Course overview

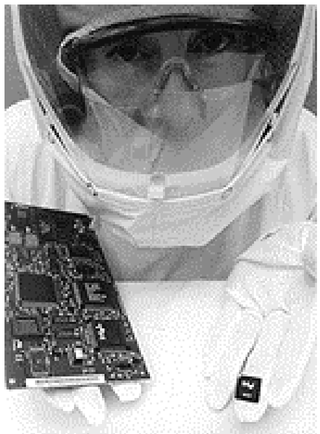
- Introduction
  - VLSI design
  - Modeling aspects
  - Design elements
- VHDL syntax
  - Main characteristics
  - Advanced features
- Specification
  - Organization
  - Verification
- Complex example
  - Partitioning
  - Design
- Simulation
  - Time model
  - Comparisons
- Synthesis
  - Combinational blocks
  - Sequential blocks

Introduction to VLSI Design

Cristiana Bolchini, Fabrizio Ferrandi, Franco Fummi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
bolchini | ferrandi | fummi@elet.polimi.it

VLSI Design

- VLSI increasing complexity
- "Time to market"



Structural Design

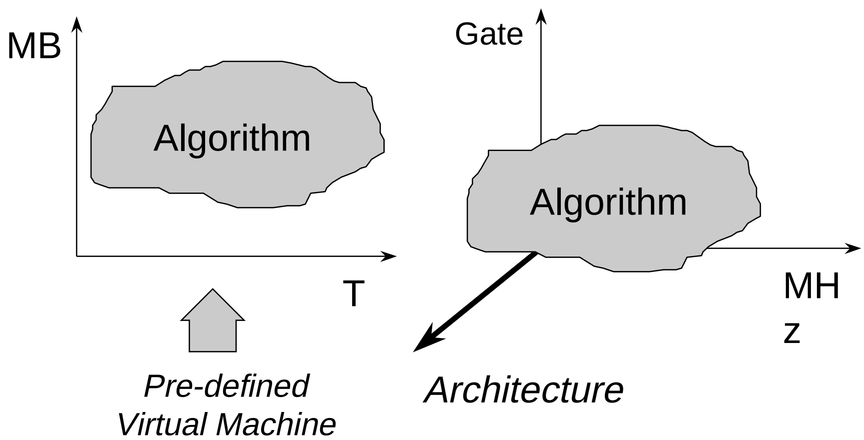
↓

Algorithmic Design

↓

Hardware Description Languages

Software versus Hardware Design



Example: GCD modeled in C

```
#include <stdio.h>

int gcd(int xi, int yi)
{
    int x, y, temp;

    x = xi;
    y = yi;
    while (x > 0){
        if (x <= y){
            temp = y;
            y = x;
            x = temp;
        }
        x = x - y;
    }
    return(y);
}
```

```
main()
{
    int xi, yi, ou;

    scanf("%d %d", &xi, &yi);
    ou = gcd(xi, yi);
    printf("%d\n", ou);
}
```


Hardware requirements (1)

- Input /Output
  - S `printf, scanf...`
  - H component interface must be defined
- Timing
  - S CPU instructions are executed at the CPU clock speed
  - H one or more explicit CLOCK signals must be defined

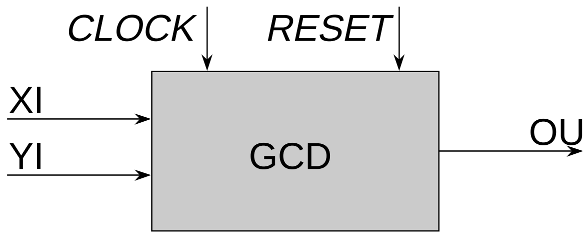
Hardware requirements (2)

- Variables size
  - S hidden implicit definition (integer 4bytes, char 1byte, …)
  - H all pre-defined and user-defined types must be translated into bit vectors
- Relationships operands/operators
  - S *all* operators in the C libraries are accepted
  - H explicit mapping of operands on operators

Hardware requirements (3)

- Memory elements identification
  - S the optimization module of the compiler transparently maps variables onto CPU registers and memory elements
  - H the synthesis tool identifies memory elements by analyzing the algorithmic semantics
- Modules synchronization
  - S sequential execution of instructions
  - H inherently parallel execution of all components

Entity/Architecture




```
ENTITY gcd IS
    PORT ( clock, reset : IN bit;
            xi,yi : IN unsigned (size-1 DOWNTO 0);
            ou : OUT unsigned (size-1 DOWNTO 0)
    );
END gcd;
```

variable size

Example: GCD modeled in VHDL

```
ARCHITECTURE behavioral OF gcd IS
BEGIN
    PROCESS
        VARIABLE x, y,temp : unsigned (size-1 DOWNTO 0);
    BEGIN
    WAIT UNTIL clock = '1';
    x := xi;
    y := yi;
    WHILE (x > 0) LOOP
        IF (x <= y) THEN
            temp:=y;
            y := x;
            x:=temp;
        END IF;
        x := x - y;
    END LOOP;
    ou <= y;
    END PROCESS;
END behavioral;
```

timing

memories

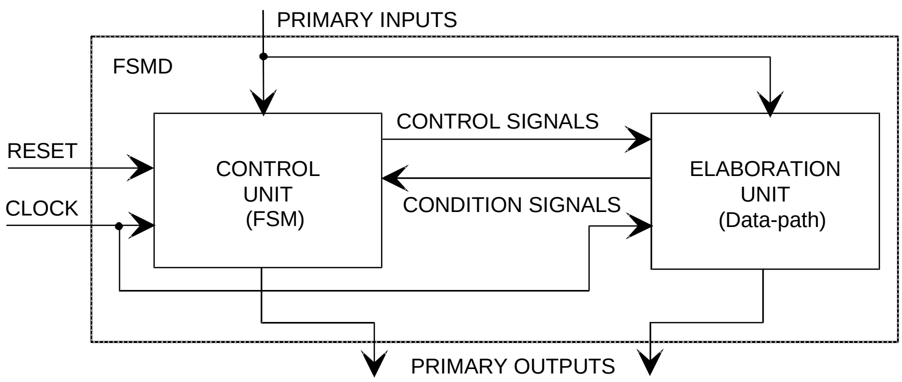
Algorithmic (Behavioral) Synthesis

- Identification of a *target* architecture
  - FSM + Data-Path (FSMD)
- Identification of time instants for each operation:
  - time order
  - time length

  ⇒ scheduling
- Identification of operators:
  - data size
  - time performances

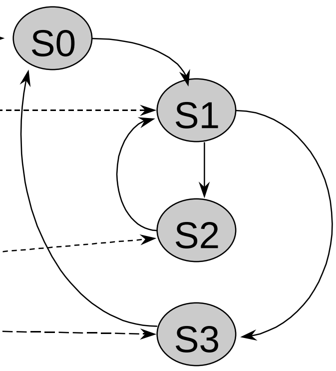
  ⇒ allocation

FSMD Model



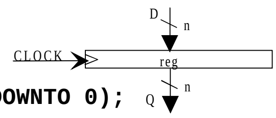
Scheduling (FSM)

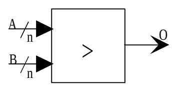
```
ARCHITECTURE behavioral OF gcd IS
BEGIN
   PROCESS
      VARIABLE x, y,temp : unsigned (size-1 DOWNTO 0);
   BEGIN
   WAIT UNTIL clock = '1';
   x := xi;
   y := yi;
   WHILE (x > 0) LOOP
      IF (x <= y) THEN
         temp:=y;
         y := x;
         x:=temp;
      END IF;
      x := x - y;
   END LOOP;
   ou <= y;
   END PROCESS;
END behavioral;
```

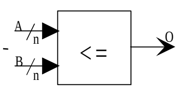


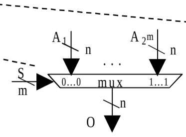
Allocation (Data-Path)

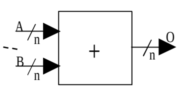
```
ARCHITECTURE behavioral OF gcd IS
BEGIN
   PROCESS
      VARIABLE x, y,temp : unsigned (size-1 DOWNTO 0);
   BEGIN
   WAIT UNTIL clock = '1';
   x := xi;
   y := yi;
   WHILE (x > 0) LOOP
      IF (x <= y) THEN
         temp:=y;
         y := x;
         x:=temp;
      END IF;
      x := x - y;
   END LOOP;
   ou <= y;
   END PROCESS;
END behavioral;
```











Register Transfer Level (RTL)

- Interconnection of FSM + Data-path
  - control/condition signals are identified
  - FSM is represented by states and transitions
  - Data-path is represented by registers, multiplexers and operators

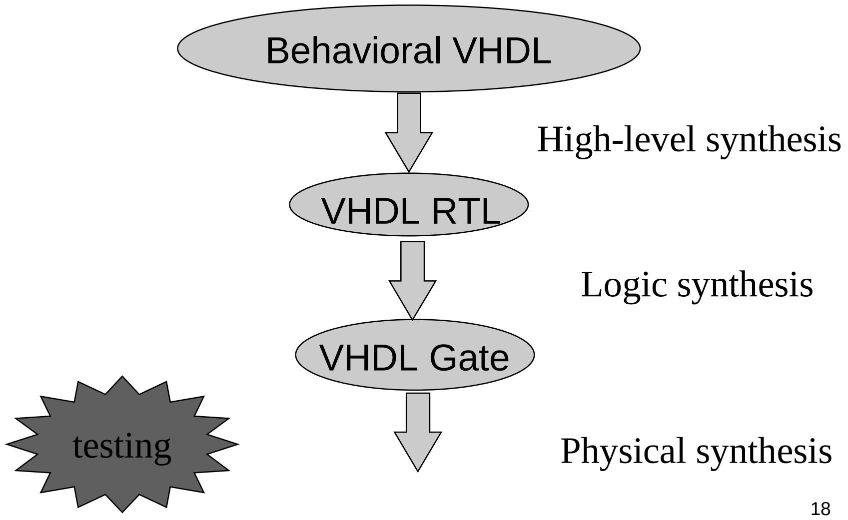
⇓

- Automatic translation into a set of registers and library components (RTL)

Logic Synthesis

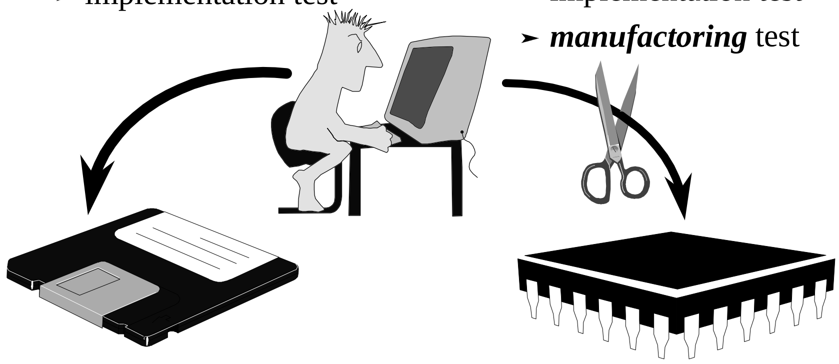
- Each RTL module is optimized by means of:
  - area
  - delay
  - power
- Efficient synthesis algorithms exist for:
  - two-level synthesis
  - multiple-level synthesis
- A logic representation of each RTL module is generated and optimized

General synthesis flow

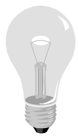


Software versus Hardware Testing

- Software
  - specification test
  - implementation test
- Hardware
  - specification test
  - implementation test
  - ***manufactoring*** test



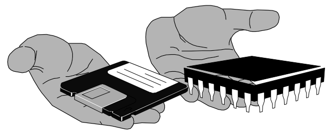
Trends for Algorithmic Level



```
IF (x <= y) THEN
    temp:=y;
    y := x;
    x:=temp;
END IF;
```

- Power estimation

- Testability estimation



- HW/SW partitioning