Summer School on Real-world Crypto and Privacy – Hardware tutorial

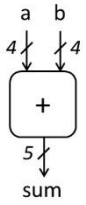
June 20, 2019, Šibenik, Croatia

Nele Mentens (KU Leuven)
Pedro Maat Costa Massolino (Radboud University)

TUTORIAL INSTRUCTIONS

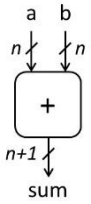
In this tutorial, we will design and simulate a hardware architecture for elliptic curve cryptography. First, we will gradually build the datapath. Next, we will add a memory unit and control logic to implement an elliptic curve point doubling. We simulate the behavior of each new design by following Step 3 of the pre-tutorial instructions. Note that we only do behavioral simulation in this tutorial; for actual hardware implementation, we would need tools for synthesis and physical implementation (as explained in the introductory tutorial presentation). Download the VHDL design files from https://github.com/nmentens/Sibenik2019.

1. Simulate the 4-bit adder architecture



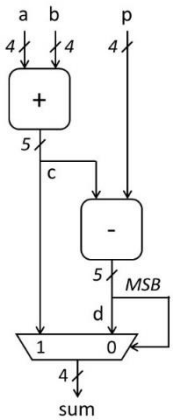
The first design we will simulate, is a 4-bit adder, with *a* and *b* as inputs and *sum* as the output. Open the file and try to understand the VHDL code of both the design module ("add4.vhd") and the testbench ("tb_add4.vhd"). Next, run the testbench and check if the output waveforms correspond to the expected behavior.

2. Simulate the n-bit adder architecture



Try to understand the VHDL code of the n-bit adder ("addn.vhd") and testbench ("tb_addn.vhd"). Notice that n gets the value 8 in the testbench. Run the testbench and check the outputs.

3. Simulate the 4-bit modular adder architecture



The 4-bit modular adder computes the addition of the inputs (*a* and *b*), and subtracts the modulus (*p*) from the intermediate result (*c*), with *d* as a result. The final result of the modular addition is either *c* or *d*, depending on the sign of *d*. The sign is determined by the MSB, i.e. the most-significant bit. The multiplexer drives the output in the following way:

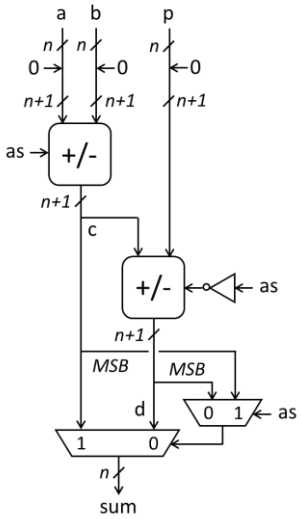
- If the MSB of *d* is 0, $d \geq 0$ and *sum* = *d;*
- If the MSB of *d* is 1, $d < 0$ and *sum* = *c*.

Use the design module ("modadd4.vhd") and the testbench ("tb_modadd4.vhd") to understand and simulate the design.

4. EXERCISE: Design and simulate an n-bit modular adder architecture

Design an n-bit modular adder architecture starting from the design module template ("modaddn.vhd"). Use the testbench ("tb_modaddn.vhd") to simulate the design with n = 8. Change the testbench such that it can handle 128-bit input and output values. Use hexadecimal representation for the input values. Example: `a <= x"AB";` is the same as `a <= "10101011";`

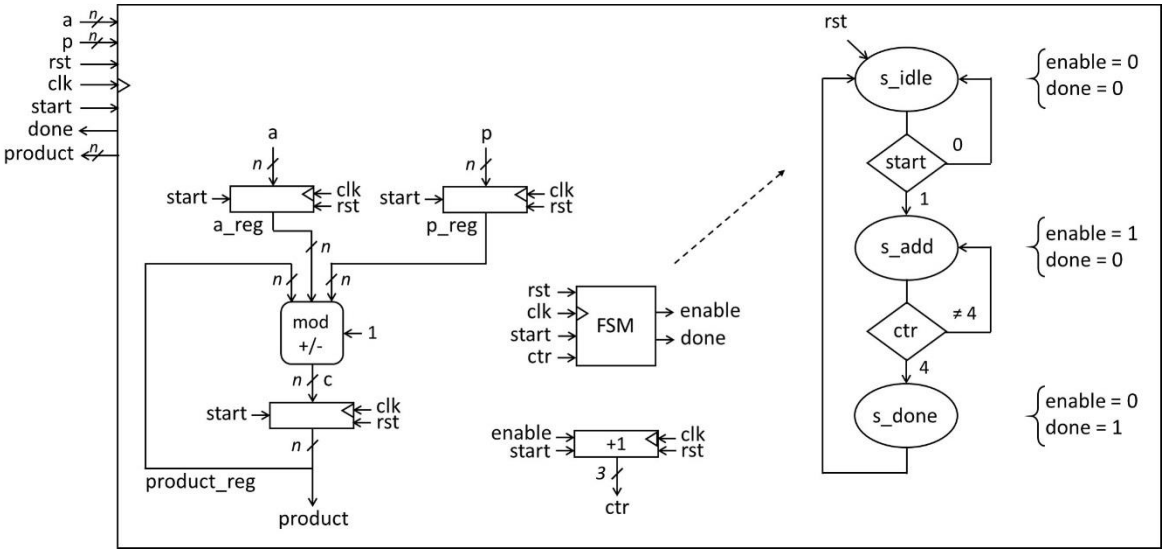
5. Simulate the n-bit modular adder/subtracter architecture



The modular adder/subtracter performs either a modular addition (when the $\overline{\text{add}}$/subtract signal, *as*, is 0) or a modular subtraction (when *as* is 1) on the inputs *a* and *b* with modulus *p*. Try to understand the architecture and the design file ("modaddsubn.vhd"). You also need "addsubn.vhd" as a submodule. Simulate the behavior of the module using the testbench ("tb_modaddsubn.vhd"). We will use this module in the design of the elliptic curve point doubling.

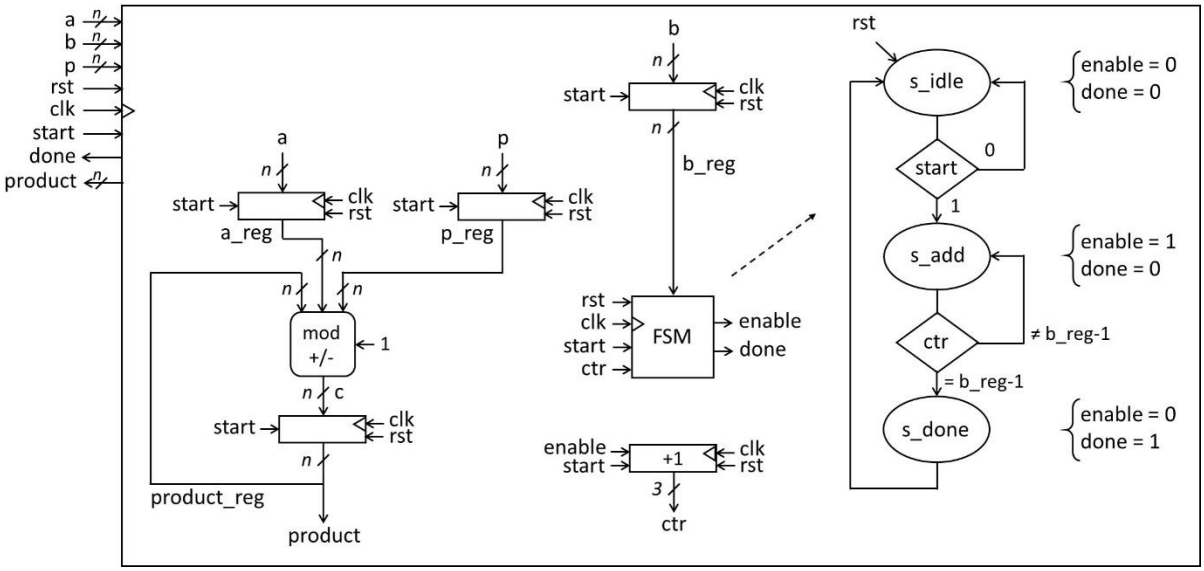
6. Simulate the n-bit constant multiplier (multiplication by 5)

This is a module that we will not use in the design of the elliptic curve point doubling, but it should be a good starting point for the following two exercises on n-bit modular multipliers. Try to understand the architecture and the design file ("modaddn_mult5.vhd") and run the testbench ("tb_modaddn_mult5.vhd"). The architecture of the design module contains input registers to store *a* and *p*. New values are loaded when *start* = 1. The *start* signal also initiates a finite state machine (FSM) that interacts with a counter to make sure the modular addition is performed 5 times.



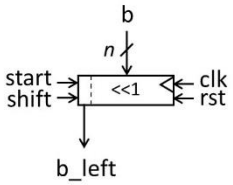
7. EXERCISE: Design and simulate an n-bit multiplier (by consecutive modular additions) and simulate the behavior

Start from the design module template ("modaddn_mult.vhd") and use the testbench ("tb_modaddn_mult.vhd"). The goal is to build a modular multiplier by consecutive modular additions of the multiplicand. Note that this is not an efficient way of implementing a modular multiplier, because it requires an impractical number of modular additions to be executed.



8. EXERCISE: Design and simulate an n-bit modular multiplier (through a left-to-right modular double-and-add algorithm)

Start from the design module template ("modmultn.vhd") and use the testbench ("tb_modmultn.vhd"). The architecture of the design module contains input registers to store *a*, *b* and *p*. The register to store *b* is also shiftable. When the *shift* input is active, the register shifts its content one position to the left (and shifts in a 0 on the right side). The registers are already present in the template design file.



INPUTS: a = (a_{n-1}, …, a_0), b = (b_{n-1}, …, b_0), p = (p_{n-1}, …, p_0)
OUTPUT: b*a mod p

1. s ← 0
2. for i from n-1 downto 0 do
   2.1 s ← s + s
   2.2 if b_i = 1 then s ← s + a
3. return s

The FSM uses the left bit of *b* to determine the next step in the algorithm. First, draw the FSM and the hardware architecture. Then, design the architecture in VHDL. Finally, run the simulation to check if the product is as expected.

9. Simulate the single-port memory unit

Use "ram_single.vhd" and "tb_ram_single.vhd". Try to understand how the register file works and run the simulation.

10. EXERCISE: Design and simulate a point doubling architecture

Design the hardware architecture for the n-bit point doubling operation given by the algorithm below (source: "Complete addition formulas for prime order elliptic curves", Eurocrypt 2016, Renes et al, https://eprint.iacr.org/2015/1060.pdf). Use the memory unit, the modular multiplier and the modular adder/subtracter. Start from the design module template "ecc_double.vhd". Draw the architecture on a piece of paper first. First use the testbench "tb_ecc_double_small.vhd" to test your design with an 8-bit curve. Then use the testbench "tb_ecc_double_nist.vhd" to test your design with a standardized 256-bit NIST curve.

Algorithm 6: Exception-free point doubling for prime order short Weierstrass curves $E/\mathbb{F}_q$: $y^2 = x^3 + ax + b$ with $a = -3$.

Require: $P = (X : Y : Z)$ on E: $Y^2Z = X^3 - 3XZ^2 + bZ^3$.

Ensure: $(X_3 : Y_3 : Z_3) = 2P$.

1. $t_0 \leftarrow X \cdot X$	2. $t_1 \leftarrow Y \cdot Y$	3. $t_2 \leftarrow Z \cdot Z$
4. $t_3 \leftarrow X \cdot Y$	5. $t_3 \leftarrow t_3 + t_3$	6. $Z_3 \leftarrow X \cdot Z$
7. $Z_3 \leftarrow Z_3 + Z_3$	8. $Y_3 \leftarrow b \cdot t_2$	9. $Y_3 \leftarrow Y_3 - Z_3$
10. $X_3 \leftarrow Y_3 + Y_3$	11. $Y_3 \leftarrow X_3 + Y_3$	12. $X_3 \leftarrow t_1 - Y_3$
13. $Y_3 \leftarrow t_1 + Y_3$	14. $Y_3 \leftarrow X_3 \cdot Y_3$	15. $X_3 \leftarrow X_3 \cdot t_3$
16. $t_3 \leftarrow t_2 + t_2$	17. $t_2 \leftarrow t_2 + t_3$	18. $Z_3 \leftarrow b \cdot Z_3$
19. $Z_3 \leftarrow Z_3 - t_2$	20. $Z_3 \leftarrow Z_3 - t_0$	21. $t_3 \leftarrow Z_3 + Z_3$
22. $Z_3 \leftarrow Z_3 + t_3$	23. $t_3 \leftarrow t_0 + t_0$	24. $t_0 \leftarrow t_3 + t_0$
25. $t_0 \leftarrow t_0 - t_2$	26. $t_0 \leftarrow t_0 \cdot Z_3$	27. $Y_3 \leftarrow Y_3 + t_0$
28. $t_0 \leftarrow Y \cdot Z$	29. $t_0 \leftarrow t_0 + t_0$	30. $Z_3 \leftarrow t_0 \cdot Z_3$
31. $X_3 \leftarrow X_3 - Z_3$	32. $Z_3 \leftarrow t_0 \cdot t_1$	33. $Z_3 \leftarrow Z_3 + Z_3$
34. $Z_3 \leftarrow Z_3 + Z_3$		