierachy order of the files (source and testbench), so it can be useful to derive scripts both for simulation and synthesis.
- Verify the model by testing it with a simple testbench to perform at least 10 different floating point multiplications[(1),(2)].
- Build a table summarizing the results asked in the following points:
  1. Synthesize with `compile`. Find the maximum frequency and the area. Verify the netlist behaviour via simulation.
  2. Repeat the previous step issueing the `optimize_registers` command after `compile`. Find the maximum frequency and the area. Verify the netlist behaviour via simulation.

3. Repeat the previous step issueing only the `compile_ultra` command. Find the maximum frequency and the area. Verify the netlist behaviour via simulation.
4. Force Design Compiler to flatten the hierarchy and to implement the Significands multiplier (Mantissa multiplier in fpnew_fma.sv [(3)]) as a CSA multiplier. Find the maximum frequency and the area with the commands `compile` and `optimize_registers`. Verify the netlist behaviour via simulation.
5. Repeat the previous step by forcing Design Compiler to implement the Significands multiplier as a PPARCH mulitplier. Find the maximum frequency and the area with the commands `compile` and `optimize_registers`. Verify the netlist behaviour via simulation.

- Add as an appendix in your report the full text of the `report_timing` and `report_area` commands. For the CSA and PPARCH architecture add the full text of the `report_resources` command as well.

Note: the maximum frequency corresponds to the minimum period with slack met and equal to zero.

[(1)] you can extend the testbench already provided in cvfpu_lite project. You also have in the project a simple C program (fp_num.c), which prints floating point numbers in exhadecimal in compliance with the IEEE 754-2008 standard.
[(2)] you will see some warnings during the compilation concerning potential conflicts with always_comb and always_latch variables. Extra checking is done later when invoking vsim, so you can neglect this kind of warning.
[(3)] in the hierarchy of the design you find the mantissa multiplier in: *gen_operation_group[0]* $\rightarrow$ *i_opgroup_block* $\rightarrow$ *gen_parallel_slices[0]* $\rightarrow$ *active_format* $\rightarrow$ *i_fmt_slice* $\rightarrow$ *gen_num_lanes[0]* $\rightarrow$ *active_lane* $\rightarrow$ *lane_instance* $\rightarrow$ *i_fma*.

1.2 R8-MBE multiplier implementation

Design in SystemVerilog a Radix-8 Modified Booth Encoder (R8-MBE) based multiplier for unsigned data and use it as the Mantissa multiplier in fpnew_fma.sv. You can refer to Section 2 and Fig. 1 of paper [4] for a detailed explanation of R8-MBE. As pointed out in [4] the main issue of an R8-MBE circuit is the generation of the $3X$ term, which requires an n-bit adder to implement $3X = 2X + X$. Indeed, in [2] it is shown that one can implement the encoder generating $3X$ relying on three main approaches: 1) non-redudant form, where the encoder outputs an array, which contains $3X = 2X + X$; 2) fully reduntant form, where the encoder outputs two arrays, where the first one contains $4X$ or $2X$ or 0 and the second one contains X or 0; 3) partially redundat form, where the encoder outputs an array and some dots, namely the sum $3X = 2X + X$ is split in short partial additions (k-bit each) by cutting the carry propagation; then, the carries are propagated as dots in the adder plane. As it can be observed the non-redundant form exploits the maximum compression capability of the encoder, whereas the fully redundant one aims at avoiding the use of adders in the encoder at the expense of lower compression. Partially redundat forms are a trade-off. **Implement the R8-MBE with a non redundant form.**

The adder plane must rely on a Dadda-like (as late as possibile) tree [3], where compression must be achieved by using (when possible) 5/3 (4/2) compressors, then full adders (3/2 compressors) and half adders. Note that at a certain level l in the tree, a 5/3 compressor in column j receives a bit, named *Tin*, from column $j-1$ (zero in some cases) and compresses 4 bits (x_1, x_2, x_3, x_4) belonging to column j into:

1. one bit, named S and belonging to column j for the next level $(l-1)$;
2. one bit named C belonging to column $j+1$ for the next level $(l-1)$;
3. one bit, named *Tout* to be connected to the *Tin* of the compressor in the next column $(j+1)$.

If the next column $(j+1)$ in current level l does not need a compressor, then *cout* is propagated as a dot for column $j+1$ in the next level $(l-1)$. You can refer to Section 2 and Fig. 4 of the paper [5] for the details on 5/3 (4/2) compressors.

Sign extension bits in the Dadda-like-tree must be simplified in order to reduce the number of compressors as mentioned in Section 2 of paper [4] and explained in [1].

Once the design is finished you have to:

1. Simulate the multiplier first as a stand-alone component and then included in the FPU as the Mantissa multiplier;
2. Synthesize the FPU including your muliplier with `compile_ultra`. Find the maximum frequency and the area. Verify the netlist behaviour via simulation.
3. Add as an appendix in your report the full text of the `report_timing` and `report_area` commands.

References

[1] G.W. Bewick. *Fast multiplication: algorithms and implementation - Appendix A - Sign Extension in Booth Multiplier*. PhD thesis, Stanford, 1994.

[2] M.J. Flynn. Advanced computer arithmetic - lecture 7: Integer multiplication.

[3] M. Martina. About wallace and dadda trees.

[4] B.K. Mohanty and A. Choubey. Efficient Design for Radix-8 Booth Multiplier and its application in Lifting 2-D DWT. *Circuits Systems and Signal Processing*, 2017.

[5] A. Strollo, E. Napoli, D. De Caro, N. Petra, and G. Di Meo. Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers. *IEEE Trans. on Circuits and Systems - I*, 2020.