H05H4, H05E7, Chapter II
Ontwerp van micro-elektronische systemen
Design of micro-electronic systems

Ingrid Verbauwhede,
Acknowledgements: H. De Man
2009
KULeuven

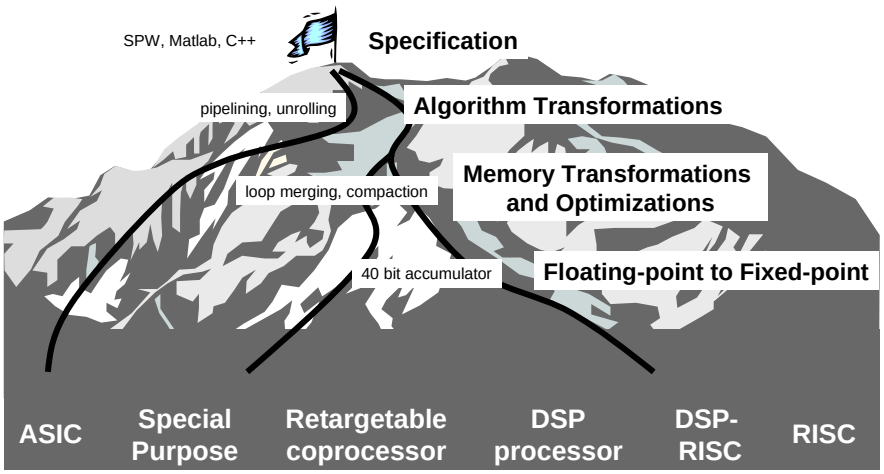
– 1

Summary Lecture 1

- **Introduction: Post PC-area**
- **what is a Integrated System-on-Chip?**
  - **Very heterogeneous device**
  - **Many IP modules**
  - **Much more than a general purpose processor**
- **What is system-on-chip design?**
  - **Vertical refinement of design steps**
  - **Horizontal design space exploration**
  - **Skiing down a mountain**
- **How do you describe hardware?**
  - **GEZEL hardware description language**

– 2

Skiing down a mountain



– 3

Today: Terminology

Skiing down = "methodology"

When skiing down:

- **Terminology**
- **Levels of abstraction**
- **Y-chart of Gajski-Kuhn**
- **Models of computation**

Reading:

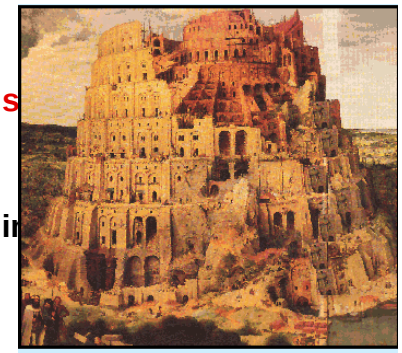
- **Chapter II of Course notes: Methodology & Terminology**

– 4

II. Methodology and Terminology

Think and plan before you act!

- ***Methodology*** **is a set of practices that, when applied in the right sequence, allow engineers to *master* the design process**
- ***Master*****: designing the right s given cost and time budget**
- ***Terminology*****: tools and engi disaster...**



Tower of Babel, in painting by Bruegel, 1563

– 5

Outline

- **Design Space and Design Flow**
- **Hierarchy and Abstraction Levels**
- **Design Flow for SoC's**
- **CAD tools**
- **Design Methodology**

– 6

Design Space and Design Flow

Design: translate abstract idea into interrelation of elements that, by exchanging info amongst themselves, achieve a useful interaction with the environment

Design flow: systematic sequence of well defined design activities that, in a time and cost effective way, leads to a production plan

A design flow traverses a *design space* based on 3 system representation axes

– 7

Three Design Representations

The 3 axes:

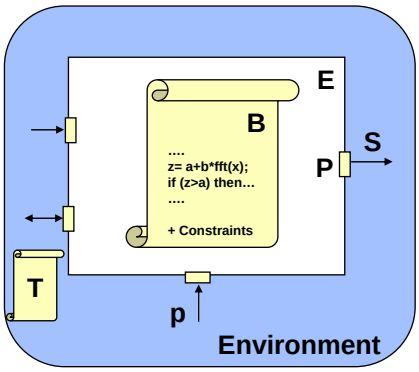
- **Behaviour (What?)**
- **Structure (How? = Composition of sub-behaviours)**
- **Physical Implementation (Production Plan, Geometry)**

– 8

Behaviour (gedrag)

Behaviour ::= relation between signals defining interaction with environment

Signal: type, time representation, dimension
Port : “ “ “ + mode



E: entity
P: port,
S: signal
p: parameters, generics
B: behaviour
T: testbench

First design task: **translate requirements into formal behaviour**
= concurrently executable Models of Computation (MoC)
Specs **= behaviour + constraints + testbench**

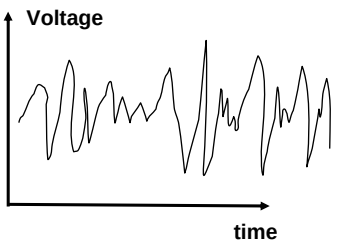
– 9

Signals

Signals are functions that map time into values

[X -> Y] = {f | domain(f) = X & range(f) = Y}
X = time domain
Y = some other physical value

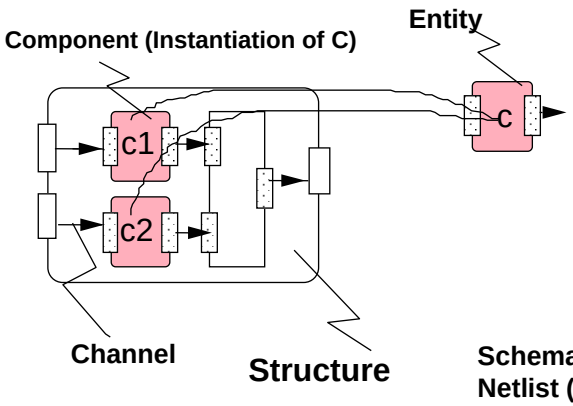
Ex.: Sound signals
= waveform with time dimension
and voltage levels



– 10

Structure

Results from refinement of behaviour in sub-behaviours of simpler nature…
A way to master complexity...



– 11

Example: VHDL Structural description

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity SOMETHING is
  port (SEL, A, B: in STD_LOGIC;
        F : out STD_LOGIC);
end;
architecture STRUCTURAL of SOMETHING is
component INV
  port (A: in STD_LOGIC;
        F: out STD_LOGIC);
end component;
component AOI
  port (A, B, C, D: in STD_LOGIC;
        F : out STD_LOGIC);
end component;
signal SELB: STD_LOGIC;
begin
  G1: INV port map (SEL, SELB);
  G2: AOI port map (SEL, A, SELB, B, F);
end;
```

Gate level

Entity

Structural architecture

Component declarations

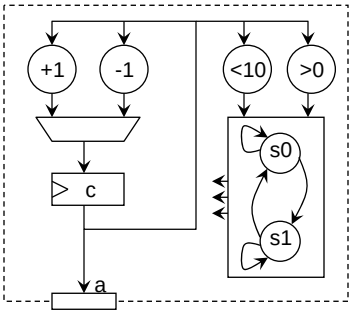
Internal Signal declaration

Structure of entity

– 12

An FSMD in GEZEL

```
dp updown(out a : ns(4)) {
  reg c  : ns(4);
  sfg inc { c = c + 1;
            a = c;     }
  sfg dec { c = c – 1;
            a = c;     }
}


fsm ctl_updown(updown) {
  initial s0;
  state   s1;
  @s0 if (c < 10) then (inc) -> s0;
                  else (dec) -> s1;
  @s1 if (c > 0)  then (dec) -> s1;
                  else (inc) -> s0;
}
```



– 13

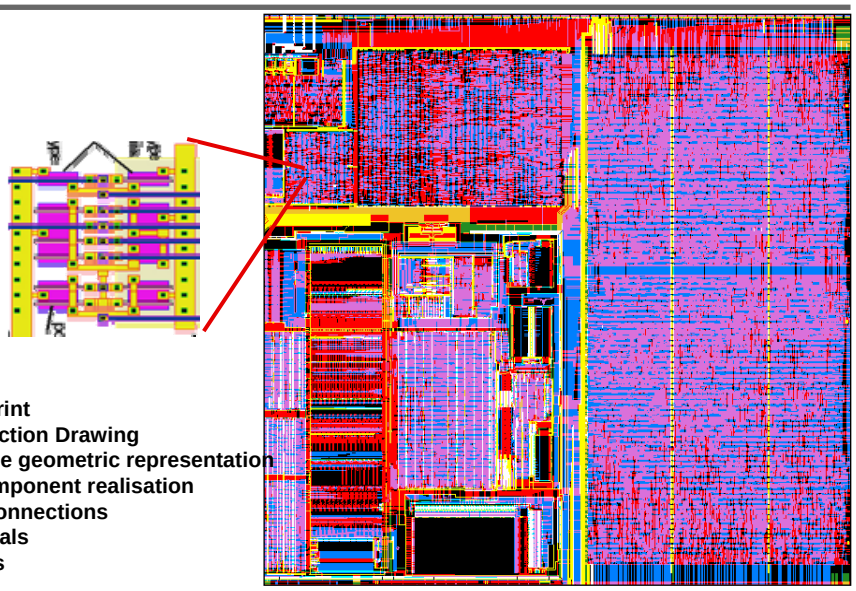
To compare: Equivalent SystemC model

```
SC_MODULE(fsm_counter) {
  sc_in <bool>          clk;
  sc_in <sc_uint<2> >             ;
  sc_out<sc_uint<3> >            ;
  sc_signal<int>        state, state_next;
  void eval_logic();
  void update_regs();
  SC_CTOR(fsm_counter) {
    SC_METHOD(eval_logic);
    sensitive << flags_counter << state;
    SC_METHOD(update_regs);
    sensitive_pos(clk);
    state = state_next = 0;
  }
};
void fsm_counter::eval_logic() {
  sc_uint<3> flags = flags_counter.read();
  switch(state) {
  case 0:
   if (flags[0]) {
    state_next = 1;
    ins_counter.write(c_do_dn | c_do_io);
   } else {
    state_next = 0;
    ins_counter.write(c_do_up | c_do_io);
   }
   break;
  case 1:
   if (flags[1]) {
    state_next = 0;
    ins_counter.write(c_do_up | c_do_io);
   } else {
    state_next = 1;
    ins_counter.write(c_do_dn | c_do_io);
   }
   break;
  }
}
void fsm_counter::update_regs() {
  state = state_next;
}
```

```
const int counter_do_io = 1;
const int counter_do_up = 2;
const int counter_do_dn = 4;
SC_MODULE(dp_counter) {
  sc_in <bool>          clk;
  sc_in <sc_uint<3> >              ;
  sc_in <sc_uint<2> >    ud;
  sc_out<sc_uint<3> >    a;
  sc_out<sc_uint<2> >             ;
  sc_signal<sc_uint<3> > c, c_next;
  sc_signal<sc_uint<2> > u, u_next;
  sc_signal<sc_uint<3> > nc;
  void eval_logic();
  void update_regs();
  SC_CTOR(dp_counter) {
   SC_METHOD(eval_logic);
   sensitive << c << nc << ud;
   SC_METHOD(update_regs);
   sensitive_pos(clk);
   c = c_next = 0; u = u_next = 0;
  }
};
void dp_counter::eval_logic() {
  sc_uint<3> sfg = ins_counter.read();
  if (sfg & counter_do_io) {
   u_next = ud.read();
   a.write(nc);
    flags_counter.write(u);
  }
  if (sfg & counter_do_up) {
    nc = c.read() + 1;
    c_next = nc;
  }
  if (sfg & counter_do_dn) {
    nc = c.read() - 1;
    c_next = nc;
  }
}
void dp_counter::update_regs() {
  u = u_next;
  c = c_next;
 }
```


– 14

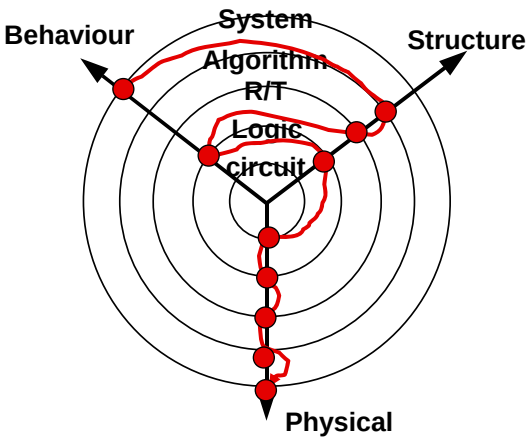
Physical Representation (Layout)



Blueprint
Production Drawing
Precise geometric representation
Of component realisation
Interconnections
Materials
Layers

– 15

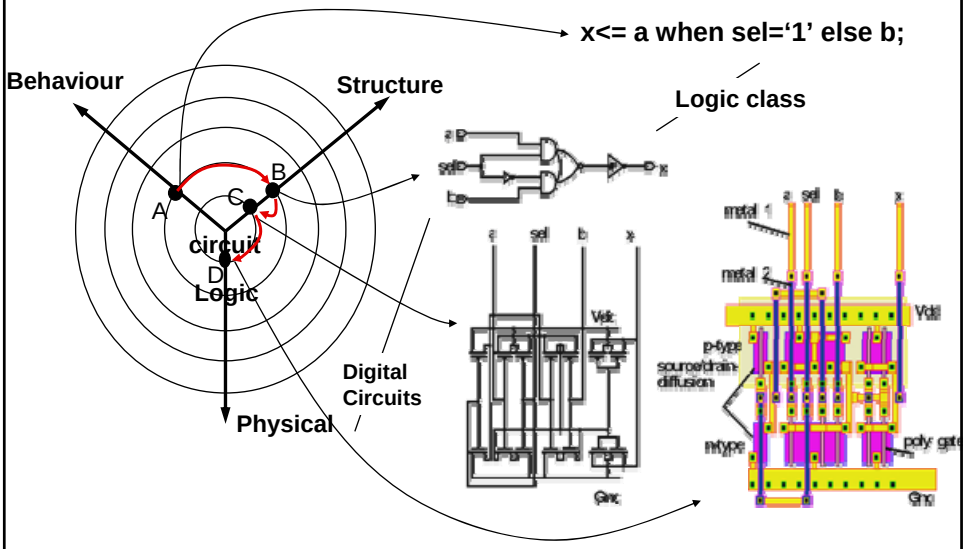
Y-Chart (Gajski-Kuhn)



Design Space: axis measures abstraction. High is less detailed
Levels of abstraction is circles
Design Flow is path in Y chart

– 16

Example of Design Flow



– 17

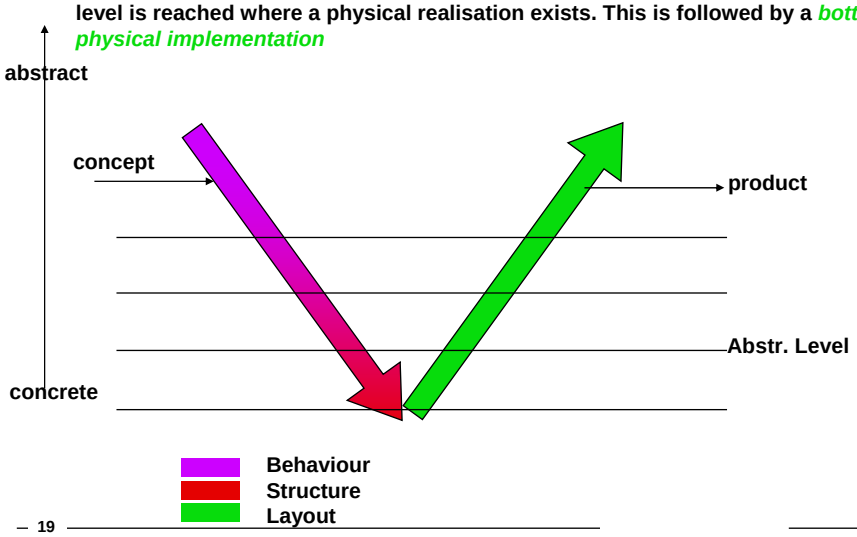
Outline

- Design Space and Design Flow
- **Hierarchy and Abstraction Levels**
- Design Flow for SoC's
- CAD tools
- Design Methodology

– 18

Hierarchy and Abstraction Levels

Design is *step by step* decomposition of abstract *behaviour* into a *managable* interaction of more and more subsystems (*structure*) at *lower abstraction* until a level is reached where a physical realisation exists. This is followed by a *bottom up physical implementation*



– 19

Abstraction levels

- Define steps to your goal. Decide what the important issues are at a given step. Focus only on these issues before proceeding.
  - "A model must be as simple as possible, but not simpler" (Dixit A. Einstein)
- Basis:
  - Signal Types
  - Time representation
  - Behavioural Primitives
  - Structural Primitives
  - Design Activities
- Digital signals and components much easier to abstract than analog!

– 20

Digital Abstraction Levels

System:
- From requirements to executable behaviour,
- ADT, Concurrent Communicating Processes, Events, CT

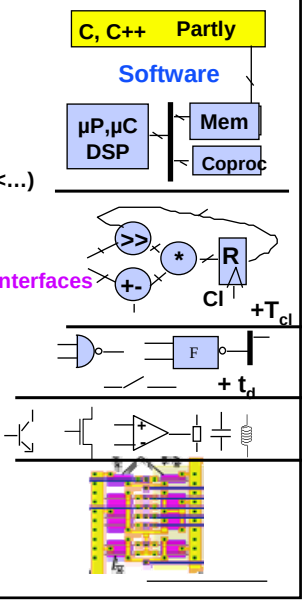
Algorithmic:
- Refinement of behaviour to Hw-Sw architecture
- DT, ADT to bitvector, int; primitive operations (+,-,*,>>,<<…)

Register Transfer:
- Clocked system: clock tick
- Bitvectors; RT-operations->RT-operators, FSM's, Store, Interfaces

Logic:
- Bit, Boolean, Std_Logic
- Int Gate Delay, Boolean fnct, FSM, gate, ff, switch

Transistor:
- v(t), i(t), ODE's, Netw, Eq. R.L,C,E,I,M…

Polygon



ADT = Abstract Data Type, DT = Discrete Time

– 21

Outline

- Design Space and Design Flow
- Hierarchy and Abstraction Levels
- **Design Flow for SoC's**
- CAD tools
- Design Methodology

– 22

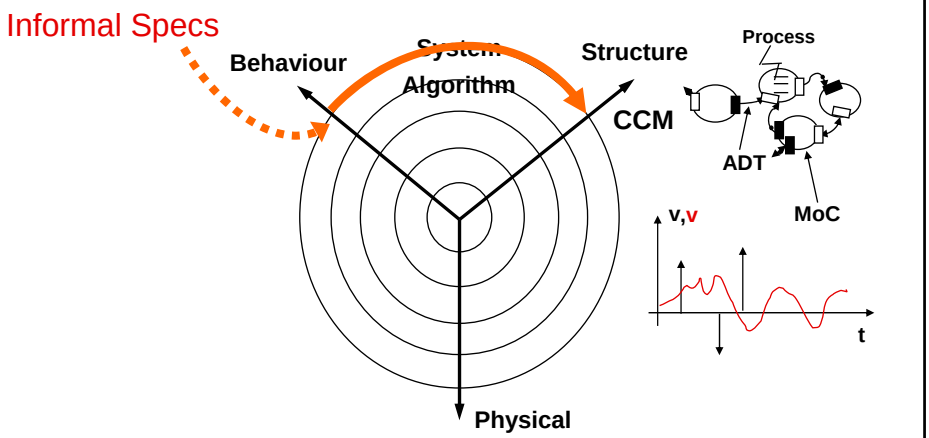
Design Flow for SoC

- ***Quick tour*** **of how SoC are designed from concept to layout**
- ***Definition*** **of Abstraction Layer, design representations and design activities**
- ***Terminology***

– 23

Level 1: System Level

From Informal Specs to
Executable *C*ommunicating *M*odels of Computation=Process



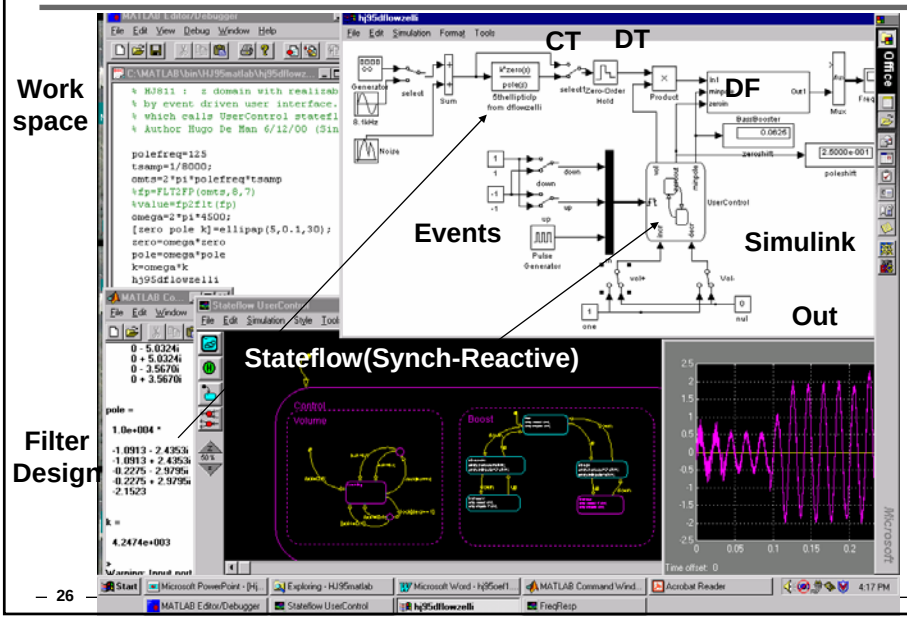
GOAL : functional validation: ARE WE DESIGNING THE RIGHT SYSTEM?
-> Golden specification

Tools (also for algorithmic level)

- **MATLAB-SIMULINK-STATEFLOW**
- **C++ Class libraries (System-C, SpecC)**
- **Co-design: GEZEL – FSMD with e.g. System-C and C specifactions**
- **(VHDL: too slow, lacks system concepts, incompatible with software parts)**

– 25

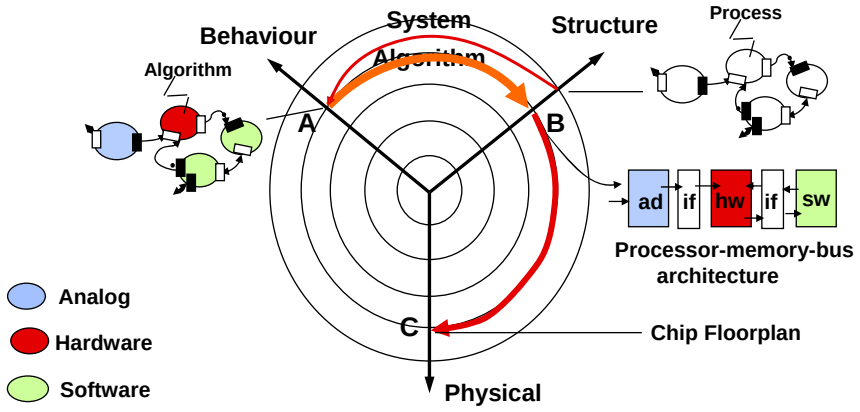
MATLAB-SIMULINK-STATEFLOW



– 26

Level 2: Algorithmic Level

A) Optimize process = algorithm. (Data, operations, storage refinement)
B) Architectural Mapping = Hw/Sw , processor allocation, assignment
C) Create chip Floorplan



– 27

Design activities at algorithmic level

ARE WE DESIGNING THE SYSTEM RIGHT?

- **Between System and Algorithm (70% of gain!!!)**
  - Code transformations: reduce ops, storage
  - Expand f(x) -> +, -, *,z^{-1}, >>,…
  - ADT-> bitvectors (precision! WL=cost)
  - Refine communication
  - Verify w.r.t. system spec
- **At Alg. Level : Map into architecture**
  - Hw-Sw-A/D partition
  - Allocation
  - Assignment
  - Software process scheduling (RTOS)
- **Floorplanning**

– 28

Example: Code transformation

```
c=[2.00  1.75  2.25] ;
for i=1:3
   y(i)=0;
   for j=1:3
      a(i,j)=max(x(j+(i-1)*3+1),x(j+(i-1)*3));
   end;
end;

for i=1:3
   for j=1:3
   y(i)=y(i)+c(i)*a(j,i);
   end;
   uit=y(i);
end;
```

```
c=[2.00  1.75  2.25] ;
for i=1:3
   y(i)=0;
end;
for i=1:3
   for j=1:3
      x1=x(j+(i-1)*3+1);
      x2=x(j+(i-1)*3);
      if x1-x2>0 max=x1; else max=x2;
      end;
      y(j)=y(j)+max;
   end;
end;
for i=1:3
   uit=c(i)*y(i);
end;
```

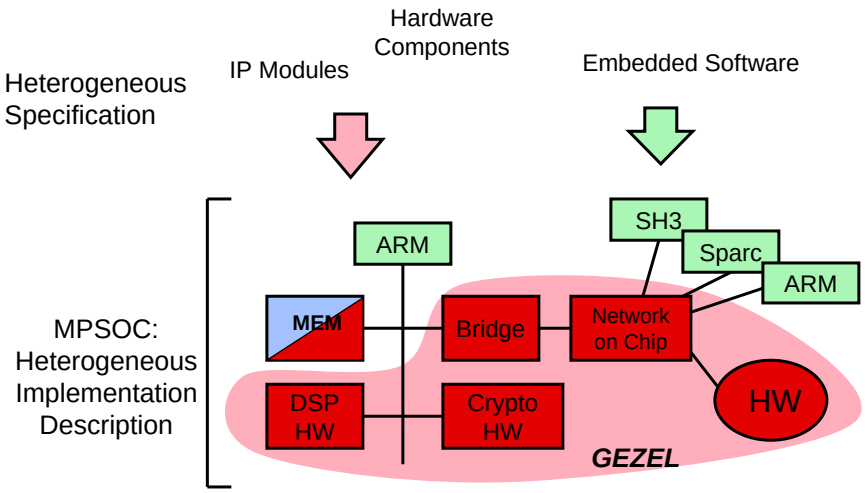
Matrix a(3,3)
3 registers y(3)
10 registers x(10)

Change loop traversal
Merge loops
Expand max function

3 y registers
2 x registers x1 en x2
Save: 22-5=17 registers of n bits
Save 9-3=6 mpy, simplify mpy!

– 29

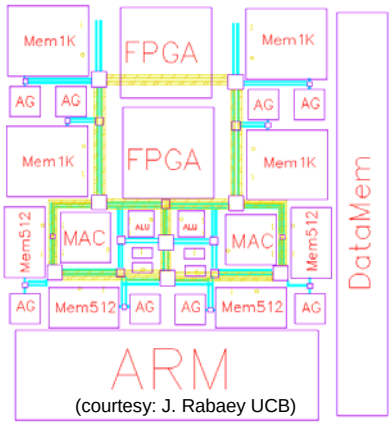
GEZEL Design Environment



– 30

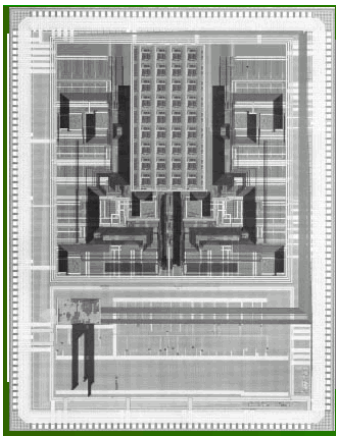
Floorplan and layout

Geometric organisation of chip based on area and W/L estimates of architectural components.



(courtesy: J. Rabaey UCB)

Floorplan



Final layout

– 31

Level 3: Register Transfer Level (RT)

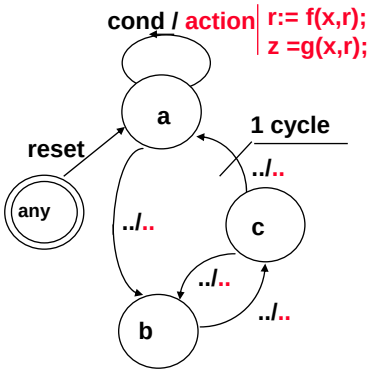
- ***System model:*** **FSMD.**
  Register ->RT-operations->Register
- ***Time:*** **clock cycle**
- ***Signal:*** **bitvector, array of std_logic (VHDL)**
- ***Behaviour RT(BRT):*** **= communicating StateTransitionGraph (STG).**
  Focus= *time scheduling* of RT-operations
- ***Structure RT (SRT):*** **=FSMD: *operations to operators***

– 32

Start: Behavioural RT (BRT)

Datamodel = FSM + DP actions

Language template = clocked "case"



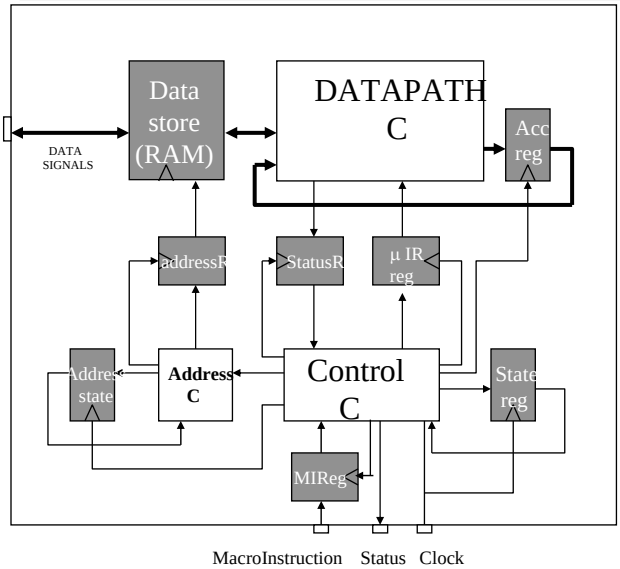

```
forever {
wait until cl'event and cl='1';
if(reset) state := a; else
case (state)
{
 a : action1;state:=nextstate(a);

 b : action2;state:=nextstate(b);

 c : action3;state:=nextstate(c);
}
        }
```

Evolution of state = schedule !
aaaabcbcbcaaa...

VHDL, Verilog, GEZEL,...

– 33

End: Structural RT level = FSMD



C : Combinational network of RT-operators

R(eg) M-S registers

if(c[&clock]) reg:=f(x,y,z):;
z = g(x,y,reg);

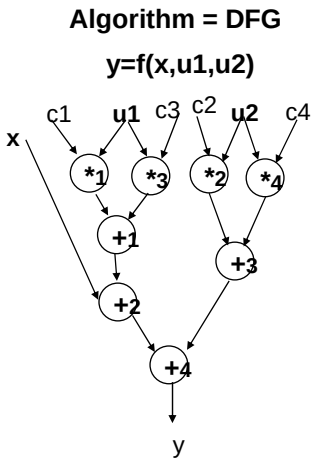
– 34

Design activities at RT

Three main tasks

- **Scheduling**
- **Allocation**
- **Assignment**

Example: algorithm description in data flow graph representation (DFG)



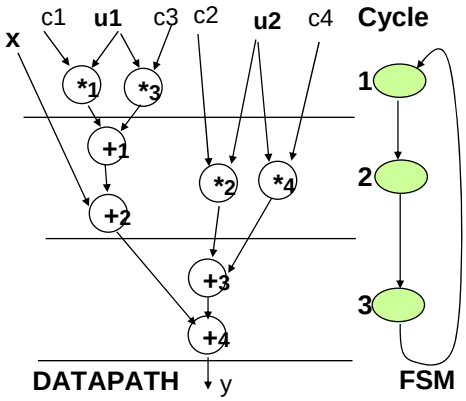
– 35

Task 1: Scheduling

Decide the clock cycle for each operation

Optimization goal can be:

- **Mimimum execution time**
- **Minimum hardware cost**
- **Something in between**



Schedule for 2mpy||, 2add-> (allocation)

– 36

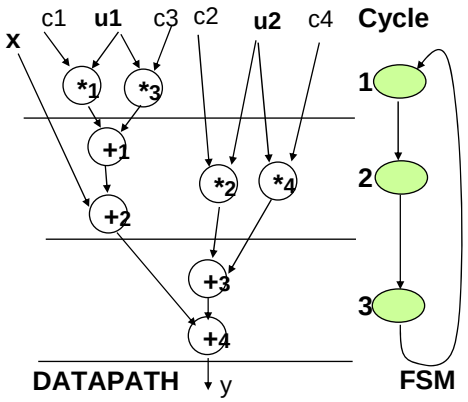
Task 2: Allocation

Decide how many operatORS of each type are needed

Example:

2 multipliers, 2 adders

A strategy to decide this automatically?



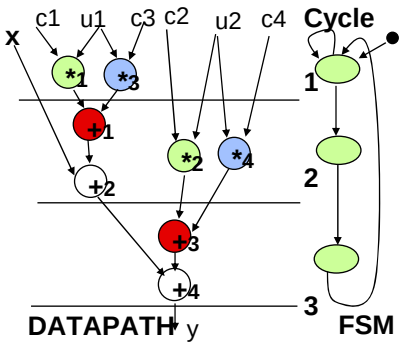
– 37

Task 3: Assignment

Decide which operatIONS goes on which operaTOR

Such that no access conflicts (I.e. every unit can be used only once in every clock cycle)

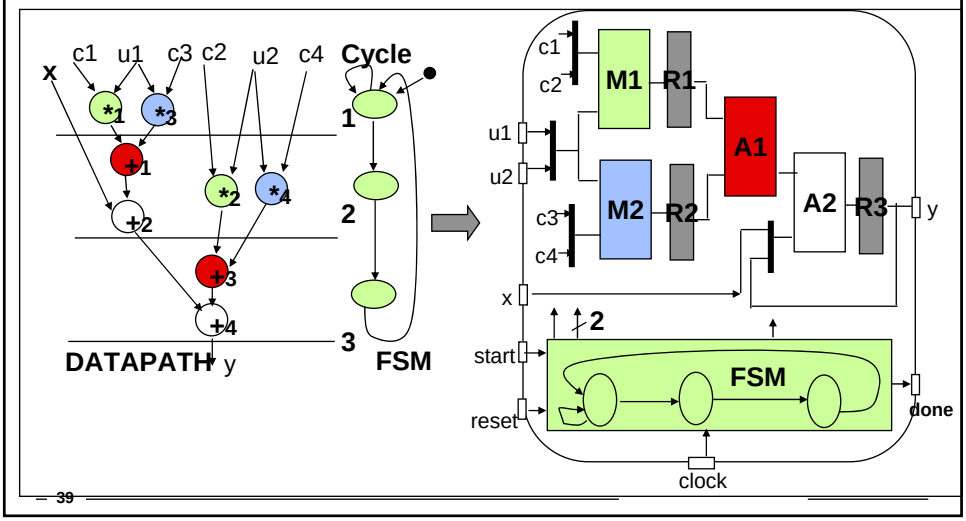
Such that minimum overall cost, mainly interconnect, bus and storage cost.



– 38

RT synthesis

From BRT to optimized SRT



– 39

RT tasks

Scheduling/Allocation/Assignment

- **Not necessary all present**
- **E.g. compilation on an existing processor**

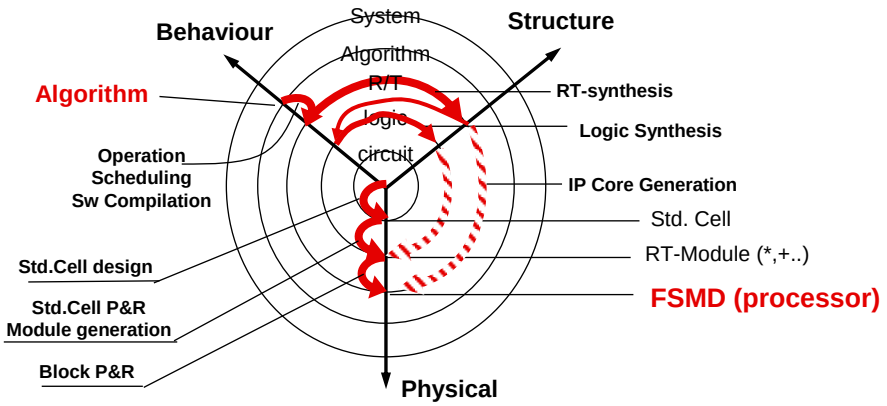
- **Not necessary in this sequence**
- **E.g. AR|T designer tools:**
  Allocation/Assignment/Scheduling

Carefully check what is the optimization goal:

Real-time constraint? Or minimum hardware?

– 40

Register-Transfer & Logic Design

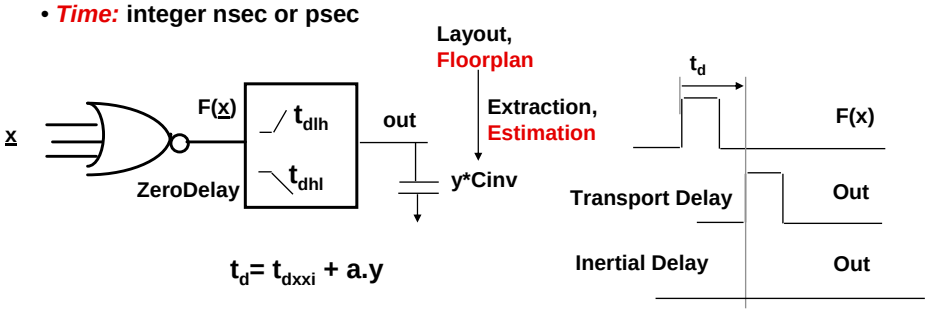


→ High Level Synthesis = Scheduling/Allocation/Assignment
= Silicon Compilation

– 41

Level 4: Logic Level Design

- ***Behaviour***: Boolean expressions, FSM
  Clocked VHDL: `wait until clock'event and clock='1'`
- ***Signal Types:*** Bit, Std_Logic (1,0,x,H,L,W,U,Z,-)
- ***Structure***: Gates, FF, Tri-State, tr. Gates, Load in Cinv
- ***Time:*** integer nsec or psec

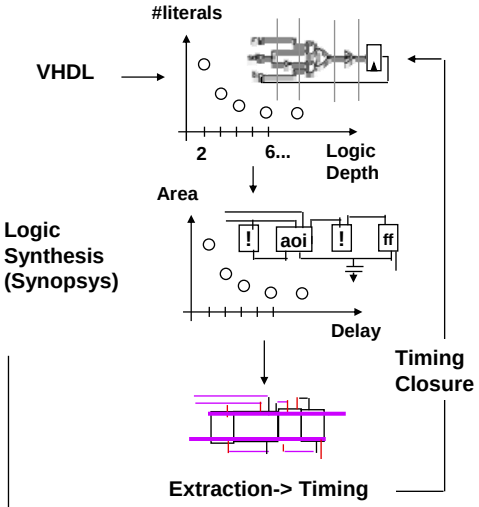


$t_d = t_{dxi} + a.y$

– 42

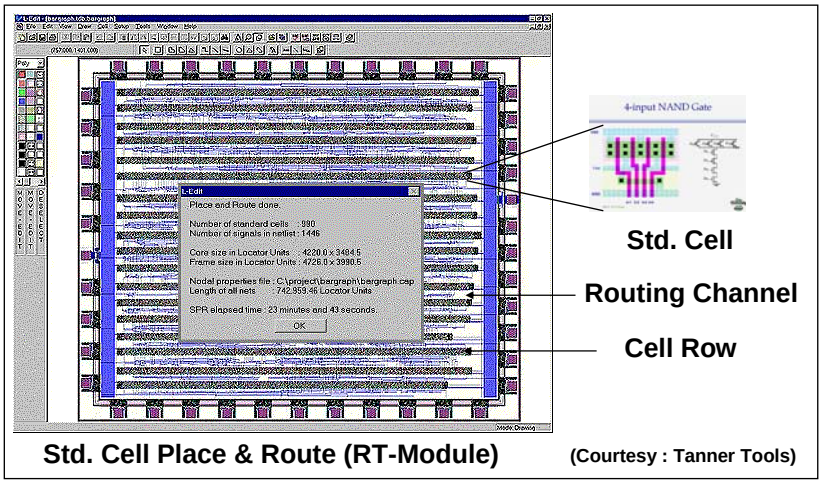
Logic Design Activities

- **Logic and FSM synthesis**
  - State minim., coding
  - Multilevel Logic Optimisation
- **Technology Mapping**
  - Functions to library cells
  - Minimal Area for given delay
- **Timing Verification**
  - Estimate wiring load C
  - Critical logic path
- **Layout**
  - P&R C extraction from wiring ...



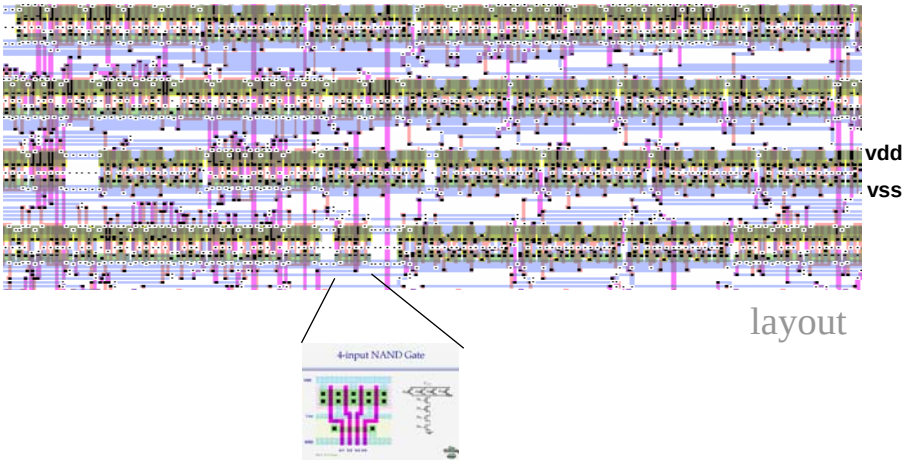
– 43

Standard Cell Layout



– 44

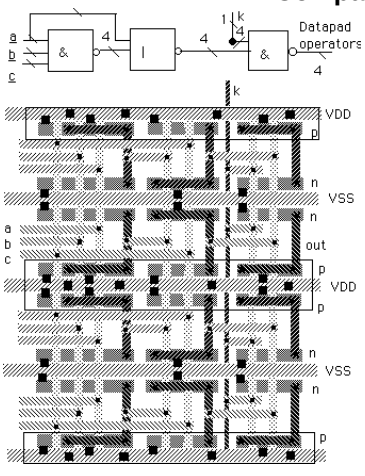
Standard Cell Zoom In

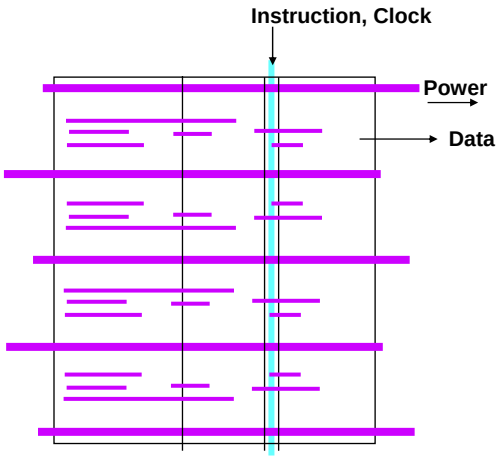


– 45

Module Generation

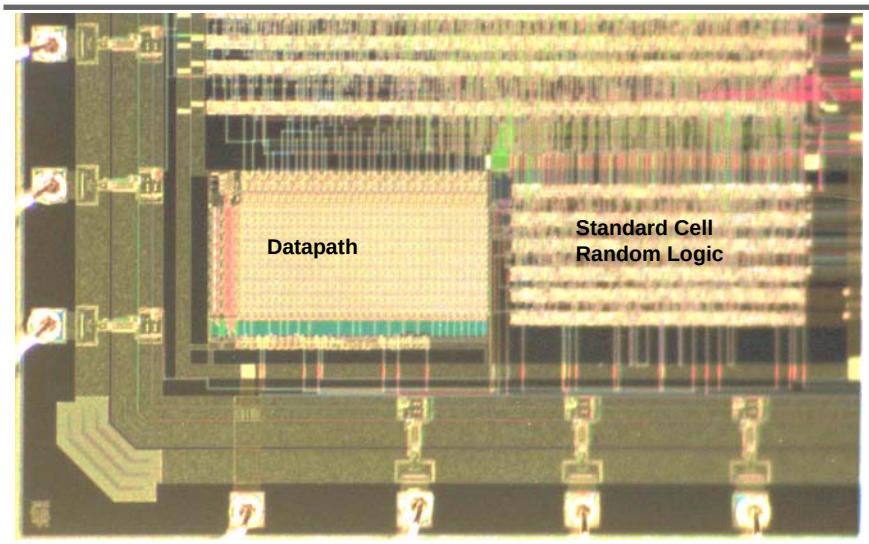
For data-path operators: structure is in bit-slices
Computer generated layout as function of wordlength
Compact, predictable IP





– 46

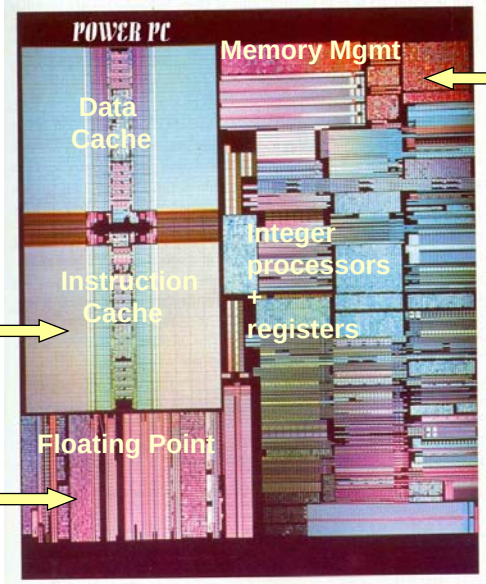
Standard Cell and Module



Courtesy: J. Van Campenhout RUG

– 47

Global Chip Layout (Power PC)



Standard Cell

Array Module Generator

Bit-Slice Module Generator

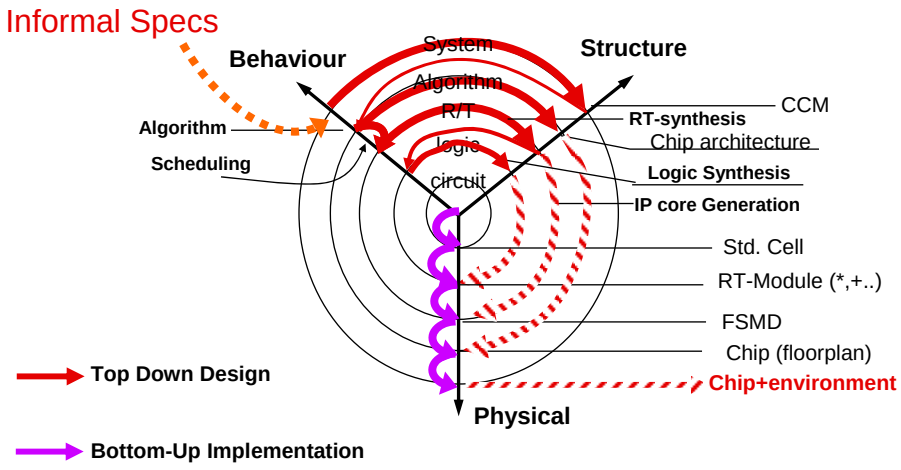
– 48

Level 5: Circuit Level

- **Logic expression to transistor schematic**
- **Transistor dimensioning (P,A,T)**
- **Layout strategy**
- **Logic cell layout**
- **Characterisation**
  - **VTC, levels, noise, power, speed...**

– 49

Global SoC design-flow

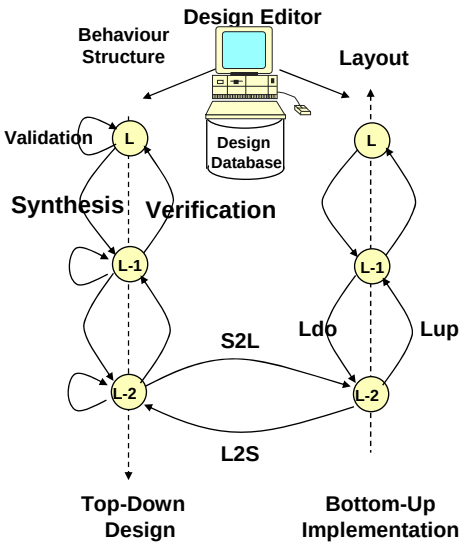


– 50

Outline

- Design Space and Design Flow
- Hierarchy and Abstraction Levels
- Design Flow for SoC's
- **CAD tools**
- Design Methodology

SoC CAD tools



Design Editor **-B, S, L**
Design Database **-All B,S,L views**
Validation
Synthesis
-High Level (scheduling)
-RT- and Logic Level
-A & D circuit synthesis
Verification
-Formal (proof of properties)
-Simulation
-Timing
S2L
-Place and Route
-Module generation
-Clock Network synthesis
L2S
-Structure Extraction
-Layout- and electrical rule check
Lup
-Block Place and Route
Ldo
-Floorplanning

Outline

- Design Space and Design Flow
- Hierarchy and Abstraction Levels
- Design Flow for SoC's
- CAD tools
- **Design Methodology**

Design Methodology

- **A set of *best practices* that lead to a time and cost effective implementation of a complex system**
- **How to avoid design iterations?**
- **So that system satisfies the constraints but not more**

"Sometimes we have to kill the engineer"

Best Practices

1. Spend most of the time at the top and document

2. Use interactive CAD tools with exploration capabilities.
   - A paintbrush makes no Van Gogh... The best CAD tool does not make a good engineer

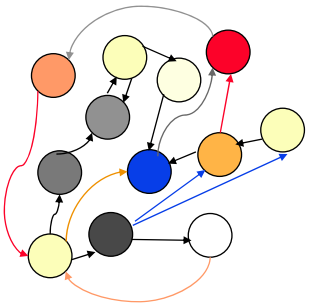
3. Keep it simple, stupid
   - What is simple, what is complex?

– 55

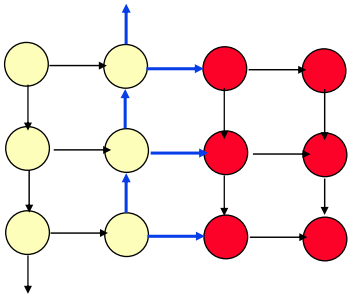
Complexity

Complexity increases with:

- # different components (not # of components!)
- # different interactions (interconnect, protocol, types)
- Lack of structure in interactions



Complex



Simple

– 56

KISS

- Use encapsulation
  - Both in software and hardware
- Synchronise actions and transactions
  - Locally synchronous, globally asynchronous
  - Localise heavy computation and traffic
- Keep communication simple and local
  - Most problems are in interfaces
  - Standardise interfacing
  - Develop by refinement, avoid loops
  - Minimize global traffic, localise traffic
- Separate function and communication
  - Key concept of reuse - plug-and-play
- Use structured interconnect - avoid spaghetti

– 57

Modularity

- Modularity means that a system is built from a *minimal* number of *re-usable parameterisable* entities or *modules*
- Document views, abstract to interface and behaviour
- Fights data explosion, encourages concurrent engineering

– 58

Conclusion Ch II

Different levels of abstraction:

- **System,**
- **Algorithm**
- **Register transfer**
- **Logical**
- **Circuits**

Different views at each level:

- **Behavioral**
- **Structural**
- **Physical**

– 59