

Hands-on hardware tutorial

Nele Mentens

Selected Areas in Cryptography Summer School

August 13, 2019, Waterloo, Canada

Outline

- Implementation platforms + design flows
- Introduction to VHDL
- Hardware tutorial

Outline

- **Implementation platforms + design flows**
- Introduction to VHDL
- Hardware tutorial

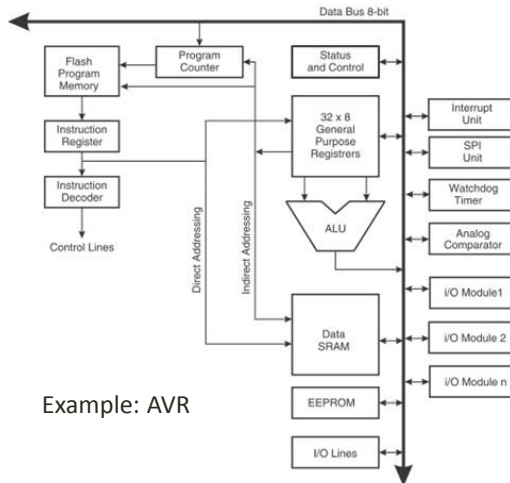
SAC summer school, 2019, Waterloo, Canada

Implementation platforms

- Microprocessor
- FPGA = Field-Programmable Gate Array
- ASIC = Application-Specific Integrated Circuit

SAC summer school, 2019, Waterloo, Canada

Microprocessor architecture



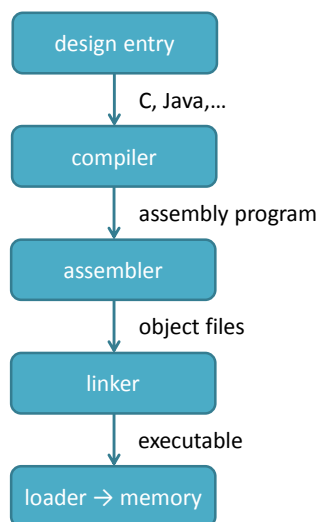
Example: AVR

The CPU is the heart of a microprocessor and contains a.o.:

- ALU (Arithmetic Logic Unit)
- register file
- program memory

SAC summer school, 2019, Waterloo, Canada

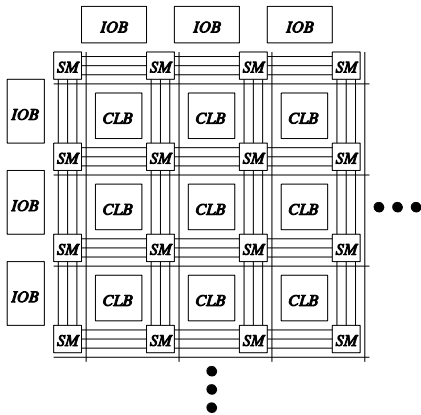
Microprocessor design flow



- The hardware architecture of a microprocessor is fixed
- The code describes what should be executed on the fixed hardware
- The instructions end up in the program memory

SAC summer school, 2019, Waterloo, Canada

FPGA architecture



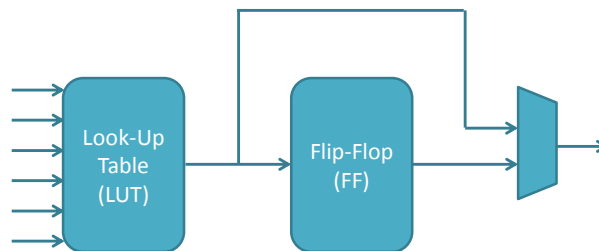
Basic components:

- CLB = Configurable Logic Block
 - CLBs consist of slices.
 - Slices consist of
 - Look-Up Tables (LUTs),
 - Multiplexers,
 - Flip-Flops (FFs),
 - Carry logic.
- SM = Switch Matrix
- IOB = Input/Output Block

SAC summer school, 2019, Waterloo, Canada

FPGA slice

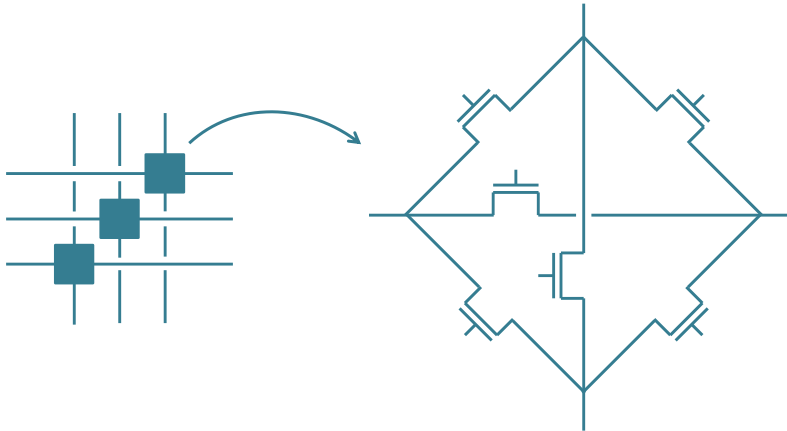
basic content of a slice



SAC summer school, 2019, Waterloo, Canada

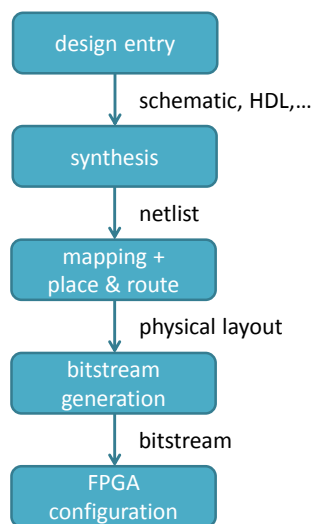
FPGA switch matrix

basic principle of a switch matrix



SAC summer school, 2019, Waterloo, Canada

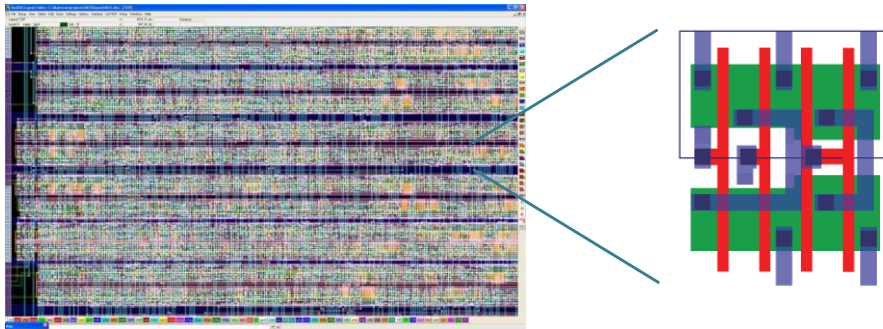
FPGA design flow



- The hardware architecture of an FPGA is configurable
- The code describes the hardware that we need
- The bitstream ends up in the configuration memory
- The area is measured in terms of occupied LUTs, flip-flops, dedicated hardware blocks

SAC summer school, 2019, Waterloo, Canada

ASIC architecture

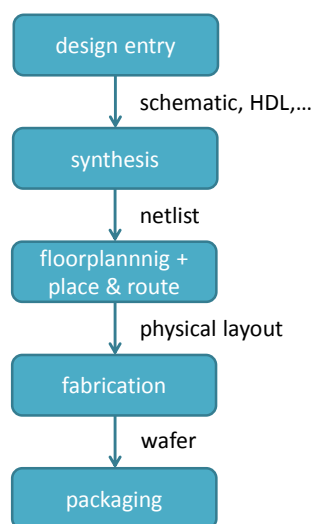


Basic components:

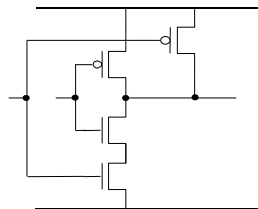
- Standard cells from a standard cell library
 - Logic cells and sequential cells

SAC summer school, 2019, Waterloo, Canada

ASIC design flow

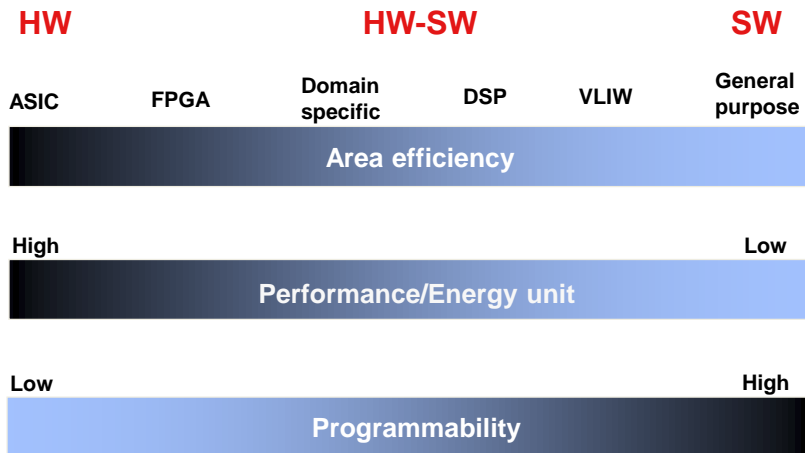


- The hardware architecture of an ASIC is defined by the designer
- The code describes the hardware that we need
- The GDS file contains the physical information that goes to the foundry
- The area is measured in terms of the number of equivalent NAND gates (Gate Equivalent = GE)



SAC summer school, 2019, Waterloo, Canada

Implementation platforms: comparison



SAC summer school, 2019, Waterloo, Canada

Outline

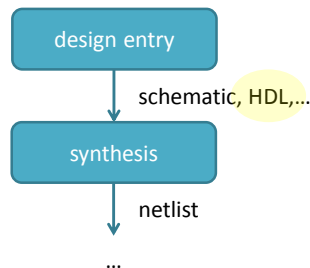
- Implementation platforms + design flows
- **Introduction to VHDL**
- Hardware tutorial

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Standard

- VHDL (VHSIC Hardware Description Language)
 - VHSIC = Very High Speed Integrated Circuit
- International standard
 - First standard: IEEE 1076-1987
 - Most recent update: IEEE 1076-2008
- Used for both ASIC and FPGA design

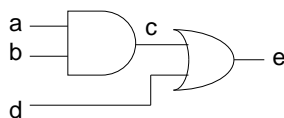


SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Hardware vs. software

- Description language for hardware \neq programming language
- Programming language (e.g. C):
 - hardware = processor
 - hardware is already designed, implemented and fabricated
 - code: describes how the hardware will be used
 - code is compiled for a specific processor
- Hardware description language (e.g. VHDL)
 - hardware = FPGA or ASIC design
 - hardware is designed
 - code: describes which hardware will be designed
 - code is synthesized for a specific FPGA or ASIC technology
 - example:



`c <= a and b;`
`e <= c or d;`

`e <= c or d;`
`c <= a and b;`

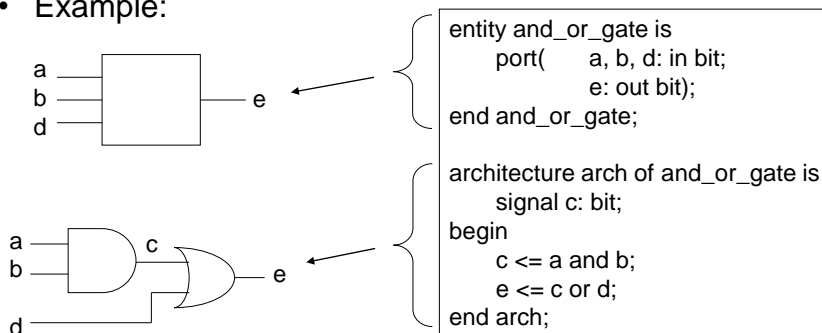
2x the same implementation

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Entities and architectures

- The VHDL code of each component consists of
 - an interface description: entity,
 - a behavioral description: architecture.
- Example:

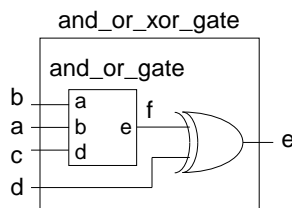


SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Hierarchy

- Hierarchy can be built in.
- There is hierarchy when a component contains an instantiation of another component.



```

entity and_or_xor_gate is
    port(a, b, c, d: in bit;
        e: out bit);
end and_or_xor_gate;

architecture arch of and_or_xor_gate is
    component and_or_gate is
        port(a, b, d: in bit;
            e: out bit);
    end component;
    signal f: bit;
begin
    inst_and_or_gate: and_or_gate
        port map(a => b,
            b => a,
            d => c,
            e => f);

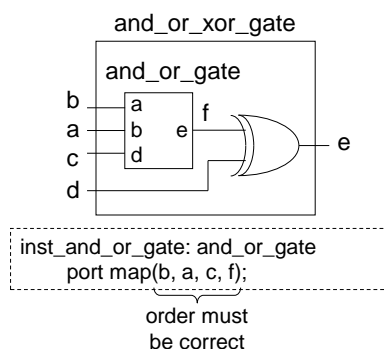
    e <= d xor f;
end arch;
    
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Hierarchy

- Hierarchy can be built in.
- There is hierarchy when a component contains an instantiation of another component.



```
entity and_or_xor_gate is
  port(a, b, c, d: in bit;
        e: out bit);
end and_or_xor_gate;

architecture arch of and_or_xor_gate is
  component and_or_gate is
    port(a, b, d: in bit;
          e: out bit);
  end component;
  signal f: bit;
begin
  inst_and_or_gate: and_or_gate
    port map(a => b,
             b => a,
             d => c,
             e => f);
  e <= d xor f;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

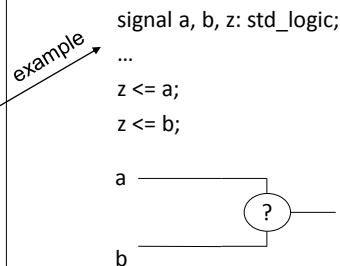
bit vs. std_logic

- The package “std_logic_1164” in library “ieee” contains a.o. the types “std_ulogic” en “std_logic”, consisting of 9 values (instead of 2 for “bit”)

```
type std_ulogic is (
  'U', -- Uninitialized
  'X', -- Forcing Unknown
  '0', -- Forcing 0
  '1', -- Forcing 1
  'Z', -- High Impedance
  'W', -- Weak Unknown
  'L', -- Weak 0
  'H', -- Weak 1
  '-', -- Don't Care);

subtype std_logic is resolved std_ulogic;

type std_ulogic_vector
  is array (NATURAL range <=>) of std_ulogic;
type std_logic_vector
  is array (NATURAL range <=>) of std_logic;
```



- It is advised to always use “std_logic” instead of “bit”

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Concurrent and sequential statements

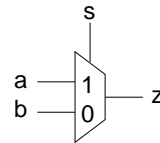
- Concurrent statements: are implemented in parallel and executed at the same time
- Sequential statements: can only occur in a process

example:

```
entity mux is
  port( a, b, s: in std_logic;
        z: out std_logic);
end mux;

architecture arch of mux is
begin
  p1: process(a, b, s)
  begin
    if s = '1' then
      z <= a;
    else
      z <= b;
    end if;
  end process;
end arch;
```

sensitivity
list

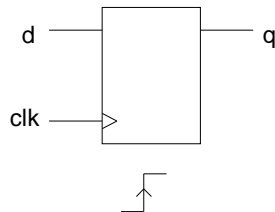


SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Storage elements

- D-flipflop:



```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
  port( d, clk: in std_logic;
        q: out std_logic);
end dff;

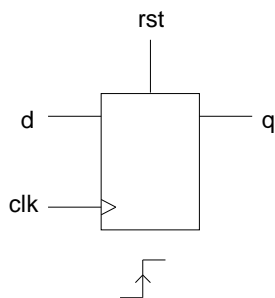
architecture arch of dff is
begin
  store: process(clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Storage elements

- D-flipflop with asynchronous reset:



```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
    port( d, clk, rst: in std_logic;
          q: out std_logic);
end dff;

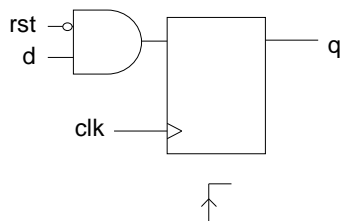
architecture arch of dff is
begin
    store: process(rst, clk)
    begin
        if rst = '1' then
            q <= '0';
        elsif clk'event and clk = '1' then
            q <= d;
        end if;
    end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Storage elements

- D-flipflop with synchronous reset:



```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
    port( d, clk, rst: in std_logic;
          q: out std_logic);
end dff;

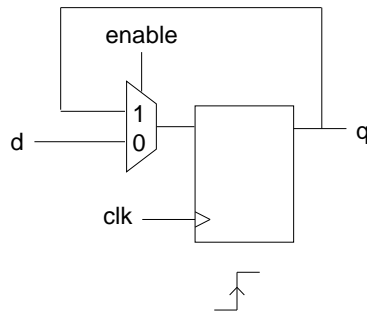
architecture arch of dff is
begin
    store: process(clk)
    begin
        if clk'event and clk = '1' then
            if rst = '1' then
                q <= '0';
            else
                q <= d;
            end if;
        end if;
    end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Storage elements

- D-flipflop with enable:



```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
    port( d, clk, enable: in std_logic;
          q: out std_logic);
end dff;

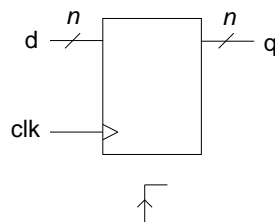
architecture arch of dff is
begin
    store: process(clk)
    begin
        if clk'event and clk = '1' then
            if enable = '1' then
                q <= d;
            end if;
        end if;
    end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Modules with parameters

- Register with a parameterizable width:



```
library ieee;
use ieee.std_logic_1164.all;

entity ffn is
    generic(size: integer:=4);
    port( clk: in std_logic;
          d: in std_logic_vector(size-1 downto 0);
          q: out std_logic_vector(size-1 downto 0));
end ffn;

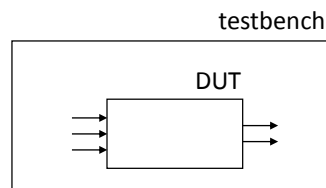
architecture arch of ffn is
begin
    p: process(clk)
    begin
        if clk'event and clk = '1' then
            q <= d;
        end if;
    end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Simulation

- A VHDL module can be simulated with a testbench:
 - Also written in VHDL
 - No ports in the entity
 - Containing an instantiation of the device under test (DUT)
- Input signals are applied internally in the testbench
- Output signals are evaluated
 - Through waveforms in a simulation window
 - In a text file
 - By comparing the behavior of the DUT to a golden reference model
 - in VHDL, directly in the testbench
 - in another programming language (e.g. C)



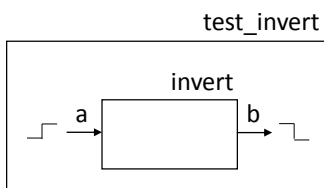
SAC summer school, 2019, Waterloo, Canada

Introduction to VHDL

Example

```
entity invert is
    port(
        a: in bit;
        b: out bit);
end invert;

architecture arch of invert is
begin
    b <= not a;
end arch;
```



```
entity test_invert is
end test_invert;

architecture arch of test_invert is
    signal a, b: bit;
    component invert is
        port(
            a: in bit;
            b: out bit);
    end component;
begin
    inst_invert: invert
        port map(a, b);

    p: process
    begin
        a <= '0';
        wait for 10 ns;
        a <= '1';
        wait for 10 ns;
        wait;
    end process;
end arch;
```

SAC summer school, 2019, Waterloo, Canada

Outline

- Implementation platforms + design flows
- Introduction to VHDL
- **Hardware tutorial**

SAC summer school, 2019, Waterloo, Canada

Hardware tutorial

Remember from Monday: “Introduction to elliptic curve cryptography”, *Craig Costello*

Scalar multiplications via double-and-add

How to (naively) compute $k, Q \mapsto [k]Q$?

$P \leftarrow Q$ $k = (k_n, k_{n-1}, \dots, k_0)_2$

for i from $n-1$ downto 0 do

$P \leftarrow [2]P$

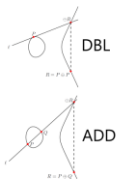
if $k_i = 1$ then

$P \leftarrow P \oplus Q$

end if

end for

return $P (= [k]Q)$



DBL

ADD

Projective space, cont.

- One practical benefit of working over \mathbb{P}^2 is that the explicit formulas for computing \oplus become much faster, by avoiding field inversions
- Thus, the fundamental ECC operation $k, P \mapsto [k]P$ becomes much faster...

$(x', y') = [2](x, y)$ $\lambda \leftarrow (3x^2 + a)/(2y)$; $x' \leftarrow \lambda^2 - 2x$; $y' \leftarrow -(\lambda(x' - x) + y)$; <div style="text-align: center; color: red;">1S + 2M + 1I</div>	$(X' : Y' : Z') = [2](X : Y : Z)$ $X' = 2XY((3X^2 + aZ^2)^2 - 8Y^2XZ)$ $Y' = (3X^2 + aZ^2)(12Y^2XZ - (3X^2 + aZ^2)^2) - 8Y^4Z^2$ $Z' = 8Y^3Z^3$ <div style="text-align: center; color: red;">5M + 6S</div>
--	---

SAC summer school, 2019, Waterloo, Canada

Hardware tutorial

In this tutorial, we will implement elliptic curve point doubling using projective coordinates

Scalar multiplications via double-and-add

How to (naively) compute $k, Q \mapsto [k]Q$?

$P \leftarrow Q$ $k = (k_n, k_{n-1}, \dots, k_0)_2$

for i from $n-1$ downto 0 do

$P \leftarrow [2]P$

if $k_i = 1$ then

$P \leftarrow P \oplus Q$

end if

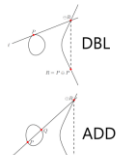
end for

return $P (= [k]Q)$

Projective space, cont.

- One practical benefit of working over \mathbb{P}^2 is that the explicit formulas for computing \oplus become much faster, by avoiding field inversions
- Thus, the fundamental ECC operation $k, P \mapsto [k]P$ becomes much faster...

$(x', y') = [2](x, y)$ $\lambda \leftarrow (3x^2 + a)/(2y)$; $x' \leftarrow \lambda^2 - 2x$; $y' \leftarrow -(\lambda(x' - x) + y)$; 1S + 2M + 1I	$(X' : Y' : Z') = [2](X : Y : Z)$ $X' = 2XY((3X^2 + aZ^2)^2 - 8Y^2XZ)$ $Y' = (3X^2 + aZ^2)(12Y^2XZ - (3X^2 + aZ^2)^2) - 8Y^4Z^3$ $Z' = 8Y^3Z^3$ 5M + 6S
---	--



SAC summer school, 2019, Waterloo, Canada

Hardware tutorial

The basic operations we need to implement, are:

- Modular addition
- Modular subtraction
- Modular multiplication
- Modular squaring

Scalar multiplications via double-and-add

How to (naively) compute $k, Q \mapsto [k]Q$?

$P \leftarrow Q$ $k = (k_n, k_{n-1}, \dots, k_0)_2$

for i from $n-1$ downto 0 do

$P \leftarrow [2]P$

if $k_i = 1$ then

$P \leftarrow P \oplus Q$

end if

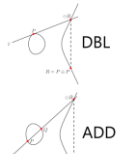
end for

return $P (= [k]Q)$

Projective space, cont.

- One practical benefit of working over \mathbb{P}^2 is that the explicit formulas for computing \oplus become much faster, by avoiding field inversions
- Thus, the fundamental ECC operation $k, P \mapsto [k]P$ becomes much faster...

$(x', y') = [2](x, y)$ $\lambda \leftarrow (3x^2 + a)/(2y)$; $x' \leftarrow \lambda^2 - 2x$; $y' \leftarrow -(\lambda(x' - x) + y)$; 1S + 2M + 1I	$(X' : Y' : Z') = [2](X : Y : Z)$ $X' = 2XY((3X^2 + aZ^2)^2 - 8Y^2XZ)$ $Y' = (3X^2 + aZ^2)(12Y^2XZ - (3X^2 + aZ^2)^2) - 8Y^4Z^3$ $Z' = 8Y^3Z^3$ 5M + 6S
---	--



SAC summer school, 2019, Waterloo, Canada

Hardware tutorial

- The basic operations we need to implement, are:
- Modular addition
 - Modular subtraction
 - Modular multiplication
 - Modular squaring

The datapath will consist of very basic modules for these operations

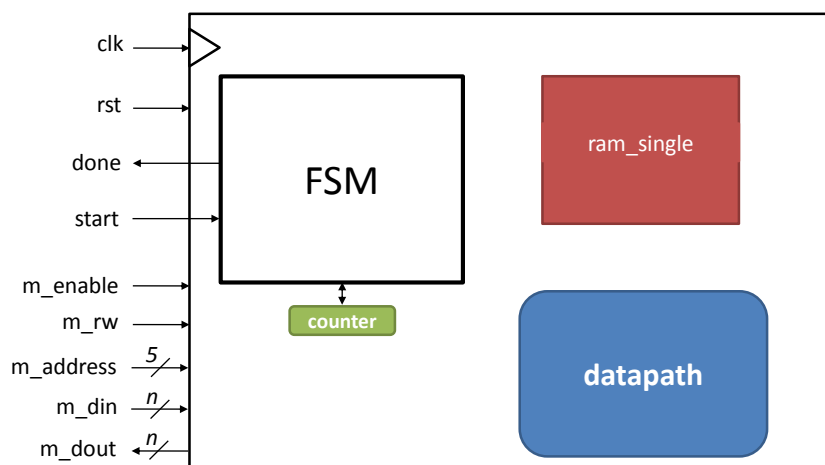
We will not design a dedicated modular squaring module; squaring can be done using the modular multiplication module

In order to store the intermediate values, a register file is needed

A finite state machine (FSM) will control the datapath and the register file

SAC summer school, 2019, Waterloo, Canada

Hardware tutorial



SAC summer school, 2019, Waterloo, Canada

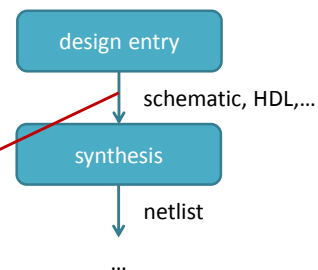
Hardware tutorial

- 4-bit adder
- n-bit adder
- 4-bit modular adder
- EXERCISE: n-bit modular adder
- n-bit modular adder/subtractor
- n-bit modular constant multiplier (multiplication by 5)
- EXERCISE: n-bit modular multiplier
 - through consecutive additions
- EXERCISE: n-bit modular multiplier
 - through left-to-right modular double-and-add
- single-port memory unit
- EXERCISE: elliptic curve point doubling

SAC summer school, 2019, Waterloo, Canada

Hardware tutorial

- For each module, the VHDL code for the module and the VHDL code for the testbench are given
- Where it says EXERCISE, the VHDL code for the module needs to be completed
- The tutorial will cover only behavioral simulation, i.e. pre-synthesis simulation based on the VHDL design without taking into account gate delays.



SAC summer school, 2019, Waterloo, Canada