High-level synthesis

Think about it…

- Standard-cell methodology and tools were developed for easy mapping of logic-level design into IC layout.
- BUT: example:

  32-bit: c = a + b.

  We do not write 32 Boolean expressions with up to 64 variables each to indicate this simple operation.

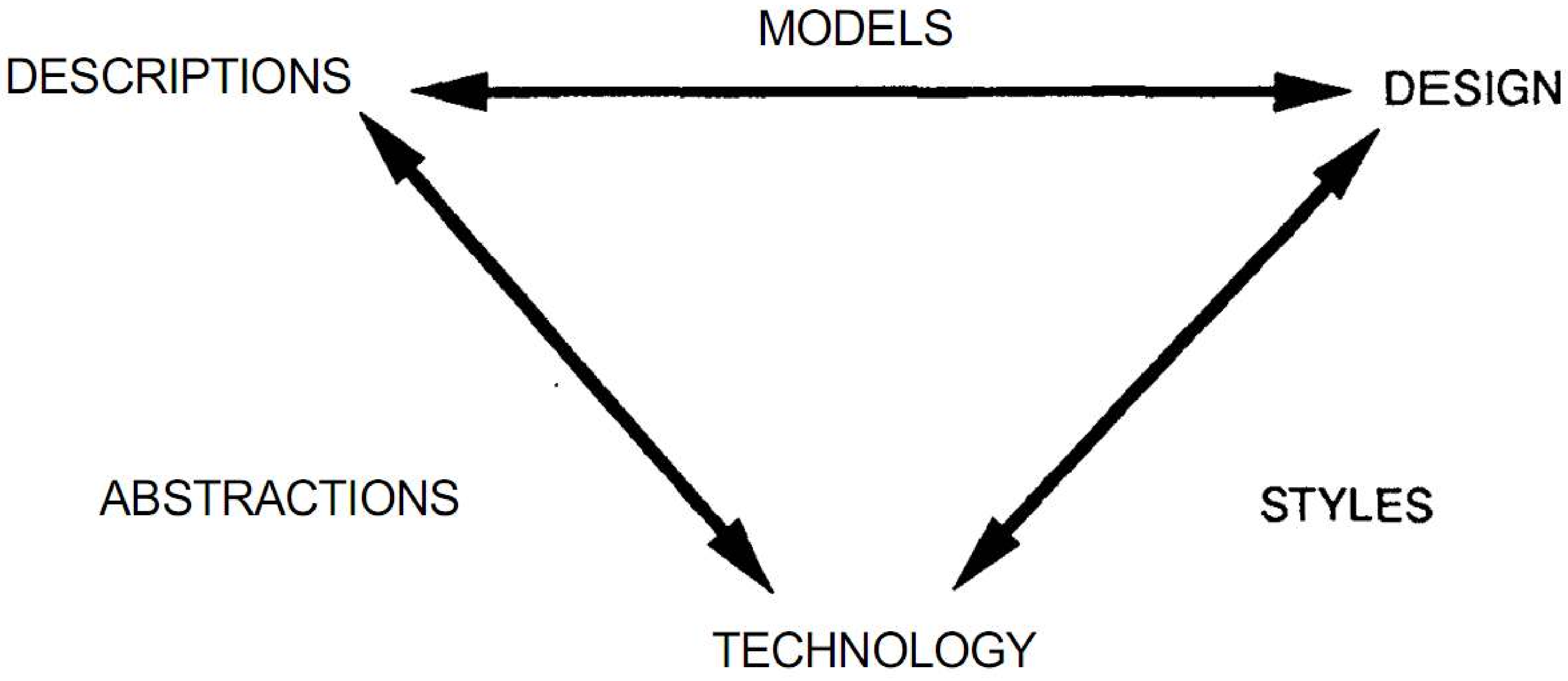
  Imagine having complex multi-chip systems described in terms of 1 million or more Boolean equations.

Main goal

- The main goal is to explore design auto-mation for synchronous digital systems at levels above the logic level in a practical environment.
- implement a system that is fast enough to allow the synthesis of large designs in a reasonable time
- high-level synthesis of synchronous digital systems

- input is a sequential specification of the function of a synchronous digital system, and possibly constraints such as the number and/or the type of the hardware modules to use, timing constraints, etc.
- assigns every operation in the specification to a control step (schedul-ing) and synthesizes the necessary hardware (allocation)
- The resulting design consists of a finite state machine (FSM) that implements the control and a netlist specifying the data path

Description-Design-Technology Dependence



Essential Issues in Synthesis

- Design Conceptualization
- Database Issues
- Technology Independence
- Design Learning
- Design Synthesis Complexity

Design Conceptualization

- Problem: design description is changing with design over time
- generate different design views for optimization or verification of different aspects of a design
- allow manual modification of synthesized design
- provide design quality metrics for exploration and design evaluation

Database Issues

- A central database stores all aspects of the design and generate different design views on demand:
  - Type-, format of a view
- initially built from the input description
- query the database for types of components, their functionality:
  - area, delay, performance, layout height, width and shape
- supply a component for any given functionality and constraint

Technology Independence

- On the layout level: technology independence can be achieved by laying dimensionless objects on a virtual grid
- Requires a change in the technology file containing spacing rules
- Requires sophisticated compaction algorithms with local optimization
- on the logic level: synthesizing design with generic gates and then mapping generic gates into library components by performing local transformations

Design Learning

- When a different ASIC library is used it is necessary to redesign higher level components to take advantage of the new library.
- learning technology-adaptation rules

Design Synthesis Complexity

- system level synthesis to translate a set of communicating processes into register-transfer components
- component synthesis to translate component descriptions into layouts

Theoretical background

- Turing-Church Thesis:

if an algorithm exists then there is an equivalent Turing machine, recursively-definable function, or applicable λ-function, for that algorithm.

Szintézis során használt algoritmusok

- THE SYNTHESIS IN-CORE MODEL
- DATA-FLOW ANALYSIS
- AS-FAST-AS-POSSIBLE SCHEDULING
- MODULE ASSIGNMENT
- PATH-BASED ALLOCATION
- Estimation
- Logic Minimization

THE SYNTHESIS IN-CORE MODEL

- Magas-szintű nyelvből gráf reprezentáció
- Vezérlés folyam gráf (CFG)
- Adatfolyam gráf (DFG)

CFG

- Irányított gráf, CFG=(N,P)
- Csúcsok halmaza (N):
  - Hozzárendelés,
  - összeadás,
  - logikai műveletek, stb…
- Élek halmaza (P):
  - Precedencia relációk,
  - egymást követő utasítások
- Összefüggő gráf

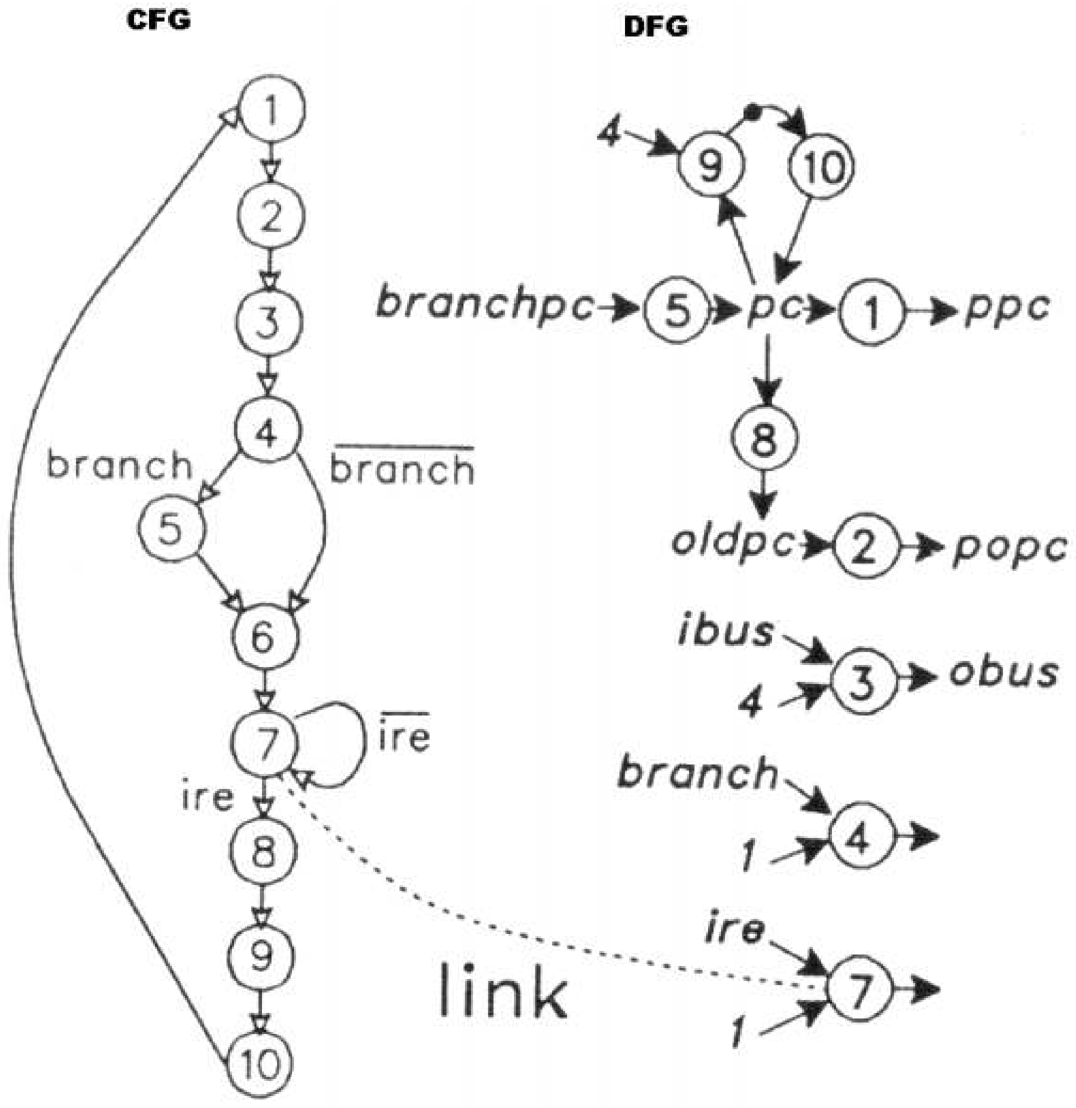
CFG

- Sorozat, szekvencia: egy él (n1,n2) eleme P-nek azt jelenti, hogy n2 következik n1 végrehajtása után
- Feltételes végrehajtás: egy művelet után egy másik művelet, ha a feltétel teljesül. A feltétel Boolean kifejezés, melynek értéke 1, akkor végrehajtódik, különben 0.
- Iteráció: Ciklusok, amelyek a folyamat iteratív viselkedését jelzik (loop).

DFG

- Irányított gráf, DFG=(N ∪ V,D)
- Csúcsok halmaza:
  - N: utasítások
  - V: változók
- Élek halmaza:
  - Adatkapcsolatok
- Nem feltétlenül összefüggő gráf

Példa



```
Entity prefetch is
      Port ( branchpc, ibus in bit32;
             branch, ire: in bit;
             ppc, popc, obus: out bit32);
end prefetch;

architecture behavior of prefetch is
begin
      process
      variable pc, oldpc: bit32:=0;
         begin
             ppc <= pc;                  --1
             popc <= oldpc;              --2
             obus <= ibus + 4;           --3
             if (branch = '1') then      --4
                    pc:= branchpc;       --5
             end if;                     --6
           wait until (ire='1');         --7
           oldpc:=pc;                    --8
           pc:=pc+4;                     --9, 10
         end process;
end behavior;
```

IBM Magas-szintű Szintézis

- The High-level IBM Synthesis system (HIS).
- HIS uses a design representation called the SSIM (Sequential Synthesis In-core Model)
- The SSIM represents control and data separately. Before high-level synthesis, the SSIM contains only the behavior of a design, i.e., a control-flow graph (CFG) and a data-flow graph (DFG).

- Scheduling consists basically in control synthesis and adds a control finite state machine (FSM) to the SSIM. The synthesized FSM is represented by its state transition graph.
- Allocation consists basically in data-path synthesis and adds a data path to the SSIM. The synthesized data-path is also represented as a graph (netlist).
- The complete SSIM resides in memory during high-level synthesis, thus achieving the necessary speeds for design space exploration and interactive design.

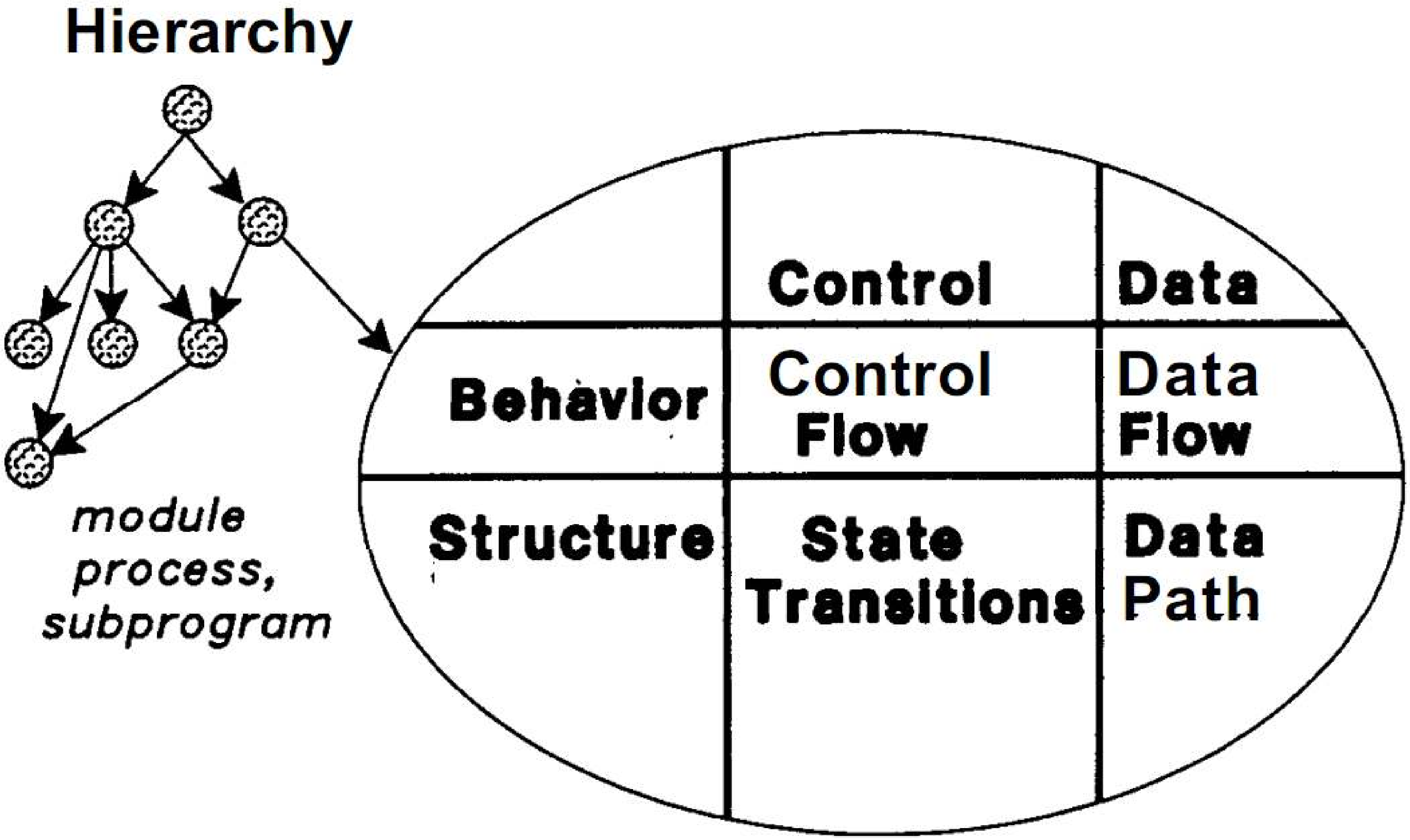


Figure 3. The Sequential Synthesis In–Core Model SSIM

DATA-FLOW ANALYSIS

- Data-flow analysis is a technique to determine the lifetime of values
- High-level synthesis often uses data-flow analysis to determine the intended behavior, for example, unfolding variables
- In HIS, lifetime information is used during allocation and scheduling

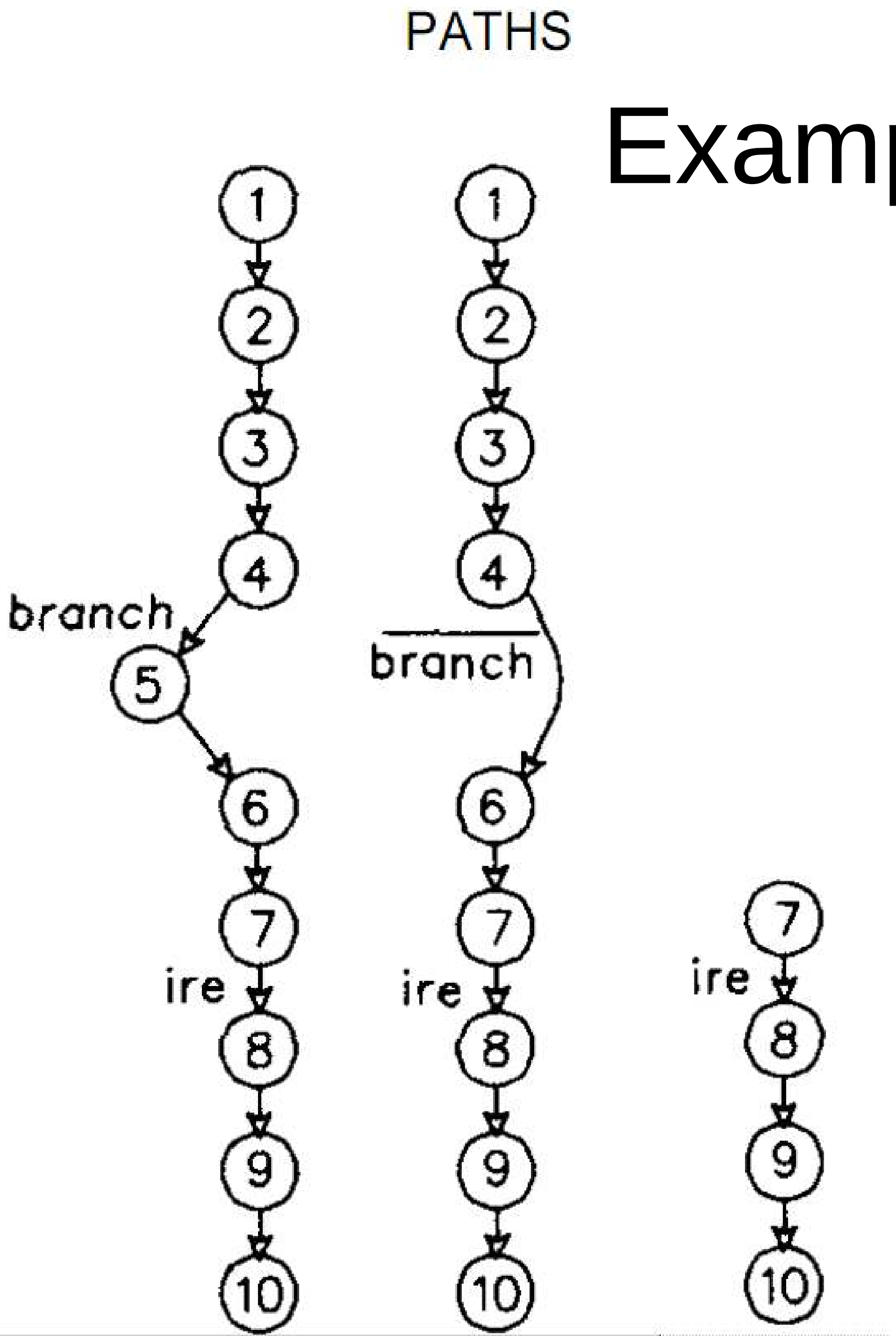
AS-FAST-AS-POSSIBLE SCHEDULING

- HIS uses As-Fast-As-Possible (AFAP) scheduling for each possible path in the CFG.
- A path in the CFG is defined by conditional branches; thus one path represents one possible execution determined by the conditions on the conditional branches.
- Although the number of paths may grow exponentially with the number of nodes
- the number of paths represents the number of different functions in the circuit
- Optimizing the length of all paths (length in number of control steps), means minimizing the number of cycles in each instruction.

AS-FAST-AS-POSSIBLE SCHEDULING

- Paths are computed by performing a depth-first traversal of the acyclic CFG.
- The CFG is made acyclic by removing the feedback edges in loops
- Paths are computed starting at a given first operation

Example



Constraints

- An implementation (and thus the schedule) is usually restricted by so called hardware constraints.
  - I/O ports can either transmit or receive only one distinct value per control step
  - registers can be written only once per control step
- Constraints explicitly specified often indicate area and delay characteristics of the desired implementation:
  - the maximum clock cycle
  - the maximum number and type of the functional units to be used

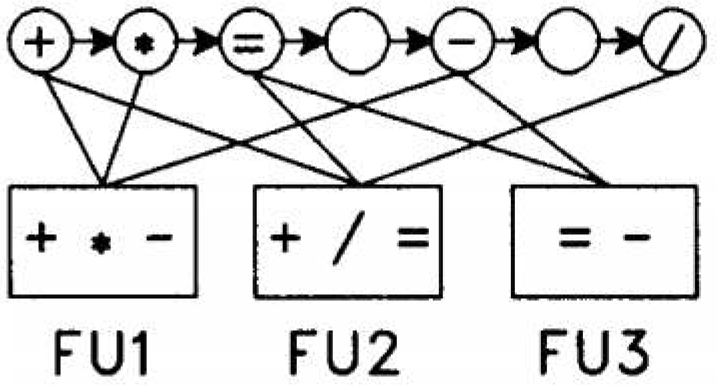
MODULE ASSIGNMENT

- The module assignment problem arises when operations can be implemented by more than one type of functional unit (FU).
- Module assignment selects the FU to implement each operation.

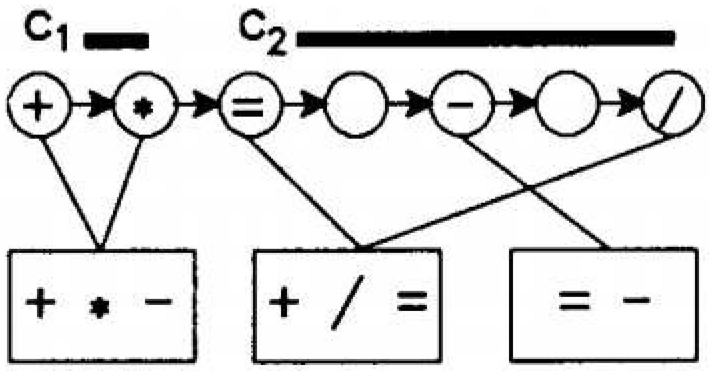
Example

```
T1    := IN1 + IN2;
T2     := IN3 * IN4;
T3     := (T2 = 15);
OUT1 := T3;
T4     := T1 - 1;
T1    := T4;
OUT2 := T1 / 4;
```

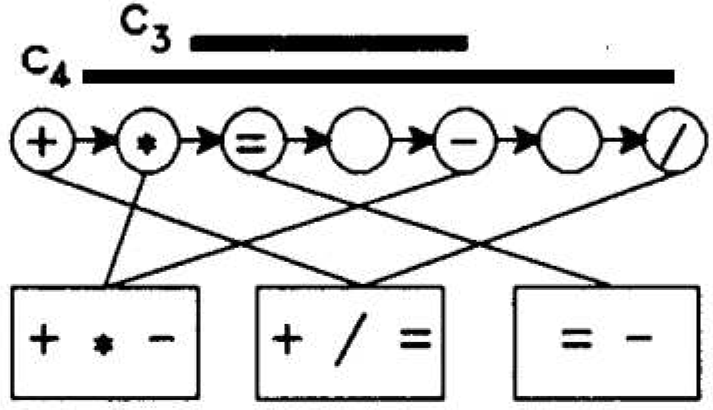
(a)



(b)



(c)



(d)

PATH-BASED ALLOCATION

- clique covering (or coloring) based allocation
- two separate steps:
  - a complete, functional initial data path is generated
  - then optimized

Initial Data-Path Allocation

- Allocating functional units
- Allocating registers
- Defining the interconnections (multiplexers) between functional units and registers
- Deriving the control signals:
  - load enable for registers
  - operation selection for functional units
  - selection for multiplexers

Data-Path Optimizations

- Optimizations in the data path comprise register, functional unit and multiplexer merging
- First a global conflict graph is generated
- The conflict graph is then colored to determine the smallest amount of hardware

Estimation

- For both constraint generation and data path optimization, it is necessary to estimate sizes and delays.
- it is extremely difficult at a high level.
- The models used are similar to typical logic synthesis delay models.

Logic Minimization

- Logic minimization within HIS is performed to optimize control signals on the fly

Output generation

- Output language generation is straightforward.
- Any language capable of representing netlists or RT-level hardware can be chosen.