High-Level Synthesis

1. Basic definition
2. A typical HLS process
3. Scheduling techniques
4. Allocation and binding techniques
5. Advanced issues

Introduction

Definition: HLS generates register-transfer level designs from behavioral specifications, in a automatic manner.

- Input:
  - The behavioral specification.
  - Design constraints (cost, performance, power consumption, pin-count, testability, etc.).
  - An optimization function.
  - A module library representing the available components at RTL.
- Output:
  - RTL implementation structure (netlist).
  - Controller (captured usually as a symbolic FSM).
  - Other attributes, such as geometrical information.

- Goal: to generate a RTL design that implements the specified behavior while satisfying the design constraints and optimizing the given cost function.

A Typical HLS Process

1. Behavioral specification:

- Which language to use?

  Procedural languages

  Functional languages

  Graphics notations

- Explicit parallelism?

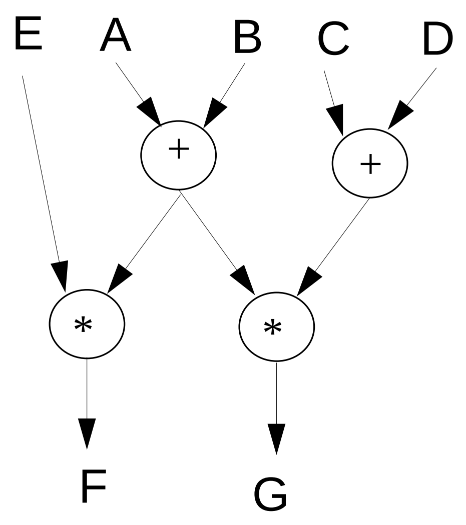
```
PROCEDURE Test;
 VAR

A,B,C,D,E,F,G:integer;
 BEGIN
  Read(A,B,C,D,E);
  F := E*(A+B);
  G := (A+B)*(C+D);
  ...
 END;
```

Input behavioral specification

2. Dataflow analysis:

- Parallelism extraction.
- Eliminating high-level language constructs.
- Loop unrolling.
- Program transformation.
- Common subexpression detection.

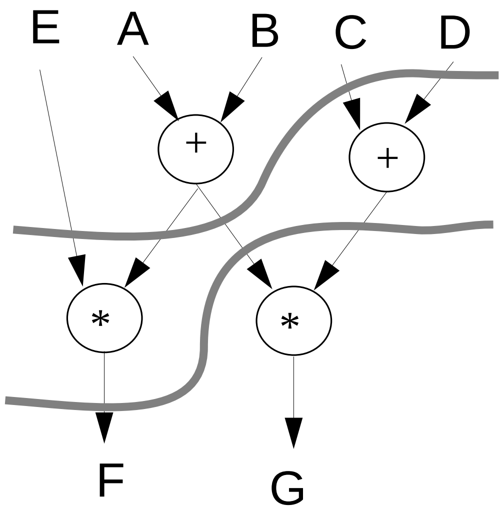


Dataflow description

A Typical HLS Process (Cont'd)

3. Operation scheduling:

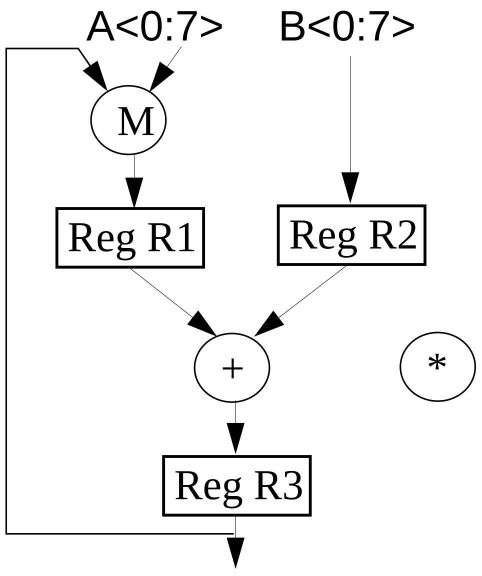
- Performance/cost trade-offs.
- Performance measure.
- Clocking strategy.



Scheduled dataflow description

4. Data-path allocation:

- Operator selection.
- Register/memory allocation.
- Interconnection generation.
- Hardware minimization.

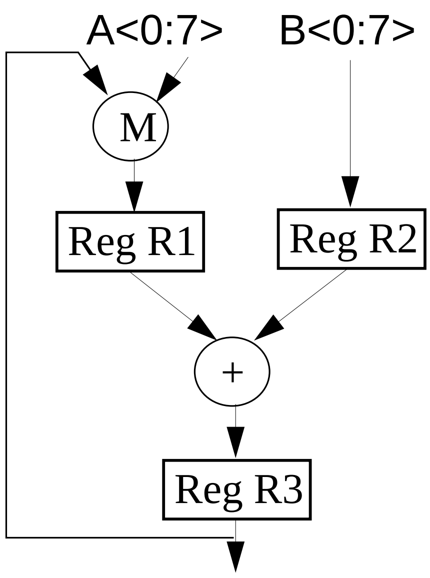


Partial data-path

A Typical HLS Process (Cont’d)

5. Control allocation:

- Selection of control style (PLA, micro-code, random logic, etc.).
- Controller generation.



Controller description:

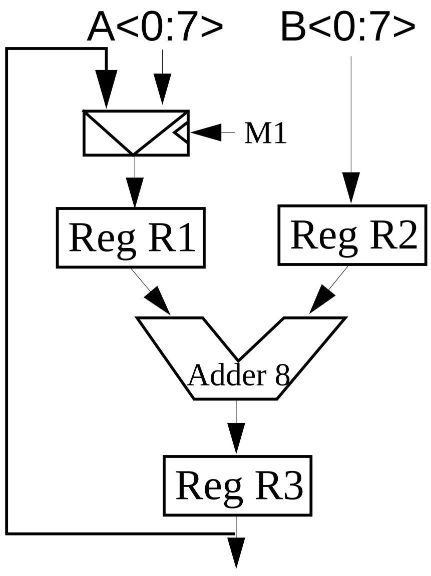
```
S1: M1=1, Load R1 next S2;
S2: Load R2 next S3;
S3: Add, Load R3 next S4;
S4: M1=0, Load R1 next...
```

RTL structure with controller description

A Typical HLS Process (Cont'd)

6. Module binding and controller implementation:

- Selection of physical modules.
- Specification of module parameters and constraints.
- Controller implementation.



Controller ROM:

```
0000 : 11000000 0001
0001 : 00100000 0010
0010 : 00011000 0011
0011 : 01000000 0100
```

Final results

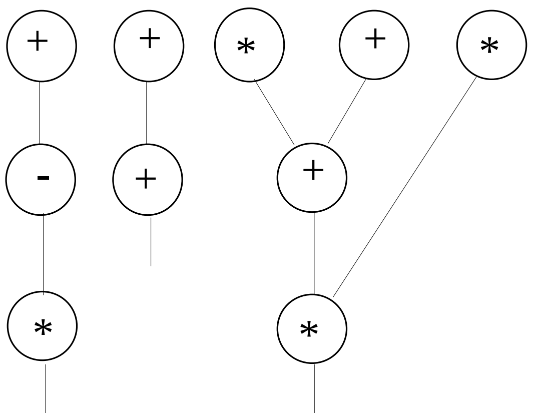
The Basic Issues

- **Scheduling** — Assignment of each operation to a time slot corresponding to a clock cycle or time interval.
- **Resource Allocation** — Selection of the types of hardware components and the number for each type to be included in the final implementation.
- **Module Binding** — Assignment of operation to the allocated hardware components.
- **Controller Synthesis** — Design of control style and clocking scheme.
- **Compilation** of the input specification language to the internal representation must be done.
- **Parallelism Extraction** — To extract the inherent parallelism of the original solution, which is usually done with data flow analysis techniques.
- **Operation Decomposition** — Implementation of complex operations in the behavioral specification.
- ...

The Scheduling Problem

- Resource-constrained (RC) scheduling:
  - Given a set *O* of operations with a partial ordering which determines the precedence relations, a set *K* of functional unit types, a type function, τ: $O \rightarrow K$, to map the operations into the functional unit types, and resource constraints m_k for each functional unit type.
  - Find a (optimal) schedule for the set of operations that obeys the partial ordering and utilizes only the available functional units.

Ex.



```
a := i1 + i2;
o1 := (a - i3) * 3;
o2 := i4 + i5 + i6;
d := i7 * i8;
g := d + i9 + i10;
o3 := i11 * 7 * g;
```

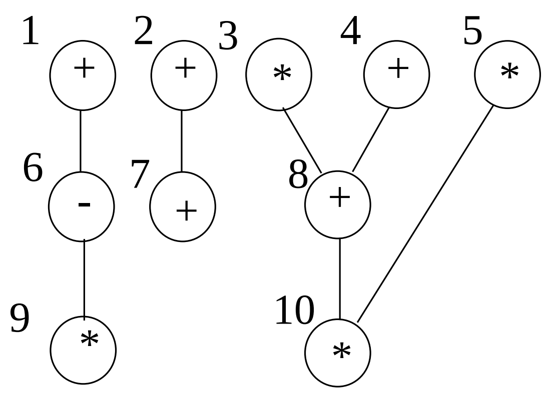
1 adder, 1 multiplier

τ: +,- $\rightarrow$ Adder
  * $\rightarrow$ Multiplier

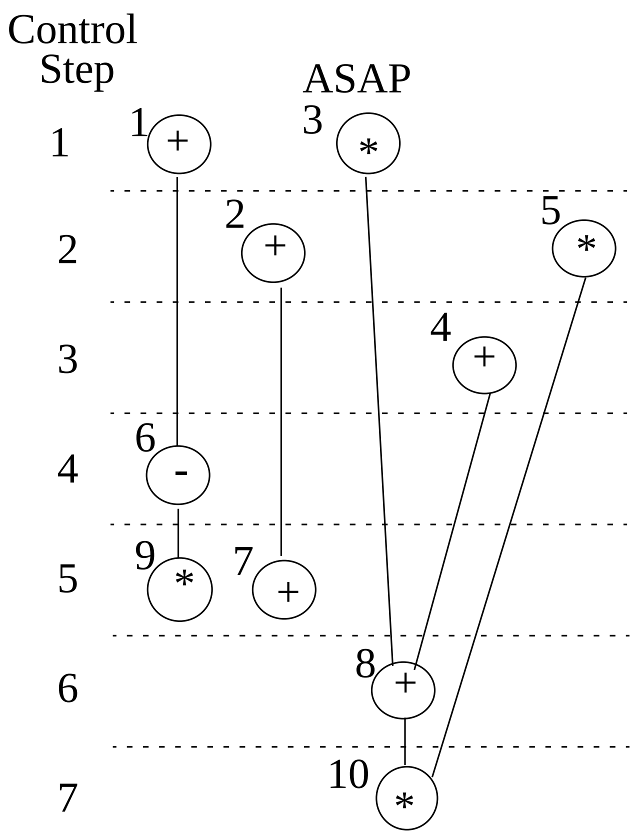
- Time-constrained (TC) scheduling:

RC Scheduling Techniques

- ASAP: As soon as possible
  - Sort the operations topologically according to their data /control flow;
  - Schedule operations in the sorted order by placing them in the earliest possible control step.



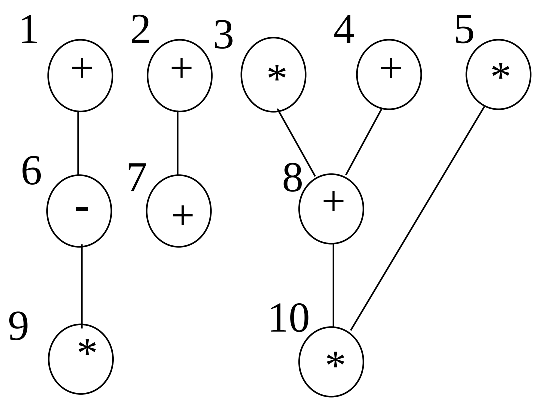
(a) Sorted DFG



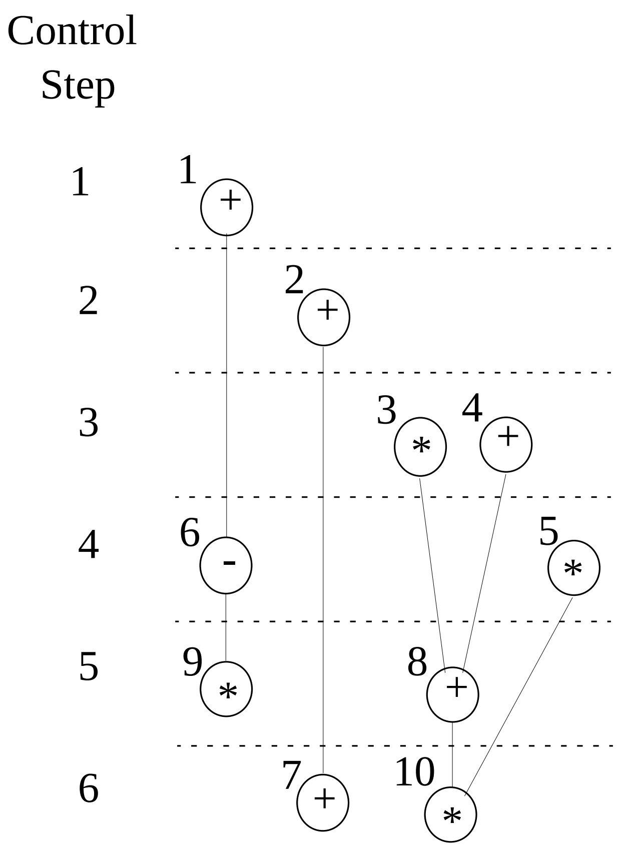
(b) ASAP schedule

RC Scheduling Techniques (Cont'd)

- ALAP: As late as possible
  - Sort the operations topologically according to their data / control flow;
  - Schedule operations in the reversed order by placing them in the latest possible control step.



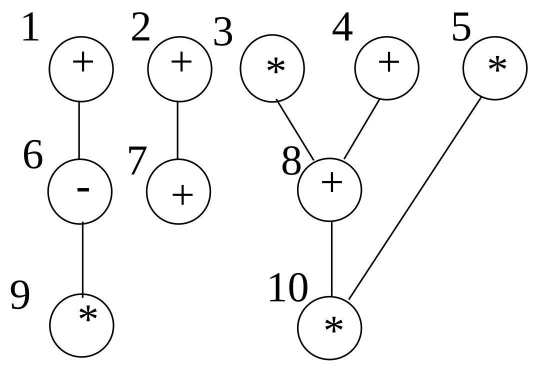
(a) Sorted DFG



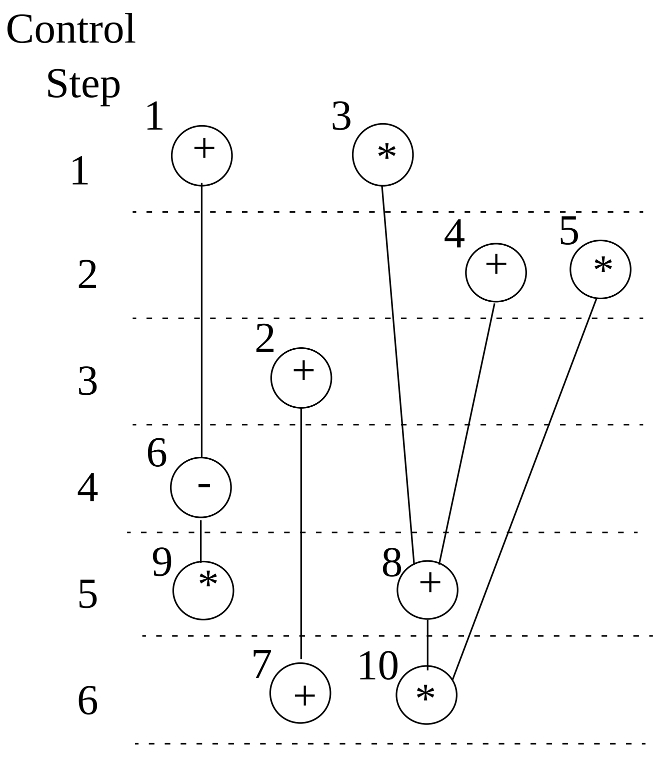
(b) ALAP schedule

RC Scheduling Techniques (Cont'd)

- List Scheduling
  - For each control step, the operations that are available to be scheduled are kept in a list;
  - The list is ordered by some priority function:
    1. The length of path from the operation to the end of the block;
    2. Mobility: the number of control steps from the earliest to the latest feasible control step.
  - Each operation on the list is scheduled one by one if the resources it needs are free; otherwise it is deferred to the next control step.



(a) DFG

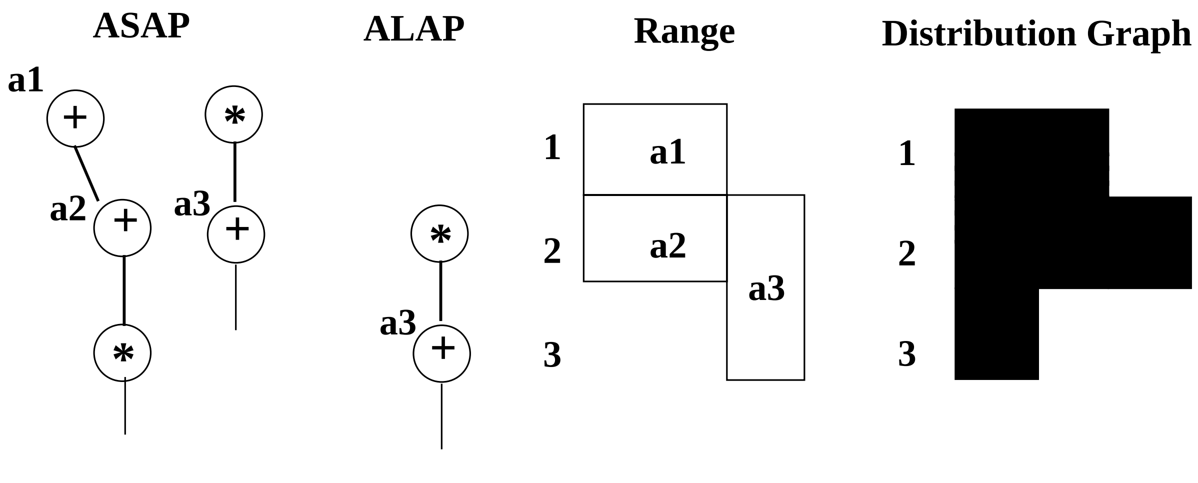


(b) List schedule

TC Scheduling Techniques

- Force-Directed Scheduling: The basic idea is to balance the concurrency of operations.
  - ASAP and ALAP schedules are calculated to derive the time frames for all operations.
  - For each type of operations, a distribution graph is built to denote the possible control steps for each operation. If an operation could be done in *k* steps, then *1*/*k* is added to each of these *k* steps.

An example:



- The algorithm tries to balance the distribution graph by calculate the force of each operation-to-control step assignment and select the smallest force:

$$Force_{(\sigma(o_i)=s_j)} = DG(s_j) - \frac{1}{\Delta T(o_i)} \cdot \sum_{s=\sigma_{ASAP}(o_i)}^{\sigma_{ALAP}(o_i)} DG(s)$$

Classification of Scheduling Approaches

- Constructive scheduling - one operation is assigned to one control step at a time and this process is iterated from control step (operation) to control step (operation).
  - ASAP
  - ALAP
  - List scheduling

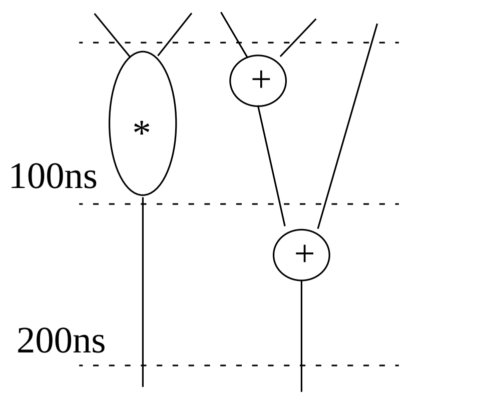
- Global scheduling - All control step and all operations are considered simultaneously when operations are assigned to control steps.
  - Force-directed scheduling
  - Neural net scheduling
  - Integer Linear Programming algorithms

- Transformational scheduling - starting from an initial schedule, a final schedule is obtained by successively transformations.

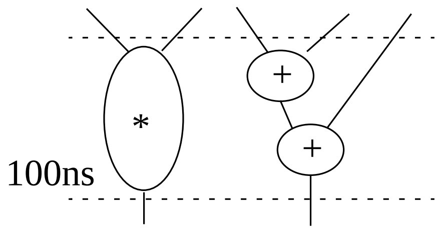
Advanced Scheduling Issues

- Control construct consideration.
  - conditional branches
  - loops

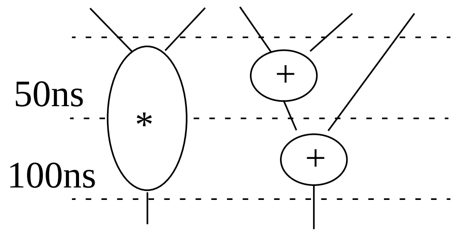
- Chaining and multicycling.



(a) No chaining or multicycling



(b) Two chained additions

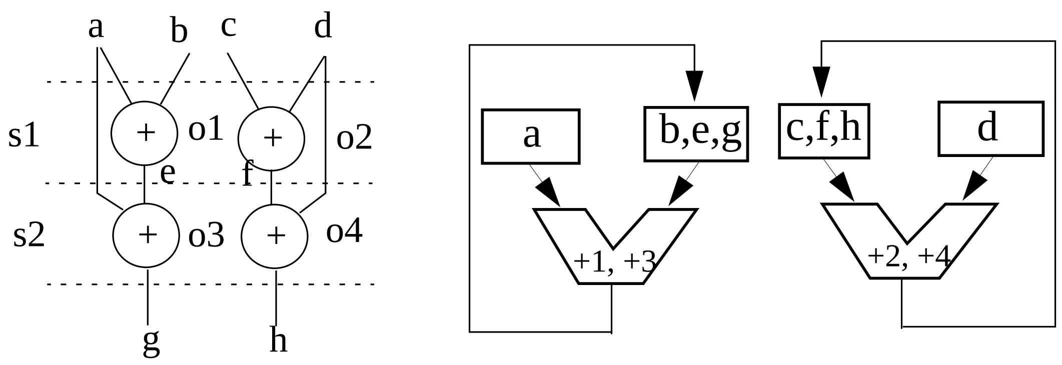


(c) A multicycle multiplication

- Scheduling with local timing constraints.

Allocation and Binding

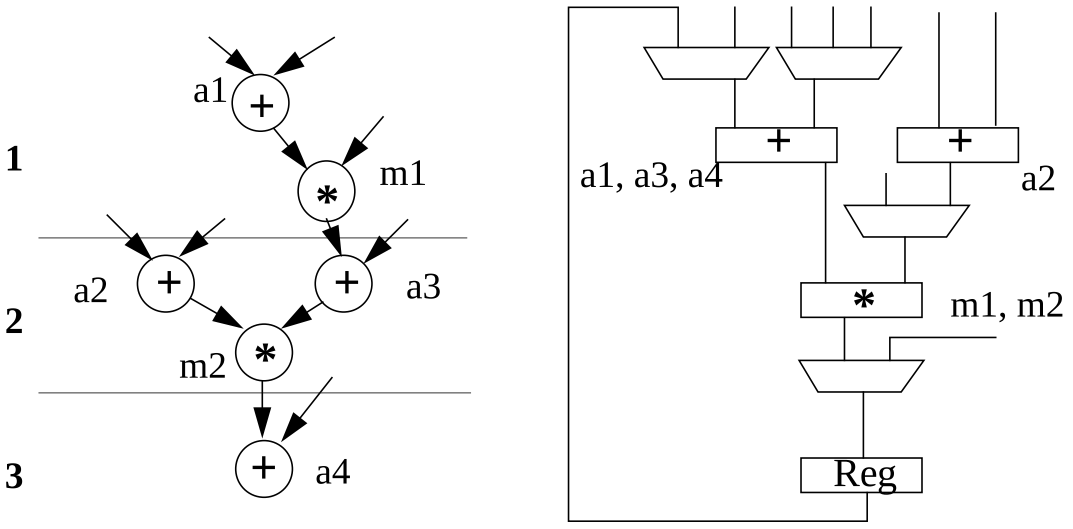
- **Allocation** (unit selection) — Determination of the type and number of resources required:
  - Number and types of functional units
  - Number and types of storage elements
  - Number and types of busses
- **Binding** — Assignment to resource instances:
  - Operations to functional unit instances
  - Values to be stored to instances of storage elements
  - Data transfers to bus instances



- Optimization goal
  - Minimize total cost of functional units, register, bus driver, and multiplexor
  - Minimize total interconnection length
  - Constraint on critical path delay

Approaches to Allocation/Binding

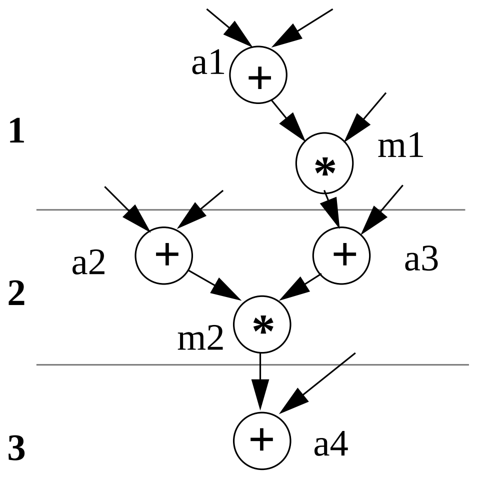
- **Constructive** — start with an empty datapath and add functional, storage and interconnects as necessary.
  - Greedy algorithms — perform allocation for one control step at a time.

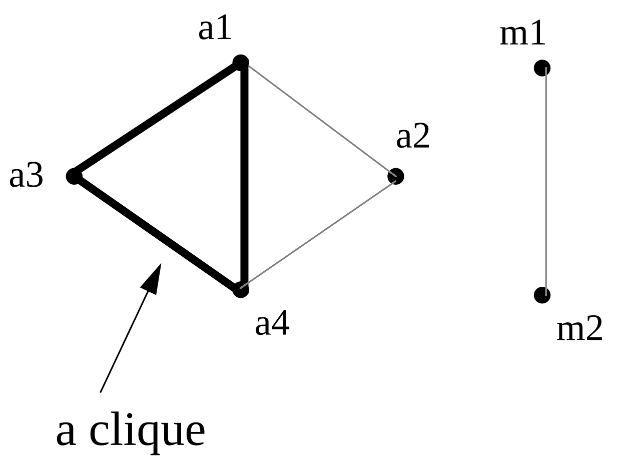


  - Rule-based — used to select type and numbers of function units, especially prior to scheduling.
- **Graph-theoretical formulations** — sub-tasks are mapped into well-defined problems in graph theory.
  - Clique partitioning.
  - Left-edge algorithm.
  - Graph coloring.
- **Transformational allocation**

Clique Partitioning

- Let *G* = (*V*, *E*) be an undirected graph with a set *V* of vertices and a set *E* of edges.
- A *clique* is a set of vertices that form a complete subgraph of *G*.
- The problem of partitioning a graph into a minimal number of cliques such that each vertex belongs to exactly one clique is called *clique partitioning*.
- Formulation of functional unit allocation as a clique partitioning problem:
  - Each vertex represents an operation.
  - An edge connects two vertices iff:
    1. the two operations are scheduled into different control steps, and
    2. there exists a functional unit that is capable of carrying out both operations.





Clique Partitioning (Cont'd)

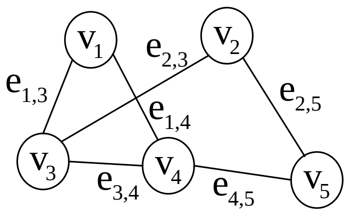
- Formulation of storage allocation as a clique partitioning problem:
  - Each value needed to be stored is mapped to a vertex.
  - Two vertices are connected iff the life-time of the two values do not intersect.

☞ The clique partitioning problem is NP-complete.

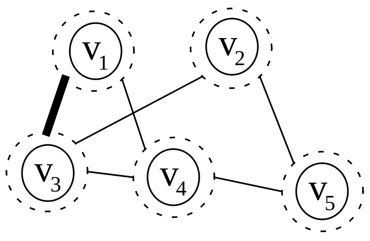
☞ Efficient heuristics have been developed; e.g., Tseng used a polynomial time algorithm which generates very good results.

Tseng’s Algorithm

- A super-graph is derived from the original graph.
- Find two connected super-nodes such that they have the maximum number of common neighbors.
- Merge the two nodes and repeated from the first step, until no more merger can be carried out.

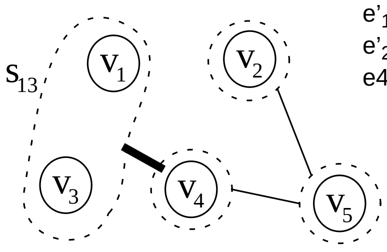


(a)



Edge	Common neighbors
$e'_{1,3}$	1
$e'_{1,4}$	1
$e'_{2,3}$	0
$e'_{2,5}$	0
$e'_{3,4}$	1
$e'_{4,5}$	0

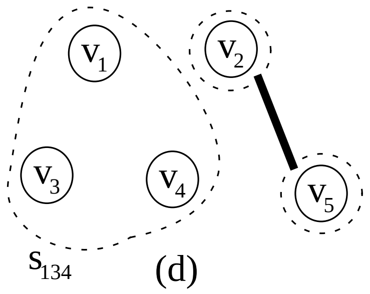
(b)



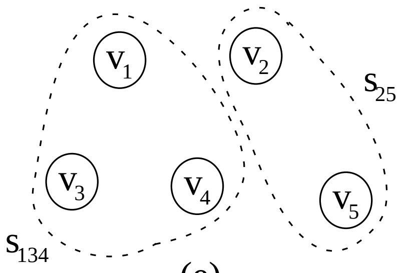
Edge	Common neighbors
$e'_{13,4}$	0
$e'_{2,5}$	0
e4,5	0

(c)

Edge	Common neighbors
$e'_{2,5}$	0



(d)



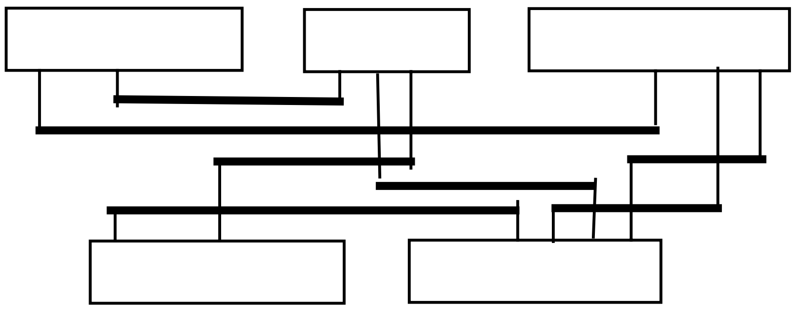
(e)

```
Cliques:
```
$S_{134}=(V_1,V_3,V_4)$

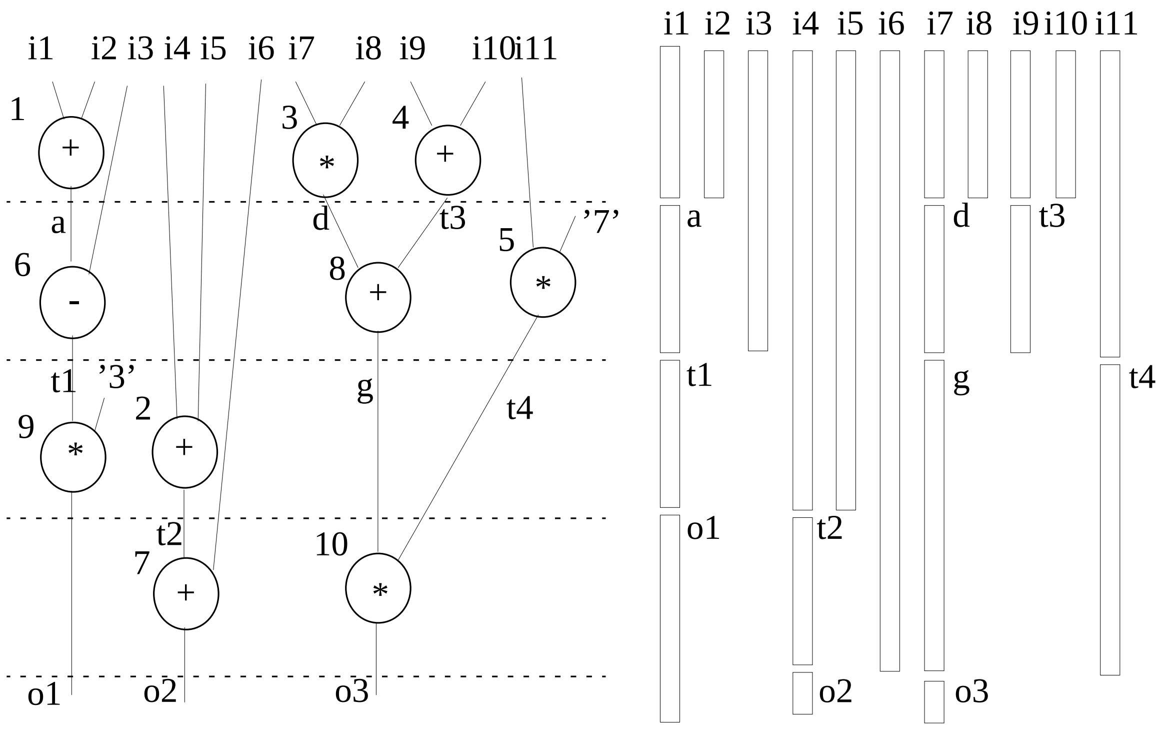
$S_{25}=(V_2,V_5)$

Left-Edge (LE) Algorithm

- The LE algorithm is used in channel routing to minimize the number of tracks used to connect points.



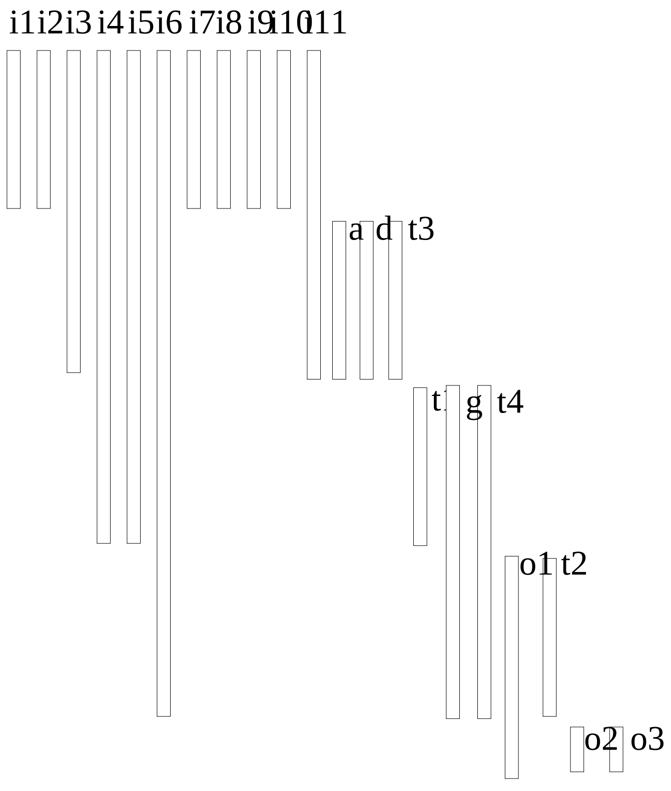
- The register allocation problem can be solved by the LE algorithm by mapping the birth time of a value to the left edge, and the death time of a value to the right edge of a wire.



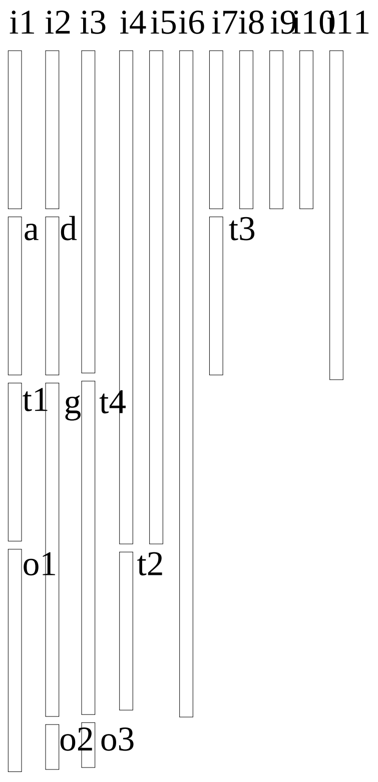
Variable life-times

Left-Edge (LE) Algorithm (Cont'd)

- The algorithm works as follows:
  - The values are sorted in increasing order of their birth time.
  - The first value is assigned to the first register.
  - The list is then scanned for the next value whose birth time is larger than or equal to the death time of the previous value.
  - This value is assigned to the current register.
  - The list is scanned until no more value can shared the same register. A new register will then be introduced.



a) sorted list of variables



b) assignment of variables into registers

Left-Edge (LE) Algorithm (Cont'd)

- The algorithm guarantees to allocate the minimum number of registers, but has two disadvantages:
  - Not all life-time table might be interpreted as intersecting intervals on a line.

        loop

        conditional branches
  - The assignment is neither unique nor necessarily optimal (in terms of minimal number of multiplexors, for example).

Advanced Issues of HLS

- Target architecture consideration, e.g. pipelining.
- General library organization.
  - Component hierarchy.
  - Many-to-many mapping between operations and physical components.
  - Multiple technology components.
- Domain-specific synthesis strategies.
  - Control-dominated applications.
  - Timing-driven optimization.
- Re-use of previous designs.
- Synthesis with commercially available sub-systems, IP-based synthesis.
- HLS with testability consideration.
- HLS with power consideration.

Advanced Issues of HLS

- Target architecture consideration.
  - Multiplexed data-path:

    Operations are mapped to combinational units.

    Storage is provided by distributed registers.

    Interconnect is established by multiplexors/nets.

    A single system clock controls all registers.
  - Bidirectional bus architecture:

    Functional units have storage capabilities.

    Register files are often supported.

    Interconnect hardware comprises bidirectional busses, multiplexors, drivers, and nets.
  - Pipelined data paths.