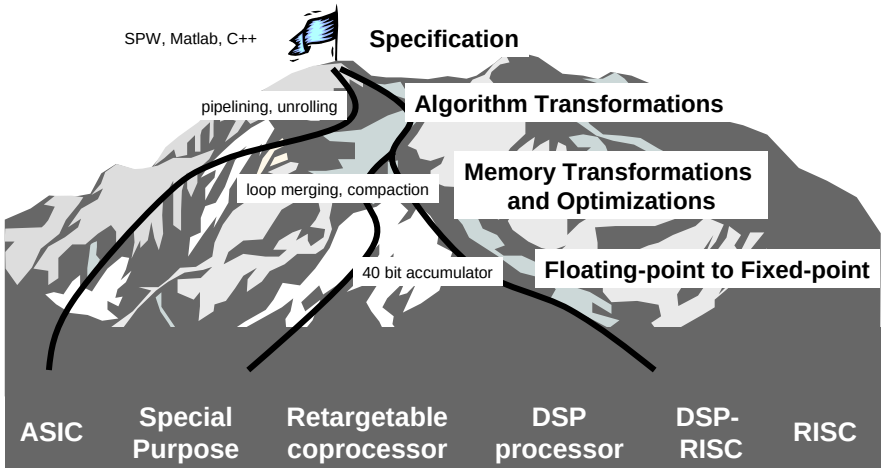
Lecture 3:
System specifications and models of computation

I. Verbauwhede,
2009
KULeuven

H05H4, H05E7: Skiing down a mountain



Overview

- Lecture 1: what is a system-on-chip
- Lecture 5: fixed point refinement
- Lecture 2: terminology for the different steps
- Lecture 3 – today: models of computation
  - Synchronous data flow graphs
  - Control flow models

Last lecture: Digital Abstraction Levels

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

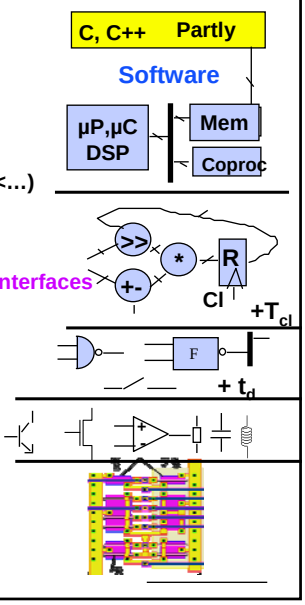
Algorithmic:
- Refinement of behaviour to Hw-Sw architecture
- DT, ADT to bitvector, int; primitive operations (+,-,*,>>,<<…)

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

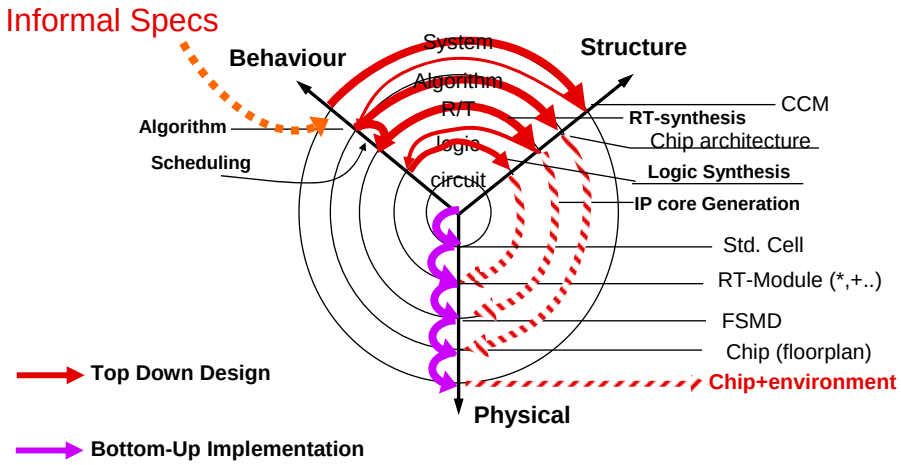
Transistor:
- v(t), i(t), ODE's, Netw, Eq. R.L,C,E,I,M…

Polygon



ADT = Abstract Data Type, DT = Discrete Time

Last lecture: Global SoC design-flow



– 5

System Specs and Models of Computation

Outline:

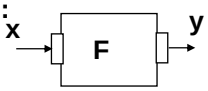
- **What are specifications?**
- How to specify TIME?
- Two most important models of computation:
  - **Synchronous data flow graphs**
  - **Control flow models**

– 6

What are specifications?

From informal specs to an 'executable' form of spec:

1. **Functional specification (what?)**
   = relation between inputs, outputs (and states)
   = formal mathematical framework to describe behavior



2. **A set of properties that must be satisfied: assertions**
   = relation that must be satisfied if the functional behavior is correct
   e.g. deterministic behavior, bounded memory

3. **A set of performance indexes (per abstraction layer) :**
   Most important ones: Power, Area, Timing, Price,
   Estimators: E(P), E(A), E(T), E(GBW)…

4. **A set of constraints:** $\underline{E}(X) < \underline{C}$

E: Estimator

Specs and Occam's principle

- **Specs needed at all abstraction layers (AL)**
- **Design is *top-down refinement* of specifications**
- ***Requires models of reality but not more...***

"Entia non sunt multiplicanda praeter necessitatem"

No more things should be presumed to exist than are absolutely necessary.

W. Occam 1280-1349

Model of computation

Model = operates on the signal and time representation at each abstraction level!

= not implementation but abstract functionality such that:

1. **As little as possible restriction on implementation (keep freedom)**
2. **Be simple and formal enough to allow good validation at given level of abstraction**
3. **Be executable**

– 9

Executable and formal models: why?

Validation at all levels:

- By construction
  - inherent to model (property guaranteed)
- By formal verification
  - proof of properties possible as property of model
- By simulation
  - check expected behaviour for all (?) inputs
  - checking of assertions

It is better to be higher in this list…hence understand computational models and their properties.

Many different models co-exist in a given system...

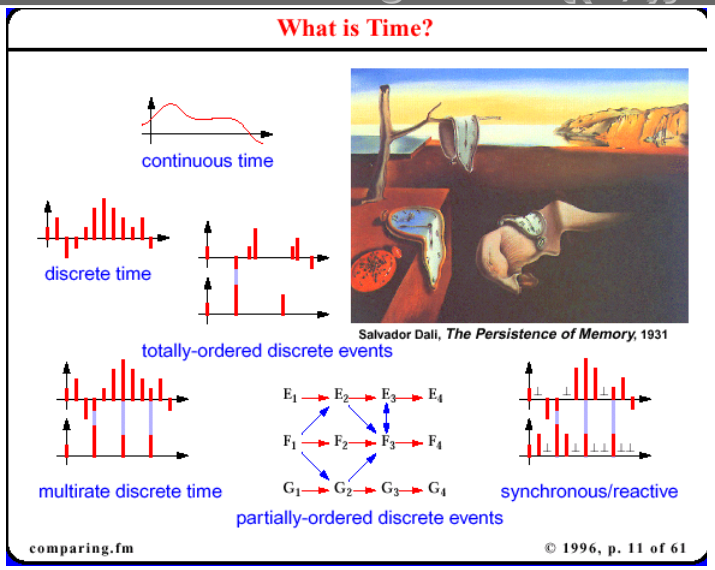
– 10

System Specs and Models of Computation

Outline:

- What are specifications?
- **How to specify TIME?**
- Two most important models of computation:
  - Synchronous data flow graphs
  - Control flow models

Many representations of time, values and signals = {(v,t)}



Salvador Dali, *The Persistence of Memory*, 1931

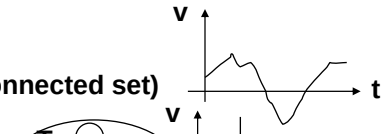
After Ed Lee, UCB

Time Representations

- **Tag t is abstraction of time (temporal order)**
  - **- Absolute time = global ordering=overspecification**
  - **- Cumbersome and harmful because reduces degree of freedom**
  - **- Order in t is order in events (t<t' <=> e<e')**
- **3 representations:**

-Absolute time

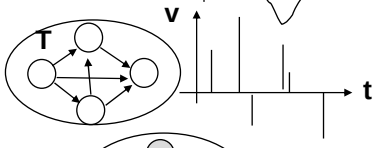
$\mathrm{T} = \Re$ **(T totally ordered closed connected set)**



-Discrete time

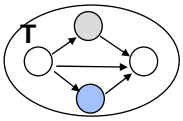
T is totally ordered discrete set

$<$ in T such that $t \neq t' \Rightarrow (t < t') \oplus (t' < t) = 1$



-Precedences

T is partially ordered discrete set



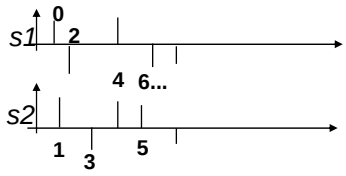
Functional Behaviour

- **Timed Models of Computation = total order**
  - **– Continuous time-**
  - **– Discrete event-**
  - **– Cycle accurate**
  - **– Instruction accurate**
  - **– Transaction accurate**
  - **– …**
- **Untimed Models of Computation = partial order**
  - **– Sequential Processes with Rendez-Vous**
  - **– Kahn Networks**
  - **– Data-Flow networks**
  - **– …**

Discrete Event System

Let *Q* be a timed system and *s*∈*Q*, let *T(s)* be set of tags appearing in any signal *s* in *s*.

- ***Q* is a *discrete event* system if, for each *s*∈*Q* there exists an *order-preserving* bijection from some subset of the natural numbers to *T(s)***



1 2 3 4 5 6 …

Any pair of events in a signal has a finite number of intervening events

There may exist concurrent events (one tag)

There is a first event

Events can be "indexed" by natural numbers

Discrete event simulation

- **Based on event queue Q. Presupposes delta causality to guarantee convergence in simulation.**

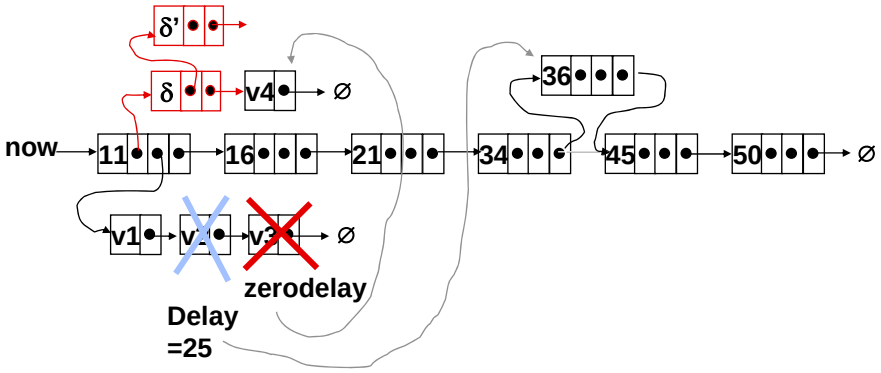
{ Put input events e_i in appropriate slots of Q;
While Q not empty
{ At next non empty slot t in Q:
for all $e_i \in$ t:
{Compute output events e_o from e_i ;
Remove e_i from slot t ;
Project e_o^* on Q at t+Δ_{oi} slot (Δ_{oi}: delay of e_o w.r.t e_i);}
}
}

Sorting!!!

Simulation mechanism of VHDL, MATLAB-STATEFLOW

*** e_o's are e_i's of fan-out processes**

Queue = Linked List



Future event: insertion
Zero Delay : 2 delta lists

Synchronous Models

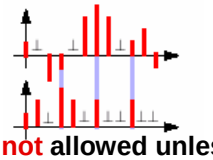
- Two events are *synchronous* if they have the same tag.
- Two *signals* are synchronous if all events in one signal are synchronous with an event in the other signal and vice versa.
- A *system* is synchronous if every signal in the system is synchronous with every other signal in the system.
- A *discrete-time system* is a synchronous discrete-event system.

Clocked discrete-time Systems

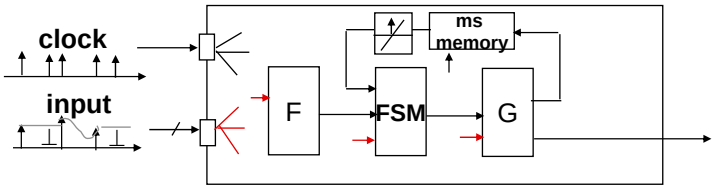
All tags t in T are determined by a set of clock ticks that are globally available. All signals in the system are considered as broadcasted events computed in zero delay processes.

Non-changing signals are considered as empty events ⊥

Input signals are synchronous to clock signal.



Delay-free loops are not allowed unless they contain master-slave memories synchronous to the clock ticks.



F,FSM,G
0 delay

Clocked Discrete-Time Simulation

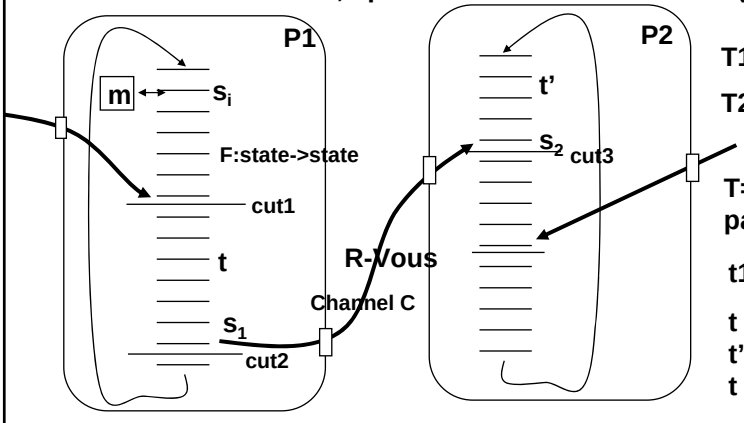
- Also called cycle based simulation
- Does not require sorting: faster than event-driven simulation (if not too many empty signals)
- Computation of acyclic process graph by topological sort at compile time (code generation).
- Ideally suited for RT-level simulation, instruction set simulators, sampled data systems (clock period is fastest rate stream period).
- Global instantaneous broadcasting communication
- Focus on functionality. Effect of computation delay is a timing verification problem at lower AL.
- Inefficient when high degree of inactivity (then DE)

Untimed Models of Computation

- **Model *Q* in which *T(s)* with $s \in Q$ is a partially ordered set. Also called *asynchronous* system**
- **Partial order creates *freedom* of implementation**
- **Important concept for distributed state based systems that progress at their own pace except at *synchronisation points* or systems that progress through *availability of data*.**
  - **Sequential processes with rendez-vous**
  - **Kahn Process and Data Flow networks**

Ex1: Sequential processes with Rendez-Vous

Sequential process*: totally ordered (infinite[1]) sequence of states s_i at which instructions read, operate and write to a store m (Turing machine)



T1: totally ordered

T2: totally ordered

T=T1∪ T2: partially ordered

t1(s1)=t2(s2)=tc

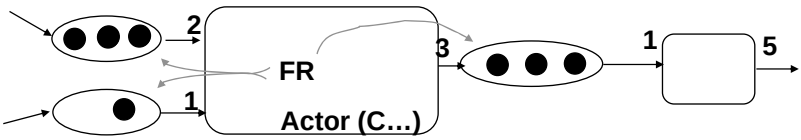
t < tc
t'< tc
t < t' or t'< t

- **Channel : point-to-point**
- **Blocking-send /Blocking-receive (no buffer required)**

***thread**

Ex 2: Data-Flow networks

- **Special case of Kahn Networks:**
- **Actors are *fired* when a prescribed number of tokens are available at the inputs. The actor *consumes* the tokens and *produces* a prescribed number of tokens at the output. *Firing rule (FR).***
- **Depending on the FR such networks have very interesting properties that make DF models *the key to describe DSP systems.***



– 23

System Specs and Models of Computation

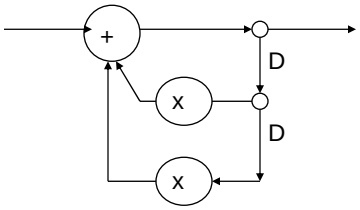
Outline:

- What are specifications?
- How to specify TIME?
- Many models of computation
- Two most important models of computation:
  - **Synchronous data flow graphs**
  - **Control flow models**

– 24

Data flow

Data flow representation of an algorithm:
- **is a directed graph**
- **nodes are computations (actors)**
- **arcs (or edges) are paths over which the data ("samples") travels.**



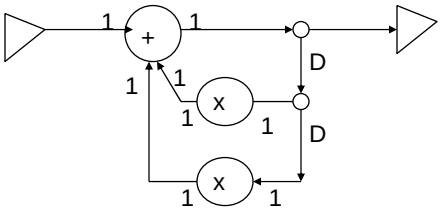
DF shows *which* computations to perform, *not sequence*.
Sequence is *only* determined by data dependencies.
Hence exposes concurrency.

Data flow (cont.)

Assume infinite stream of input samples.
So nodes perform computations an infinite times.

Node will "fire" (start its computation) when inputs are available.
Node with no inputs can fire anytime.

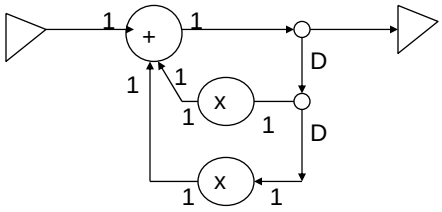
Numbers indicate the number of samples (tokens) produced, consumed by one firing.



Nodes will fire when input data is available, called "data-driven".
Hence it exposes concurrency.
Nodes must be free of "side effects": e.g. a write to a memory location followed by a read, only allowed if there is an arc between them

Data flow (cont.)

True data flow: overhead for checking the availability of input tokens is too large.
BUT, ***synchronous data flow***: the number of tokens produced/consumed is know beforehand (a priori)!
Hence, the scheduling can done a priori, at compile time. Thus there is NO runtime overhead!

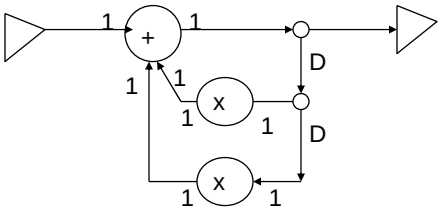


For signal processing applications: the number of tokens produced & consumed is independent of the data and known beforehand (= relative sample rates).

– 27

Synchronous Data Flow - definition

Synchronous data flow graph (SDF) is a network of synchronous nodes (also called blocks).
A node is a function that is invoked whenever there are enough inputs available. The inputs are *consumed*.
For a synchronous node, the consumptions and productions are known a priori.



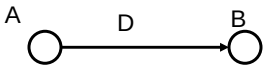
Homogeneous SDF graph: when only "1"'s on the graph.

– 28

Delay - D

Delay of signal processing

Unit delay on arc between A and B, means



n-th sample consumed by B, is (n-1)th sample produced by A.

- **Initialized by d zero samples**

A synchronous compiler

Translation from SDF graph to a sequential program on a processor

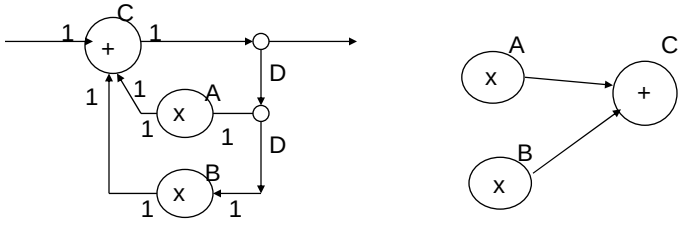
Two tasks:

- **Allocation of shared memory between blocks or setting up communication between blocks**
- **Scheduling blocks onto processors such that all input data is available when block is invoked**

Goal: create Periodic Admissible Parallel Schedule (PAPS)

Precedence graph - Schedule

Precedence graph indicates the sequence of operations:



Schedule determines when and where (which processor or which data path unit) the node fires.

Valid schedules:

A	B	C

B	A	C

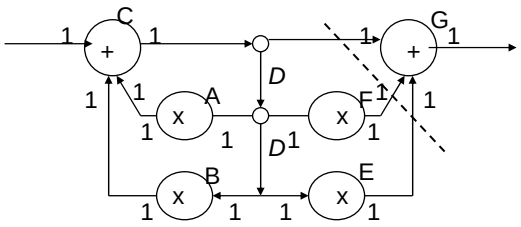
Invalid schedule:

C	A	B

– 31

Blocked Schedule

Blocked: one cycle terminates before next one starts



Static schedule

3 processors/units: valid blocked schedule

P1	A	C	G
P2	B	F	
P3	E		

With pipeline (not blocked):

P1	A	C
P2	B	F
P3	G	E

– 32

Small – large grain

Iteration period = length of one cycle = 1/throughput

Goal: minimize iteration period

Iteration period bound = minimum achievable (assuming pipelining)
= bound by total number of operations in loop divided by number of delays in the loop)

Atomic SDF graph, when nodes are primitive operations
Large grain SDF graph, when nodes are larger functions:

Example: IIR filter = small grain
JPEG = large grain

– 33

SDF graph implementation

Implementation requires:
• buffering of the data samples passing between nodes
• schedule nodes when inputs are available

Dynamic implementation (= runtime) requires
• runtime scheduler checks when inputs are available and schedules nodes when a processor is free.
• usually expensive because overhead

Contribution of Lee-87:
• SDF graphs can be scheduled at compile time
• no overhead

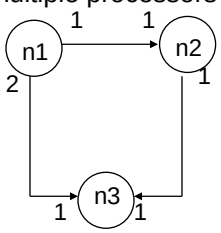
Compiler will:
• determine the execution order of the nodes on one or multiple processors or data path units
• determine communication buffers between nodes.

– 34

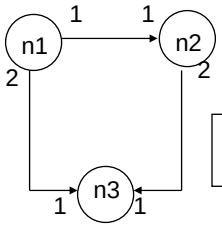
Periodic schedule for SDF graph

Assumptions:
- infinite stream of input data (the case for signal processing applications)
- periodic schedule: same schedule applied repetitively on input stream

Goal:
- check if schedule can be found:
- *Periodic admissible sequential schedule* (PASS) for a single processor or data path unit
- *Periodic admissible parallel schedule* (PAPS) for multiple processors



Rate inconsistency



PASS

n1	n2	n3	n3

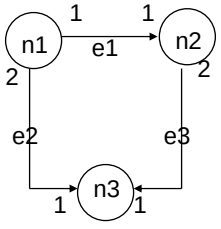
Consistent solution

Formal approach

Construct topology matrix
- each node is a column
- each arc is a row
- entry (i,j) = data produced on node i by arc j.
- consumption is negative entry

$\Gamma =$

	n1	n2	n3
e1	1	-1	0
e2	2	0	-1
e3	0	2	-1



Self loop entry?

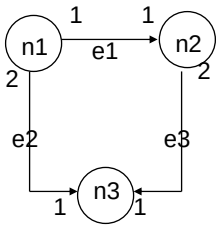
FIFO queues

b(n) = size of queues on each arc

v(n) = $\begin{bmatrix}1\\0\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\\0\end{bmatrix}$ or $\begin{bmatrix}0\\0\\1\end{bmatrix}$ indicates firing node

b(n+1) = b(n) + Γ v(n)

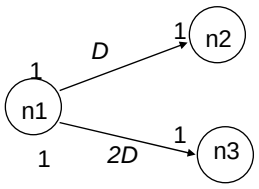
$$\Gamma = \begin{array}{c} \\ e1 \\ e2 \\ e3 \end{array} \begin{array}{c} \begin{array}{ccc} n1 & n2 & n3 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & -1 \\ 0 & 2 & -1 \end{bmatrix} \end{array}$$



b(0) = $\begin{bmatrix}0\\0\\0\end{bmatrix}$, b(1) = $\begin{bmatrix}1\\2\\0\end{bmatrix}$

FIFO queues & delays

Delays are handled by initializing b(0) with the delay values:



b(0) = $\begin{bmatrix}1\\2\end{bmatrix}$

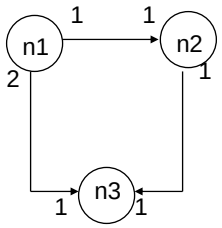
So at start-up:
- can fire n3 two times before firing n1 again

So, every directed loop must have at least one delay to be able to start

Identifying inconsistent sample rates

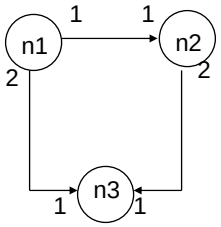
Necessary condition for the existence of periodic schedule with bounded memory

Rank of Γ is s-1 (s is number of nodes)



	n1	n2	n3
e1	1	-1	0
e2	2	0	-1
e3	0	1	-1

rank?



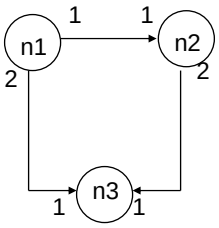
	n1	n2	n3
e1	1	-1	0
e2	2	0	-1
e3	0	2	-1

rank?

Relative firing frequency

Topology matrix with the correct rank, has a strictly positive (element-wise) integer vector ***q*** in its right nulspace:

Thus: $\Gamma q = 0$



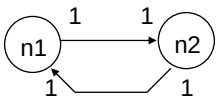
	n1	n2	n3
e1	1	-1	0
e2	2	0	-1
e3	0	2	-1

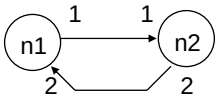
rank = 2, $q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$

q determines number of times each node is invoked!

Insufficient delays

Rank s-1 is a necessary but not a sufficient condition:





$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Scheduling for single processor

Given:
- positive integer vector ***q***, such that $\Gamma \boldsymbol{q} = 0$
- given b(0)

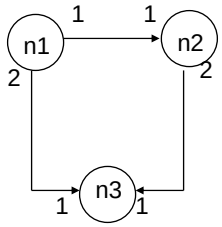
The i-th node is “runnable” if
- it has not been run *q*i times
- it will not cause the buffer size to become negative

Class “S” (sequential) algorithm creates a static schedule:
- is an algorithm that schedules a node if it is runnable
- it updates b(n)
- it stops when no more nodes are runnable.

If the class S algorithm terminates before it has scheduled each node the number of times specified in the q vector, then it is said to be *deadlocked*.

Example Class S algorithm

- Solve for smallest positive integer q
- Form a list of all nodes in the system
- for each node, schedule if runnable, try each node once
- if each node has been scheduled qi times, STOP.
- If no node can be scheduled, indicate deadlock
- else continue with the next node.



Schedule:
1 - 2 - 3 -3 is PASS
1 - 2 - 3 is not PASS
2 - 1 - 3 -3 is not PASS

(Complexity: traverse the graph once, visiting each edge once).
Optimization: minimize buffer (=memory) requirements

– 43

Schedule for parallel processors

Assumptions:
- homogeneous processors, no overhead in communication

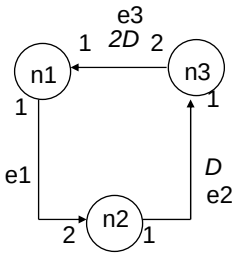
- if PASS exists, then also PAPS
  (because we could run all nodes on one processor)

A blocked periodic admissible parallel schedule is
- set of lists {Xi; i = 1, ... M}
- M is the number of processors
- Xi = periodic schedule for processor i

$\boldsymbol{p}$ is smallest positive integer vector, such that $\Gamma\boldsymbol{p} = 0$.
Then a cycle of schedule invokes every node
$\boldsymbol{q} = J\boldsymbol{p}$ times.
J is called the blocking factor (can be different from 1).

– 44

Precedence graph



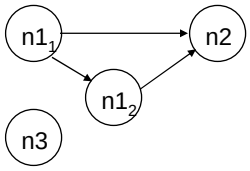
	n1	n2	n3
e1	1	-2	0
e2	0	1	-1
e3	-1	0	2

rank = 2, p = $\begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$

$\Gamma \boldsymbol{p} = 0.$

PASS: ?

Precedence graph for unity blocking factor:

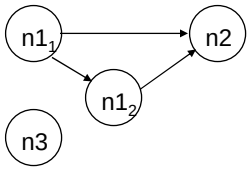


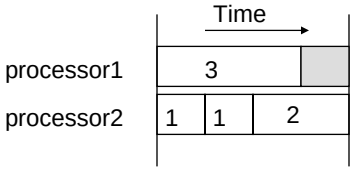
Schedule on two processors, J=1

Assumptions:

- node 1 takes 1 time unit, node 2 takes 2, node 3 takes 3

- X1 = {3}
- X2 = {1, 1, 2}

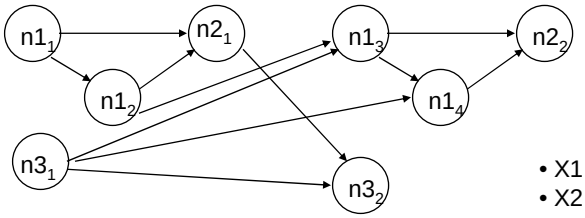




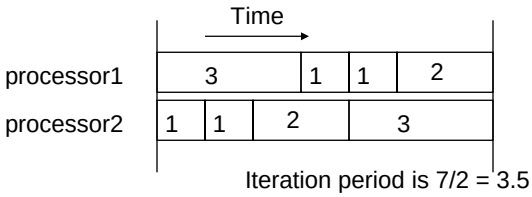
Iteration period = 4

Schedule on two processors, J=2

Assumptions:
- node 1 takes 1 time unit, node 2 takes 2, node 3 takes 3
- nodes have self loops (so nodes can not overlap with themselves)



- X1 = {3, 1, 1, 2}
- X2 = {1, 1, 2, 3}



Iteration period is 7/2 = 3.5

– 47

Why are we doing this?

The principle of synchronous data flow is used in many simulators. Based on this, multi-dimensional data flow representations have been developed.

Reality is always more complicated….
Issues in practice:
- choose schedule to minimize memory requirements.
- include non data flow nodes
  - if-then-else
  - data dependent calculations

– 48

Conclusion

Models of computation

- **Associated with levels of abstraction**
- **Allow reasoning without details**
- **Need to know the boundaries (where applicable, where not)**
- **Tagged Signal Framework is a classification system**

We will use a lot:

- **Data flow representation**
- **Control flow**