High-Level Synthesis for Reconfigurable Systems

HV: Phạm Viết Nguyên 12073127
Trần Thanh Bình 12073117

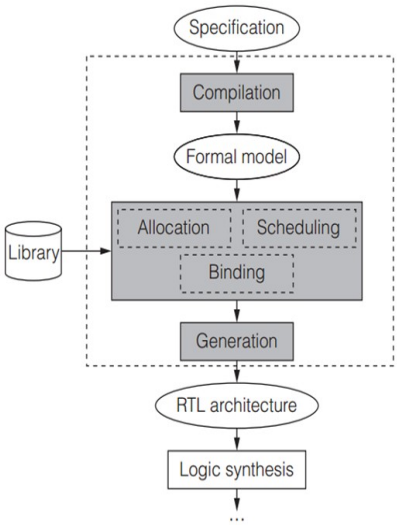
GV: Ts.Trần Ngọc Thịnh

Agenda

- **Modeling**
  - Dataflow Graphs
  - Sequencing Graph
  - Finite State Machine with Data path
  - Fundamental differences in HLS for reconfigurable computing
  - Temporal Partitioning
- **Temporal Partitioning Algorithms**
  - Unconstrained Scheduling
  - The List Scheduling Approach
  - Integer Linear Programming
  - Network Flow
  - Spectral Methods

Agenda

- **Modeling**
  1. Dataflow graphs
  2. Sequencing graphs
  3. Finite State Machine with data-path
- **High-level synthesis and temporal partitioning**

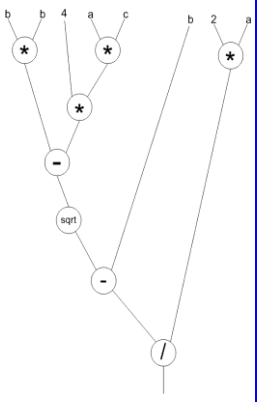


Modeling

- **High-level descriptions:**
  - Modeling is a key aspect in the design of a system
  - Models used must be powerful enough
    - Capture all user's need
    - Easy to understand and manipulate
- **Several powerful models exists:**
  - FSM,
  - State Charts,
  - DFG,
  - Petri Nets,
  - …
- **Focus in this course:**
  - Dataflow graphs,
  - Sequencing graphs,
  - Finite State Machine with Datapath (FSMD)

Dataflow Graph

- **DFG:**
  - Means to describe a computing task in a streaming mode.
  - Operators: nodes
  - Operands: Inputs of the nodes
  - Node's output can be used as input to other nodes
    - Data dependency in the graph.
- **Given a set of tasks $\{T_1,\ldots,T_k\}$, a DFG is a DAG $G(V,E)$, where**
  - V (= T): the set of nodes representing operators and
  - E: the set of edges representing data dependency between tasks.

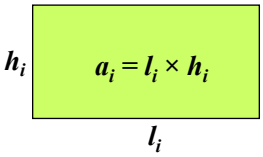


DFG for the quadratic root computation using:

$$x = \frac{(b^2 - 4ac)^{\frac{1}{2}} - b}{2a}$$

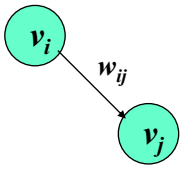
Definitions

node $v_i \in V$



implementation H_{vi}

- The latency t_i of v_i :
  - The time it takes to compute the function of v_i using H_{vi}



$$e = e_{ij} = (v_i, v_j) \in E$$

- Weight w_{ij} of $e_{ij} \in E$:
  - width of bus connecting two components H_{vi} and H_{vj}
- Latency t_{ij} of e_{ij}:
  - the time needed to transmit data from H_{vi} to H_{vj}

DFG

- Any high-level program can be compiled into a DFG, provided that:
  - No loops
  - No branching instructions.

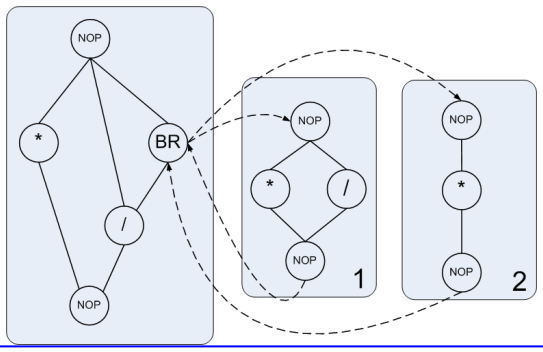
- **Therefore:**
  - DFG is used for parts of a program.
  - Extend to other models:
    - Sequencing graph
    - FSMD

Sequencing Graph

- **Sequencing Graph:**
  - Hierarchical DFG with two different types of nodes:
  1. Operation nodes: normal ”task nodes” in a DFG
  2. Link nodes or branching nodes: point to another sequencing graph in a lower level of the hierarchy.

Sequencing Graph

- **Example:**
  - According to the conditions that node BR evaluates, one of the two sub-sequencing graphs (1 or 2) can be activated.
  - Loop description:
    - Body of the loop is described in only one sub-sequencing graph: BR evaluates the loop exit condition



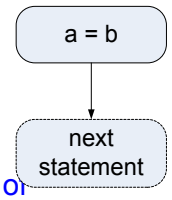
Finite State Machine with Datapath (FSMD)

- **FSMD:**
  - Extension of DFG with an FSM
  - is a 6-tuple <S, I, O, F, H, s_0>:
    - $S = \{s_0, \ldots, s_l\}$: states,
    - $I = \{i_0, \ldots, i_m\}$: inputs,
    - $O = \{o_0, \ldots, o_n\}$: outputs,
    - $F: S \times I \times O \rightarrow S$: a transition function that maps a tuple (s_i, i_j, o_k) to a state,
    - $H: S \rightarrow O$: an action function that maps the current state to output,
    - s_0: an initial state.
- **FSMD vs. FSM**
  - FSMD operates on arbitrary complex data types
    - Not just Boolean vars
  - Transition may include arithmetic operations

FSMD

- **Modeling with FSMD:**
  - The transformation of a program into a FSMD is done by
    - Transforming the statements of the program into FSMD states.
    - The statements are first classified in three categories:
      1. assignment statements,
      2. branch statements and
      3. loop statements

```
a = b;
next statement
```



1. Assignment statement

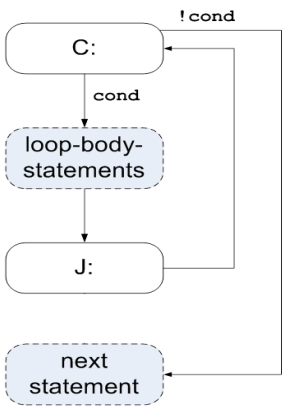
- a single state is created that executes the assignment action.
- An arc connecting the state with the state of the next statement is created.

FSMD: Loop

2. Loop statement:

- A **condition state** C and a **join state** J, both with no action are created.
- an **arc** is added:
  - **label**: the loop condition
  - **connects** C to the state of the first statement in the loop body.
- another **arc** is added:
  - **label**: complement of the loop condition
  - **connects** C to the first statement after the loop body.
- an **edge** is added:
  - connects: the state of the last statement in the loop to the join state
- another **edge** is added:
  - **connects**: join state back to the conditional state.

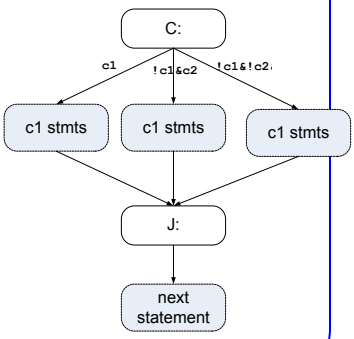
```
while(cond){
Loop-body-statements
}
next statement
```



FSMD: Branch

3. **Branch statement:**
   - a **condition state** C and a **join state** J, both with no action are created.
   - an **arc** is added:
     - **label**: the first branch condition
     - **connects** C to the state of the first statement of the branch.
   - another **arc** is added:
     - **label**: complement of the first condition ANDed with the second branch condition
     - **connects** C to the first statement of the branch.
   - ….
   - Each state corresponding to the last statement in a branch is connected to the join state.
   - Join state is connected to the state corresponding to the first statement after the branch.

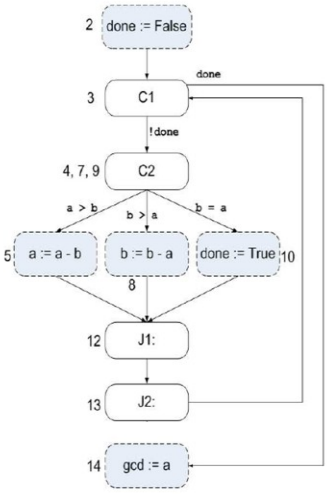
```
if(c1)
     c1-stmts
else if(c2)
     c1-stmts
else
     others stmts
next statement
```



FSMD: Example

Algorithm 6 The greatest common divisor sequential algorithm

```
 1: variable a, b, gcd: integer;
 2: done := FALSE;
 3: while (!done) do
 4:    if (a > b) then
 5:       a := a - b;
 6:    else
 7:       if (b > a) then
 8:          b := b - a;
 9:       else
10:          done = TRUE;
11:       end if
12:    end if
13: end while
14: gcd := a;
```



High-Level Synthesis

- **High-Level Synthesis (Architectural Synthesis):**
  - Transforming an abstract model of circuit behavior into a data path and a control unit.
- **Steps:**
  1. **Allocation:** defines the resource types required by the design, and for each type the number of instances.
  2. **Binding:** maps each operation to an instance of a given resource.
  3. **Scheduling:** sets the temporal assignment of resources to operators.
     - Decides which operator owns the resource at a given time

Allocation

- **Allocation (Formal Definition):**
  - For a given specification with a set of operators or tasks

$$T = \{t_1, t_2, \cdots, t_n\}$$

to be implemented on a set of resource types

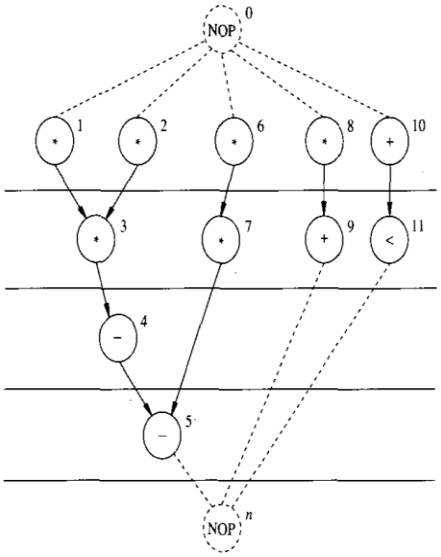
$$R = \{r_1, r_2, \cdots, r_t\},$$

allocation is a function $\alpha : R \rightarrow Z^+$, where

$\alpha(r) = z_i$ is the number of available instances of resource type r_i

Allocation

- **Example:**
  - *T = {*, *, *, -, -, *, *, *, +, +, <}*
  - *R = {ALU, MUL} = {1,2}*
    - *ALU: add, sub, compare*
  - *α(1) = 5*
  - *α(2) = 6*



Binding

- **Binding (Formal Definition):**
  - For

$$T = \{t_1, t_2, \cdots, t_n\}$$

and

$$R = \{r_1, r_2, \cdots, r_t\},$$

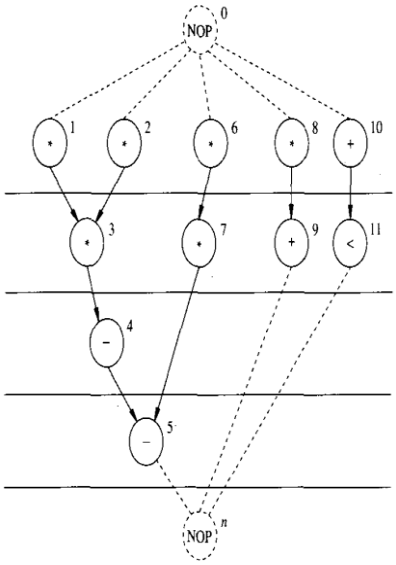
binding is a function $\beta : T \rightarrow R \times Z^+$, where

$\beta(t_i) = (r_i, b_i)$, $(1 \le b_i \le \alpha(r_i))$ is the instance of the resource type r_i on which t_i is mapped to.

Binding

- **Example:**
  - $T = \{*, *, *, -, -, *, *, *, +, +, <\}$
  - $R = \{ALU, MUL\} = \{1,2\}$
    - *ALU: add, sub, compare*

$\beta(t_1) = (2,1)$
$\beta(t_2) = (2,2)$
$\beta(t_3) = (2,3)$
$\beta(t_4) = (1,1)$
$\beta(t_5) = (1,2)$
$\beta(t_6) = (2,4)$
$\beta(t_7) = (2,5)$
$\beta(t_8) = (2,6)$
…
$\beta(t_{11}) = (1,5)$



Scheduling

- **Scheduling (Formal Definition):**
  - For

$$T = \{t_1, t_2, \cdots, t_n\}$$

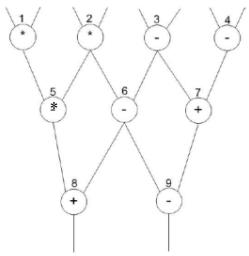
  - ***scheduling*** is a function $\varsigma : V \rightarrow Z^+$, where $\varsigma(t_i)$ denotes the starting time of task t_i.

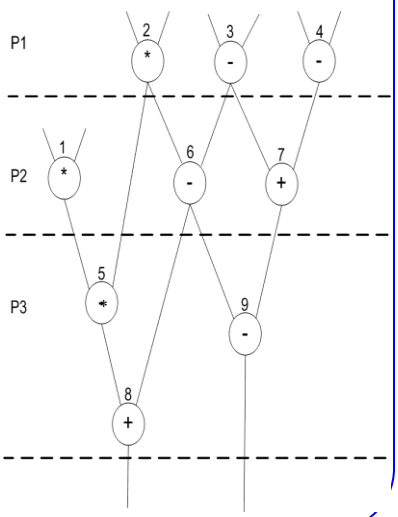
General vs. RCS High-Level Synthesis

- **Fundamental differences in RCS:**
  - Uniform resources:
    - It is possible to implement any task on a given part of a device (provided that the available resource are enough).

General vs. RCS High-Level Synthesis

- **Assumptions on a reconfigurable device**



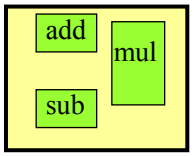


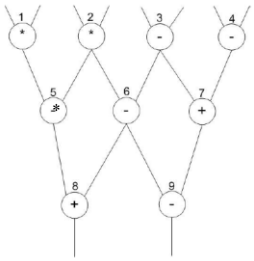
General vs. RCS High-Level Synthesis

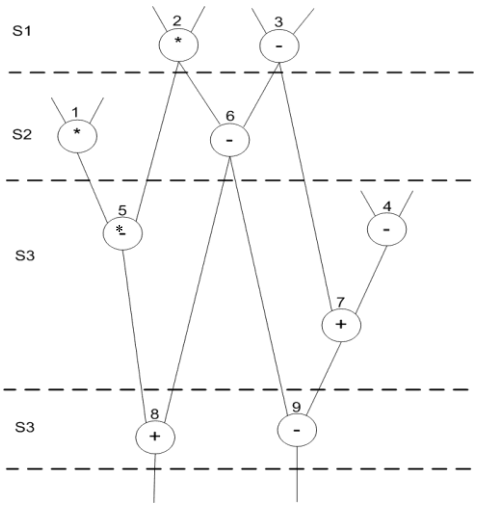
- **Example:**

$x = ((a \times b)(c \times d)) + ((c \times d) - (e - f))$

$y = ((c \times d) - (e - f)) - ((e - f) + (g - h))$







High-Level Synthesis

- **Fundamental differences in RCS:**
  - **In general HLS:**

    - Application is specified using a structure that encapsulates a data-path and a control part.

    - Control part is synthesized.
  - **In RCS:**

    - Hardware modules implemented as data-path normally compete for execution on the chip.
    - A processor is used to control selection process of the hardware modules by means of reconfiguration.

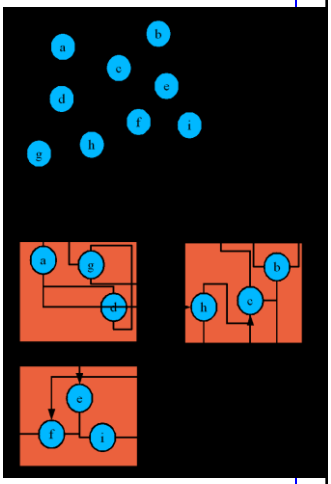
Temporal Partitioning

- Resources on the device are not allocated to only one operator but to a set of operators that must be placed at the same time and removed.

→ An application must be partitioned in sets of operators.

- The partitions will then be successively implemented at different time on the device.

Temporal Partitioning



Schedule

- **Schedule:**
  - is a function $\varsigma : V \rightarrow Z^+$, where $\varsigma(v_i)$ denotes the starting time of the node v_i that implements a task t_i.
- **Feasible Schedule:**
  - ς is *feasible* if: $\forall e_{ij} = (v_i, v_j) \in E$,

$$\varsigma(t_j) \geq \varsigma(t_i) + T(t_i) + t_{ij}$$

  - e_{ij} defines a data dependency between tasks t_i and t_j,
  - t_{ij} is the latency of the edge e_{ij},
  - $T(t_i)$ is the time it takes the node v_i to complete execution.

Ordering Relation

- **Ordering relation ≤**
  - $v_i \leq v_j \Leftrightarrow \forall$ schedule ς, $\varsigma(v_i) \leq \varsigma(v_j)$.

Note: ≤ is a *partial ordering,* as it is not defined for all pairs of nodes in G.

Partition

- **Partition:**
  - A *partition P* of the graph $G = (V,E)$ is its division into some disjoint subsets $P_1, \ldots, P_m$ such that

$$U_{k=1,\ldots,m} P_k = V$$

- **Feasible Partition:**
  - A partition is *feasible* in accordance to a reconfigurable device H with area $a(H)$ and pin count $p(H)$ if:
  - $\forall P_k \in P: a(P_k) = (\sum_{vi \in Pk} a_i) \leq a(H)$
  - $1/2 \sum_{eij \in E} w_{ij} \leq p(H)$
    - for e_{ij} = crossing edges
- **Crossing edge:**
  - an edge that connects one component in a partition with another component out of the partition.

Run Time

- **Run time of a partition $r(P_i)$:**
  - the maximum time from the input of the data to the output of the result.

Ordering Relation

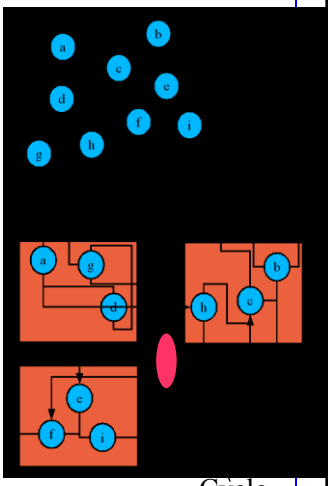
- **Ordering relation for partitions:**
  - $P_i \leq P_j \Leftrightarrow \forall v_i \in P_i, \forall v_j \in P_j$
    - either $v_i \leq v_j$
    - or v_i and v_j are not in relation.
- **Ordered partitions:**
  - A partitioning P is ordered $\Leftrightarrow$ an ordering relation $\leq$ exists on P.
  - If P is ordered, then for a pair of partitions, one can always be implemented after the other with respect to any scheduling relation.

Temporal Partitioning

- **Temporal partitioning:**
  - Given a DFG *G* = (*V,E*) and a reconfigurable device *H,* a *temporal partitioning* of *G* on *H* is an ordered partitioning *P* of *G* with respect to *H.*

Temporal Partitioning

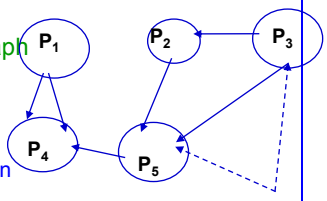
- **Cycles** are not allowed in DFG.
  - Otherwise, the resulting partition may not be schedulable on the device.



Cycle

Temporal partitioning

- **Goal:**
  - Computation and scheduling of a Configuration graph
  - **A configuration graph is a graph in which:**
    - Nodes are partitions
    - Edges reflect the precedence constraints in a given DFG

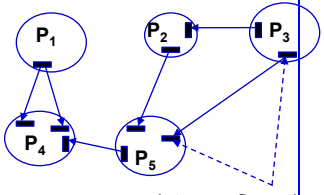


Configuration Graph

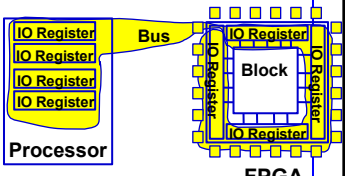
- **Formal Definition:**
  - Given a *DFG G = (V,E)*
  - and a temporal partitioning *P = {P1, ..., Pn}* of *G*, we define a ***Configuration graph*** of *G* relative to the *P*, with notation $\Gamma(G/P) = (P,E^P)$ in which the nodes are partitions in *P*. An edge $e^P = (P_i, P_j) \in E^P \Leftrightarrow \exists e = (v_i, v_j) \in E$ *with* $v_i \in P_i$ *and* $v_j \in P_j$.
- **Configuration:**
  - For a given partition *P*, each node $P_i \in P$ has an associated configuration ζ_i that is the implementation of P_i for the given device *H*.

Temporal partitioning

- Whenever a new partition is downloaded, the partition that was running is destroyed.
  - Communication through inter-configuration registers (or communication memory)
    - May sit in main memory
    - May sit at the boundary of the device to hold the input and output values
  - Configuration sequence is controlled by the host processor



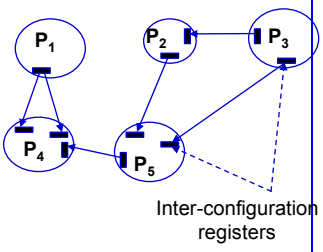
Inter-configuration registers



Device's register mapping into the processor address spaces

Temporal partitioning

- **Steps (for P_i and P_j, $(P_i \leq P_j)$):**
  1. Configuration for P_i is first downloaded into the device.
  2. Executes.
  3. P_i copies all the data it needs to send to other partitions into the communication memory.
  4. The device is reconfigured to implement the partition P_j
  5. Accesses the communication memory and collect the data.



Temporal partitioning

- **Objectives for optimization:**
  1. # interconnections: very important, since it minimizes:
     - The mount of exchanged data
     - The amount of memory for temporally storing the data
  2. # produced blocks (partitions)
     - Reduces the number of reconfigurations (total time?)
  3. Overall computation delay depends on
     - the partition run time
     - the processor used for reconfiguration
     - speed of data exchange
  4. Similarity between consecutive partitions (for partial)
  5. Overall amount of wasted resources on the chip.
     - When components with shorter run-times are placed in the same partition with other components with longer run-time, those with the shorter components remain idle for a longer period of time.

Wasted Resources

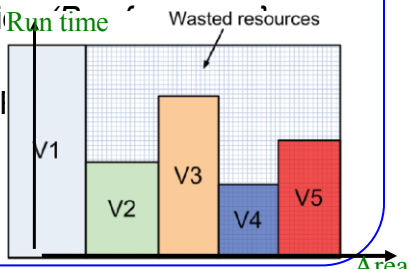
- Wasted resource $wr(v_i)$ of a node v_i:
  - Unused area occupied by the node v_i during the computation of a partition

    $\boldsymbol{wr(v_i) = (t(P_i) - T(t_i)) \times a_i}$

    $t(P_i)$: run-time of partition P_i.

    $(t_i))$: run-time of the component v_i

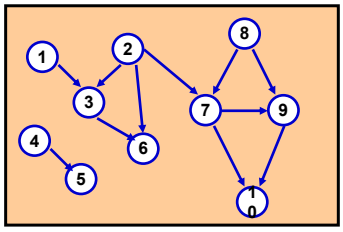
    a_i: area of v_i
- Wasted resource $wr(P_i)$ of a partiti
  $\boldsymbol{wr(P_i) = \sum_{j=1,\ldots,n} wr(v_i)}$
- Wasted resource of a partitioning I
  $\boldsymbol{wr(P) = \sum_{j=1,\ldots,k} wr(P_j)}$



Communication Overhead

- **Communication Cost:** *modelled as graph connectivity*:
- **Connectivity of a graph** ***G=(V,E)*:**

  $con(G) = 2*|E|/(|V|^2 - |V|)$
  - $|V|^2 - |V|$: *the number of all edges that can be built with V.*

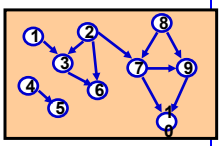


Connectivity = 0.24

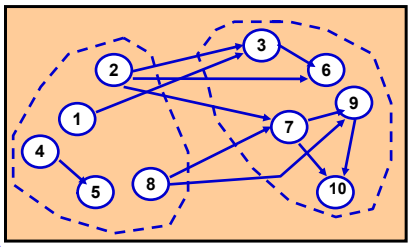
Communication Overhead

- *Quality of Partitioning P = {P1,…,Pn}:*
  - *Average connectivity over P:*

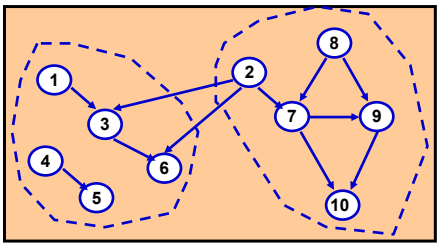
$$Q(P) = 1/n \sum_{i=1,\ldots,n} con(P_i)$$



Connectivity = 0.24



Quality = 0.25



Quality = 0.45

Communication Overhead

- Minimizing communication overhead by
  - **minimizing the weighted sum of crossing edges among the partitions.**
    - → **minimize the size of the communication memory and**
    - → **minimize the communication time.**
- Heuristic:
  - **Highly connected components are placed in the same partition (High quality partitioning)**

Temporal partitioning & Scheduling

- **Unconstrained Scheduling**
  - ASAP methods
  - ALAP methods
- **Constrained Scheduling**
  - List scheduling
  - Integer linear programming (exact method)
  - Network flow
  - Spectral method

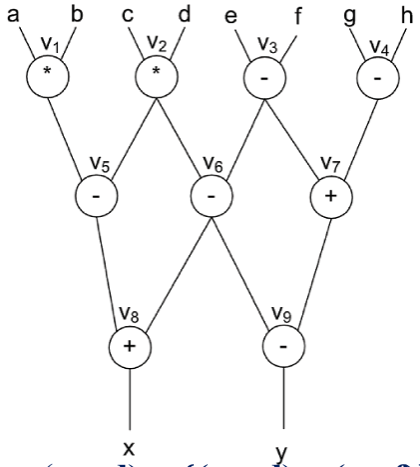
Unconstrained Scheduling

- **Unconstrained scheduling:**
  - Assumption: unlimited amount of resources
    - Device with unlimited size
  - Usually as pre-processing step for other algorithms
    - E.g. computation of the upper and lower bounds on the starting time of operations.
  - ***Lower bound:*** the earliest time at which a module can be scheduled,
  - ***Upper bound:*** the latest time at which a module can be started.

Unconstrained Scheduling

- **ASAP (as soon as possible)**
  - Defines the earliest starting time for each node in the DFG
  - Computes a minimal latency
- **ALAP (as late as possible)**
  - Defines the latest starting time for each node in the DFG according to a **given latency**
- **The mobility of a node:**
  - **(ALAP starting time) – (ASAP starting time)**
  - Mobility = 0 → node is on a critical path

Dataflow graph for example

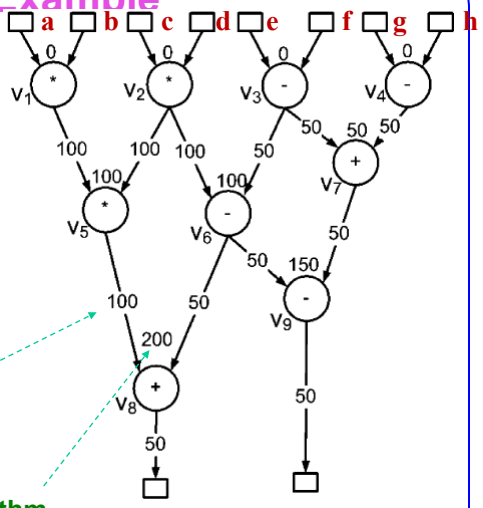


x = (a x b) x (c x d) + {(c x d) – (e - f)}

y = (c x d) – (e –f) – {(e –f) + (g-h)}

ASAP Example

- **Assumptions:**
  - *Multiplication:* latency of 100 clocks,
  - *Addition/subtraction:* 50 clocks,
  - data transmission delay is neglected.



Computation delay of the prev. node

Node's starting time as computed by the algorithm.

$x = (a \times b) \times (c \times d) + \{(c \times d) - (e - f)\}$

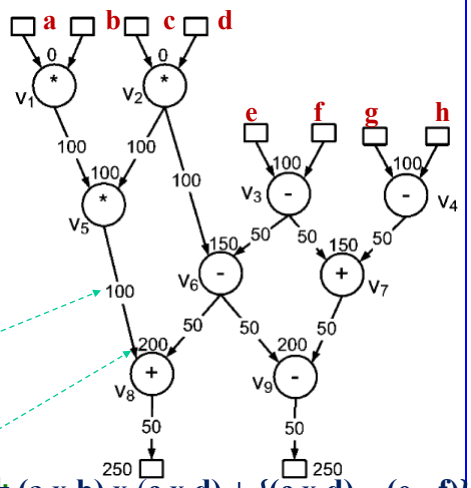
$y = (c \times d) - (e - f) - \{(e - f) + (g - h)\}$

ASAP Algorithm

1: **for** each node v ∈V **do**
2: **if** v has no predecessors **then**
3: ς(v) := 0
4: V := V − v
5: **end if**
6: **end for**
7: **while** V # ∅ **do**
8: select a vertex v_i ∈ V whose predecessors are all scheduled
9: schedule v_i by setting $\varsigma(v_i) := \max(v_j, v_i) \in E(\varsigma(v_j)+t_j)$
10: V:= V−v_i
11: **end while**

ALAP Example

- **Assumptions:**
  - *Multiplication:* latency of 100 clocks,
  - *Addition/subtraction:* 50 clocks,
  - Overall computation time: 250



Computation delay of the prev. node

Node's starting time as computed by the algorithm

$$x = (a \times b) \times (c \times d) + \{(c \times d) - (e - f)\}$$

$$y = (c \times d) - (e - f) - \{(e - f) + (g - h)\}$$

ALAP-Algorithm

1: **for** each node v ∈ V **do**
2: **if** v has no successors then
3: ς(v) := λ
4: V := V−v
5: **end if**
6: **end for**
7: **while** V # ∅ **do**
8: select a vertex v_i ∈ V whose successors are all scheduled
9: schedule v_i by setting ς(vi) := min(v_i,v_j) ∈ E(ς(v_j) − t_i)
10: V := V− v
11: **end while**

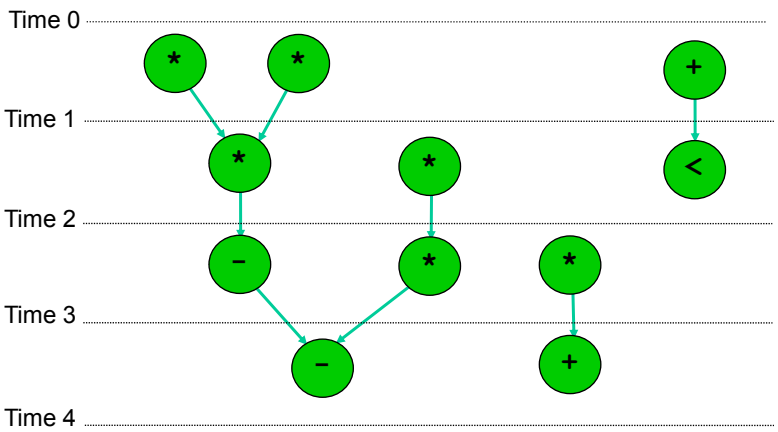
Constrained Scheduling

- **Constrained scheduling:**
  - A set of fixed resources available (ASIC).
  - Many tasks competing for a given resource,
    → One of them must be chosen according to a given criteria and the rest will be scheduled later.

1. **Extended ASAP, ALAP:**
   - Compute ASAP or ALAP
   - Assign the tasks earlier (ASAP) or later (ALAP), until the resource constraints (e.g. area) are fulfilled.

Extended ASAP

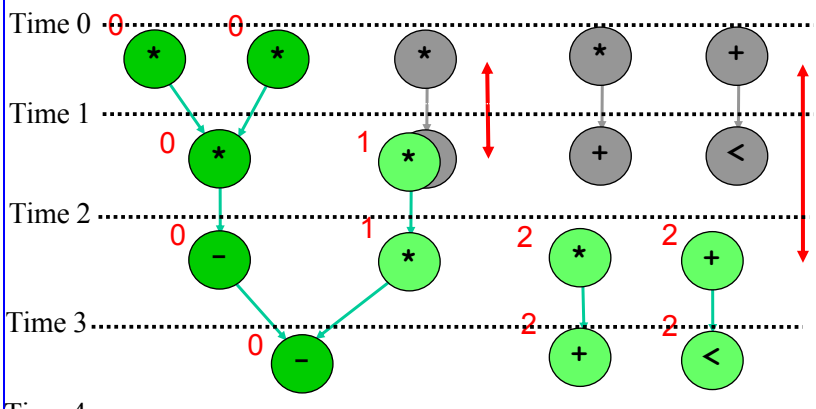
- **Constraint:**
  - 2 Multipliers, 1 ALUs (+, −, <)



Constrained Scheduling

- **List scheduling:**
  - Sort nodes in topological order
  - Assign priority to nodes
  - Criteria (priority) can be:
    - number of successors.
    - depth (length of longest path from inputs).
    - latency-weighted depth.
      - w: latency of the operation to be executed by the nodes on the path.
    - Mobility.
    - Connectivity.
    - ...

Mobility



Connectivity and quality

- **Graph Connectivity**: Given a dataflow graph G=(V,E), we define the connectivity.

  con(G)= $\frac{2|E|}{|V|^2-|V|}$ as the rapport between the number of edges in E over the number of all edges that can be built with the nodes of G

- **Quality**: Given a dataflow graph G=(V,E) and a partitioning P = {P_1, ..., P_n} of G, we define the quality.

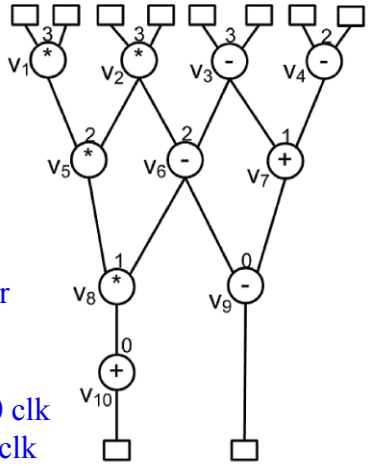
  Q(P)= $\frac{1}{n}\sum_{i=1}^{n}(con(P_i))$ of P as the average connectivity over all the partitions P_i, $1 \le i \le n$.

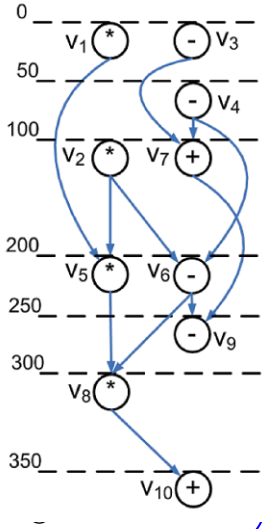
Constrained Scheduling

- **At any time step *t*:**
  - A ***ready set*** *L* is constructed (operations ready to be scheduled)
    - *L*: operations whose predecessors have already been scheduled early enough to complete their execution at time *t*.
  - Tasks are placed in *L* in decreasing priority order
  - At a given step, the free resource is assigned the task with highest priority.

List Scheduling: Example

- Resources:
  - 1 multiplier
  - 1 ALU
- Latency:
  - MUL : 100 clk
  - ALU : 50 clk
- The depth of a node as priority





Temporal Partitioning vs. Constrained Scheduling

- **In RCS,**
  - Resource *types* are not important.
    - *Amount* of basic resources are important.
  - Operators do not compete for resources.
    - They compete for area.
  - Only the starting time and the end time of the *complete partition* is usually considered.

Temporal Partitioning in RCS

sort the nodes of v according to their priorities

$P_0 := \emptyset$

while $V \neq \emptyset$ **do**

select a vertex $v \in V$ with highest priority and whose predecessors are all placed

if (a partition Pi exists with $s(Pi) + s(v) \leq s(H)$) then

$Pi = Pi \cup \{v\}$

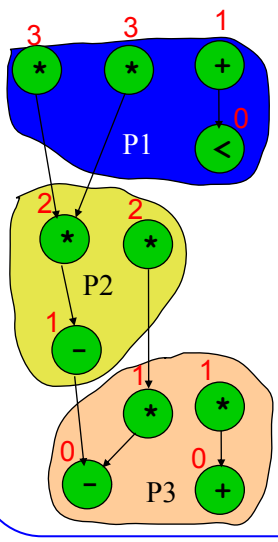
else

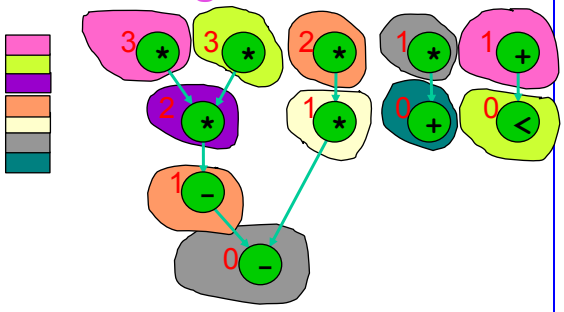
create a new partition Pi+1 and set $Pi+1 = \{v\}$

end if

end while

Temporal Partitioning vs. Constrained Scheduling





- Criterion: number of successors
- size(FPGA) = 250,
- size (mult) = 100,
- size(add) = size(sub) = 20,
- size(comp) = 10.

- Connectivity:
  - c(P1) = **1/6,**
  - c(P2) = **1/3,**
  - c(P3) = **2/6.**
- Quality: 0.28

Improvement

- **Best criteria:**
  - Total computation time of DFG:

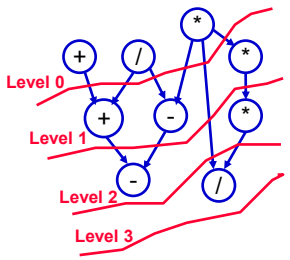
$$t_{DFG} = n \times C_H + \Sigma_{1,\ldots,n}(t_{Pi})$$

  - ***n:*** Number of partitions
  - C_H***:*** Reconfiguration time of device H
  - t_{Pi}***:*** Computation time of partition P_i.
- **Optimization:**
  - If C_H too large, then the optimization will tend to minimize the number of partitions
  - If $C_H \ll t_p$, then algorithm will tend to avoid long paths in partitions.

Improvement

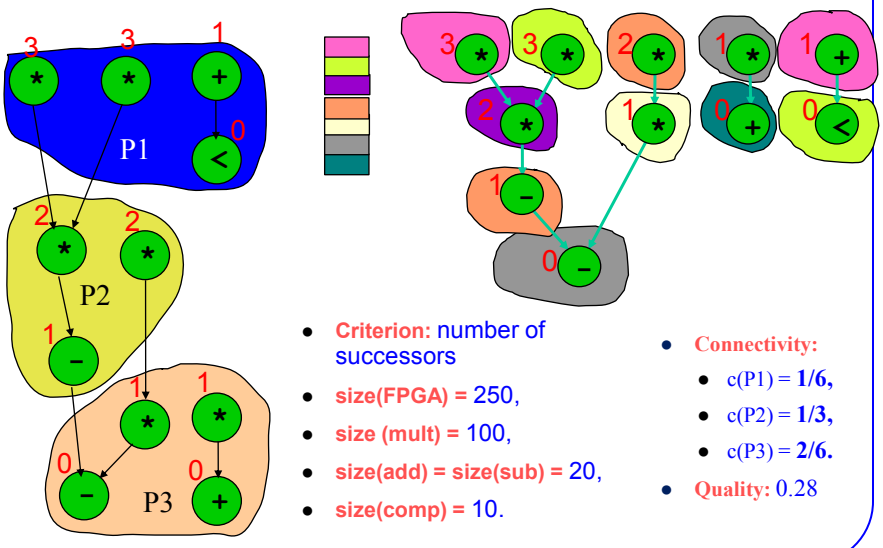
- **Advantage of LS-based temporal partitioning:**
  - Fast (linear time algorithm)

  →Local optimization possible



- **Disadvantage:**
  - Levelization:
    - Modules are assigned to partitions based more on their level number rather than their interconnectivity with other component.
  - Interconnectivity (data exchange) must be optimized.

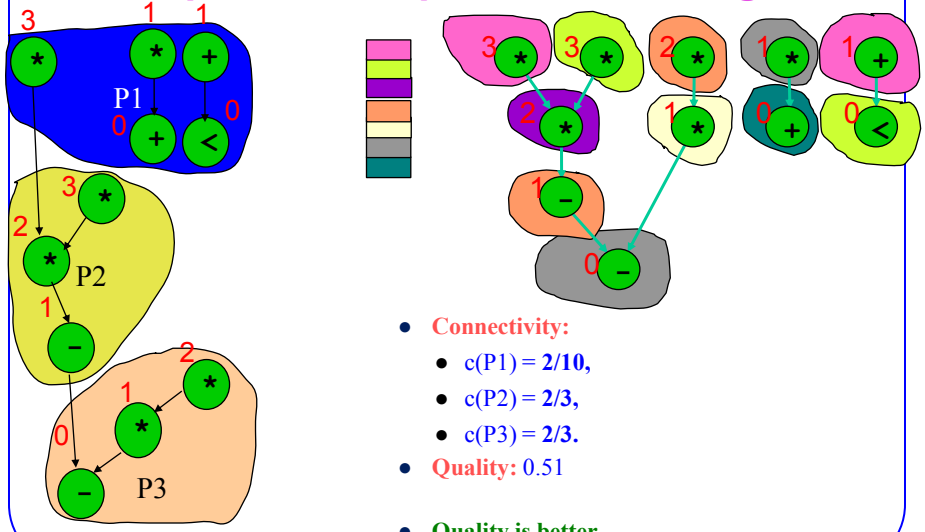
LS-Based Temporal Partitioning



- Criterion: number of successors
- size(FPGA) = 250,
- size (mult) = 100,
- size(add) = size(sub) = 20,
- size(comp) = 10.

- Connectivity:
  - c(P1) = **1/6,**
  - c(P2) = **1/3,**
  - c(P3) = **2/6.**
- Quality: 0.28

Improved Temporal Partitioning

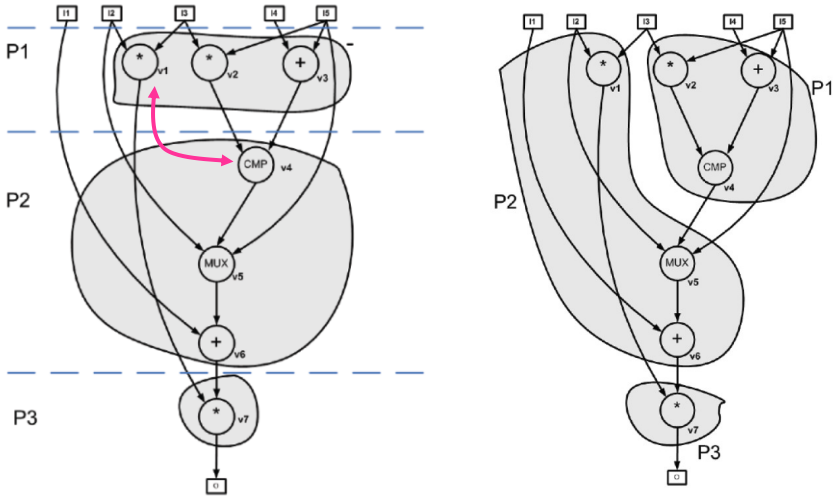


- Connectivity:
  - c(P1) = **2/10,**
  - c(P2) = **2/3,**
  - c(P3) = **2/3.**
- Quality: 0.51

- **Quality is better**

Improved List Scheduling

- **Pair wise interchange**



2.2 Temporal partitioning – ILP

- **Integer linear programming (ILP) problem.**

$$\min c^T x$$

$$Ax = b$$

$$x \geq 0$$

With A, b and c and x being matrices of integers. It consists of finding a variable x, which minimizes $c^T x$ under the side constraints Ax=b and x≥0.

2.2 Temporal partitioning – ILP

- **With the ILP (Integer Linear Programming),**
  - Temporal partitioning constraints are formulated as equations.
  - The equations are then solved using an ILP-solver.
- **The constraints usually considered are:**
  - Uniqueness constraint
  - Temporal order constraint
  - Memory constraint
  - Resource constraint
  - Latency constraint
- **Notations:**

$$(y_{vi} = 1) \Leftrightarrow (v \in P_i)$$

$$(w_{uv} \neq 0) \Leftrightarrow ((u \in P_i) \wedge (v \in P_j) \wedge (P_i \neq P_j))$$

2.2 Temporal partitioning – ILP

- **Unique assignment constraint:** Each task must be placed in exactly one partition. (m = # of partitions)

$$\forall v \in V, \sum_{i=1}^{m} y_{vi} = 1$$

- **Precedence constraint:** For each edge $e = (u, v)$ in the graph, u must be placed either in the same partition as v or in an earlier partition than that in which v is placed.

$$\forall (u,v) \in E, \sum_{i=1}^{m} i y_{ui} \leq \sum_{i=1}^{m} i y_{vi}$$

Temporal partitioning – ILP

- **Resource constraint:** The sum of the resources needed to implement the modules in one partition should not exceed the total amount of available resources.
  - Device area constraint: a(H) is the area of device H
  - Device terminal constraints: p(H) is the number of terminals (pins) of the device H.

$$\forall P_i \in P, \sum_{v \in P_i} a(v) \leq a(H)$$

$$\forall P_i \in P, \sum_{(u \in P_i) \wedge (v \notin P_i)} w_{uv} + \sum_{(u \notin P_i) \wedge (v \in P_i)} w_{uv} \leq p(H)$$

Temporal partitioning – ILP

- **Communication memory constraint:** The total amount of data to be **temporally saved must not exceed the size of the communication memory used**.
  - This constraint is captured by the following equation:

$$\sum_{(u,v) \in E \text{ and } (u \in P_i) \wedge (v \in P_{j(j>i)})} w_{uv} \leq M_s$$

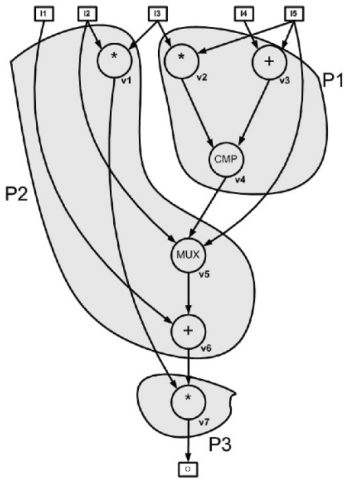
Temporal partitioning by ILP: Example

- *assignment constraint:*
  - *y11+ y12 + y13 = 1*
  - *y21+ y22 + y23 = 1*
  - *…*
  - *y71 +y72 + y73 = 1*

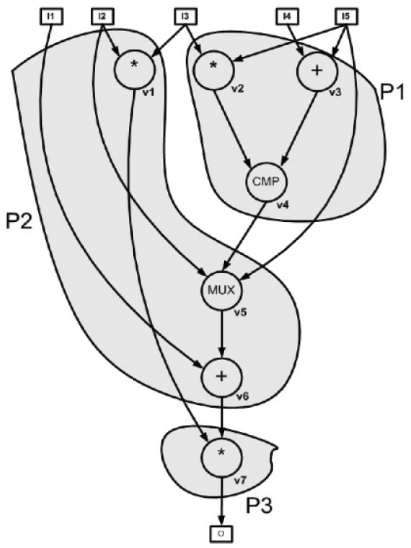
  - *Partition P1:*
  - *y22 = y23 = 0, y21 = 1*
  - *y32 = y33 = 0, y31 = 1*
  - *y42 = y43 = 0, y41 = 1*

  - *Partition P2:*
  - *y11 = y13 = 0, y12 = 1*
  - *y51 = y53 = 0, y52 = 1*
  - *y61 = y63 = 0, y62 = 1*

  - *Partition P3:*
  - *y71 = y72 = 0, y73 = 1*



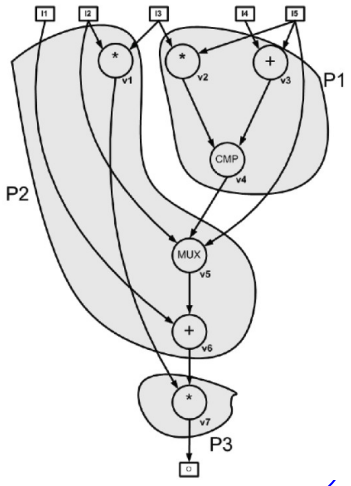
Temporal partitioning by ILP: Example



Temporal partitioning by ILP: Example

- *Precedence constraint:*

$$1 = \sum_{i=1}^{n} y_{4i} \leq \sum_{i=1}^{n} y_{5i} = 2$$
$$2 = \sum_{i=1}^{n} y_{6i} \leq \sum_{i=1}^{n} y_{7i} = 3$$



Temporal partitioning by ILP: Example

- **Resource constraint:**
  - device with a size of 200 LUTs, and 100 LUTs for the multiplication, 50 LUTs each for the addition, the comparison

Partition P_1: $\sum_{u=1}^{7} y_{u1} = (100 + 50 + 50) \leq a(H) = 200$
Partition P_2: $\sum_{u=1}^{7} y_{u2} = (100 + 50 + 50) \leq a(H) = 200$
Partition P_3: $\sum_{u=1}^{7} y_{u3} = (100) \leq a(H) = 200$

Temporal partitioning by ILP: Example

- *Communication memory constraint:*
  - ***Assume that a memory with 50 bytes is available for communication and each datum has a 32-bit width.***

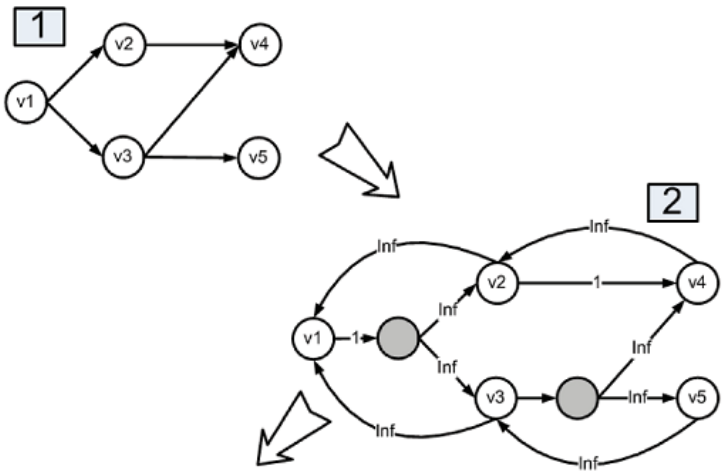
For P1 to P2

$w_{45} = 2 \text{ x } 32 = 64bit \leq Ms = 8 \times 50 = 400bit$

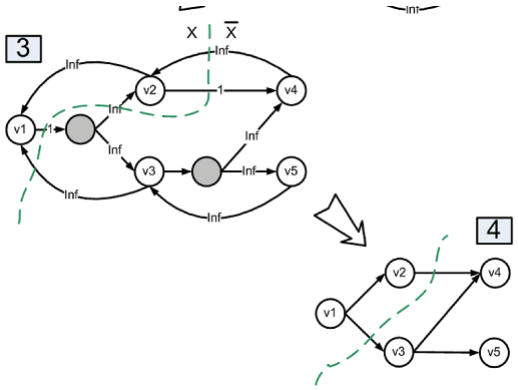
For P2 to P3

$w_{67} + w_{17} = 2 \text{ x } 32 + 2 \text{ x } 32 = 128bit \leq Ms = 8 \times 50 = 400bit$

Network-flow Transformation step



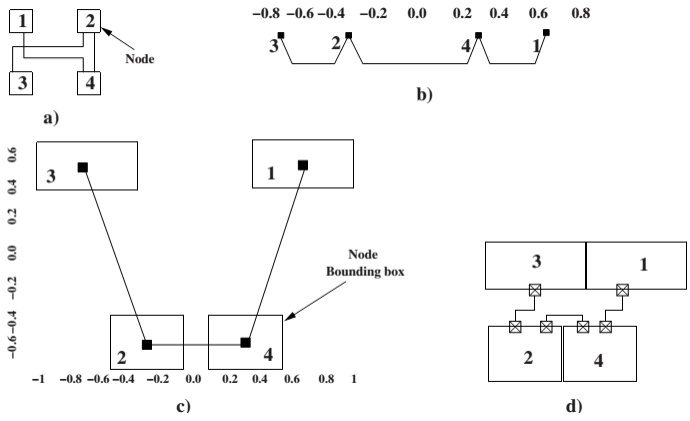
Network-flow partitioning steps



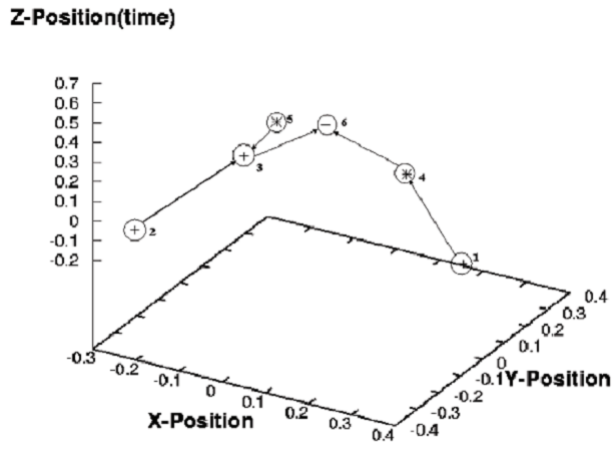
Spectral Methods

- Minimization of communication between partitions
- The connectivity of a graph can be minimized by placing components in an n-dimensional space in such a way that the sum of the distance between component pairs is minimized
  - Placement of the components in an n-dimensional vector space such as to minimize the sum of the distances between the components.
  - Derivation of a partition from an optimal placement that minimizes the sum of the distance between the components.

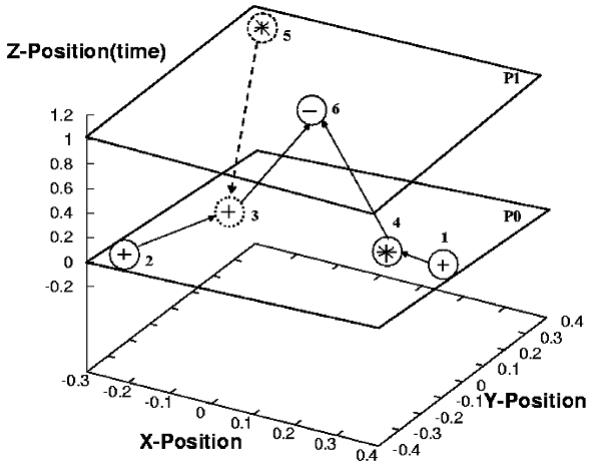
Spectral Methods - 1-D and 2-D spectral placement



Spectral Methods 3-D spectral placement



Spectral Methods - Derived partitioning



Thanks you