Embedded Systems 8

Dataflow modeling

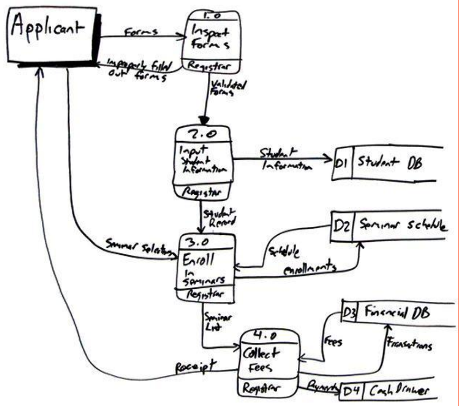
Lee/Seshia Section 6.3

Marwedel Section 2.5

- Identifying, modeling and documenting how data moves around an information system.
- Dataflow modeling examines
  - *processes* (activities that transform data from one form to another),
  - *data stores* (the holding areas for data),
  - *external entities* (what sends data into a system or receives data from a system, and
  - *data flows* (routes by which data can flow).
- Dataflow modeling focuses on *how things connect*, (imperative programming: *how things happen*).
- Scheduling responsibility of the system, not programmer

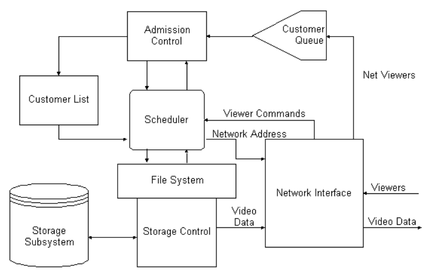
Data flow modeling

Registering for courses



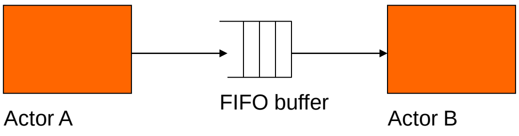
http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm

Video on demand system



www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Dataflow models



Buffered communication between concurrent components (*actors*).

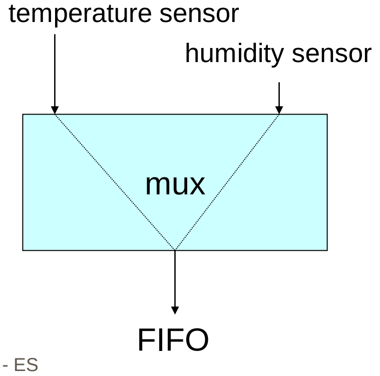
An actor can fire whenever it has enough data (*tokens*) in its input buffers. It then produces some data on its output buffers.

In principle, buffers are unbounded. But for implementation on a computer, we want them bounded (and as small as possible).

Process networks

Many applications can be specified in the form of a set of communicating processes.

Example: system with two sensors:



Alternating read

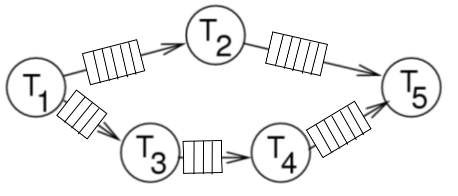
loop
read_temp; read_humidity
until false;

of the two sensors
not the right approach.

Reference model for dynamic data flow:
Kahn process networks (1974)

Describe computations to be performed and their dependence
but not the order in which they must be performed

communication via infinitely large FIFOs



Properties of Kahn process networks (1)

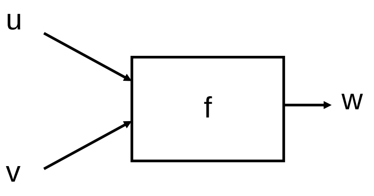
- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- Channels transmit information within an unpredictable but finite amount of time;
- Mapping from ≥1 input sequence to ≥1 output sequence;
- In general, execution times are unknown;
- Send operations are non-blocking, reads are blocking.
- One producer and one consumer;
  i.e. there is only one sender per channel;

Properties of Kahn process networks (2)

- There is only one sender per channel.
- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the arrival time.
- Therefore, Kahn process networks are deterministic (!); for a given input, the result will always the same, regardless of the speed of the nodes.

A Kahn Process

```
process f(in int u, in int v, out int w)
{
  int i; bool b = true;
  for (;;) {
    i = b ? wait(u) : wait(w);
    printf("%i\n", i);
    send(i, w);
    b = !b;
  }
}
```



Process alternately reads from u and v, prints the data value, and writes it to w

Source: Gilles Kahn, The Semantics of a Simple Language for Parallel Programming (1974)

A Kahn Process

```
process f(in int u, in int v, out int w)
{
  int i; bool b = true;
  for (;;) {
    i = b ? wait(u) : wait(w);
    printf("%i\n", i);
    send(i, w);
    b = !b;
  }
}
```

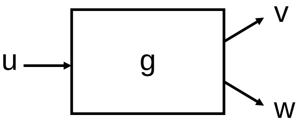
wait() returns the next token in an input FIFO, blocking if it’s empty

send() writes a data value on an output FIFO

Source: Gilles Kahn, The Semantics of a Simple Language for Parallel Programming (1974)

A Kahn Process

```
process g(in int u, out int v, out int w)
{
  int i; bool b = true;
  for(;;) {
    i = wait(u);
    if (b) send(i, v); else send(i, w);
    b = !b;
  }
}
```

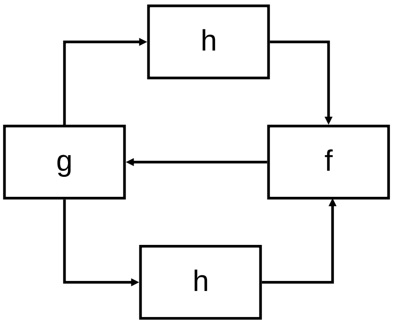


Process reads from u and alternately copies it to v and w

A Kahn System

- Prints an alternating sequence of 0's and 1's

Emits a 1 then copies input to output



Emits a 0 then copies input to output

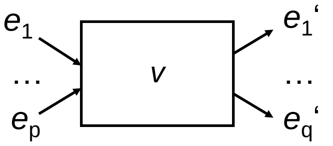
Definition: Kahn networks

A **Kahn process network** is a directed graph (*V*,*E*), where

- *V* is a set of **processes**,
- $E \subseteq V \times V$ is a set of **edges**,
- associated with each edge *e* is a **domain** D_e
- D^ω: finite or countably infinite sequences over *D*

  D^ω is a complete partial order where
  $X \leq Y$ iff *X* is an initial segment of *Y*

Definition: Kahn networks



- associated with each process $v \in V$ with incoming edges $e_1, \ldots, e_p$ and outgoing edges $e_1', \ldots, e_q'$
  is a continuous **function**
  $f_v: D_{e_1}^\omega \times \ldots \times D_{e_p}^\omega \rightarrow D_{e_1}^\omega \times \ldots \times D_{e_q}^\omega$

(A function $f: A \rightarrow B$ is **continuous** if $f(\lim_A a) = \lim_B f(a)$)

Semantics: Kahn networks

A process network defines for each edge $e \in E$ a **unique** sequence X_e.

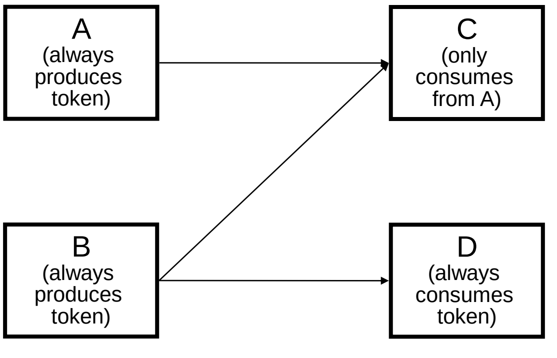
X_e is the least fixed point of the equations

$$(X_{e_1}‘, \ldots, X_{e_q}‘) = f_v(X_{e_1}, \ldots, X_{e_q})$$

for all $v \in V$.

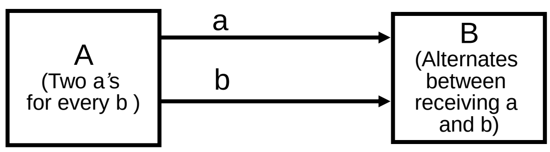
Result is independent of scheduling!

Scheduling Kahn Networks



Problem: run processes with finite buffer

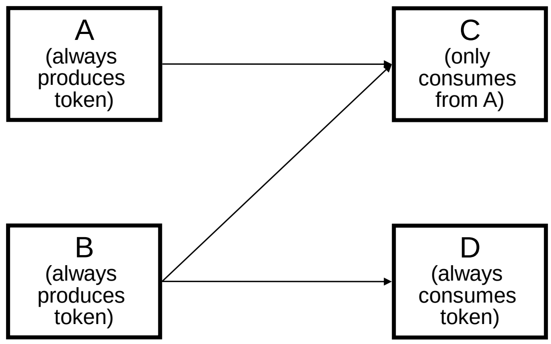
Scheduling may be impossible



Parks' Scheduling Algorithm (1995)

- Set a capacity on each channel
- Block a write if the channel is full
- Repeat
  - Run until deadlock occurs
  - If there are no blocking writes → terminate
  - Among the channels that block writes,
    select the channel with least capacity
    and increase capacity until producer can fire.

Example



Parks' Scheduling Algorithm

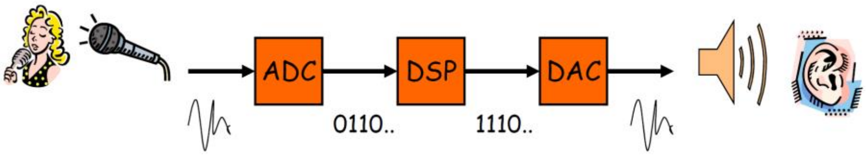
- Whether a Kahn network can execute in bounded memory is undecidable
- Parks' algorithm does not violate this
- It will run in bounded memory if possible, and use unbounded memory if necessary

Disadvantages:

- Requires dynamic memory allocation
- Does not guarantee minimum memory usage
- Scheduling choices may affect memory usage
- Data-dependent decisions may affect memory usage
- Relatively costly scheduling technique
- Detecting deadlock may be difficult

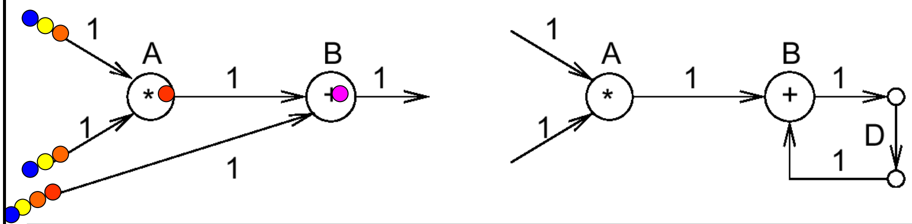
Synchronous data flow

With digital signal processors, data flows at fixed rate



Synchronous data flow (SDF)

- Restriction of Kahn networks (Berkeley, Ptolemy system)
- Asynchronous message passing=
  tasks do not have to wait until output is accepted.
- Synchronous data flow =
  all tokens are consumed at the same time.



SDF model allows static scheduling of token production and consumption.
In the general case, buffers may be needed at edges.

SDF: restriction of Kahn networks

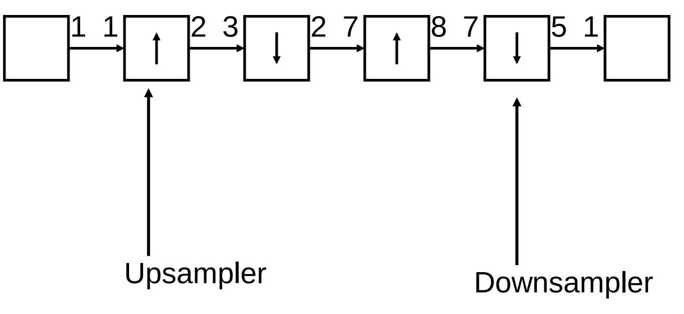
An **SDF graph** is a tuple (V, E, cons, prod, d) where

- V is a set of nodes (activities)
- E is a set of edges (buffers)
- cons: E $\rightarrow$ N number of tokens consumed
- prod: E $\rightarrow$ N number of tokens produced
- d: E $\rightarrow$ N number of initial tokens

d: „delay“ (sample offset between input and output)

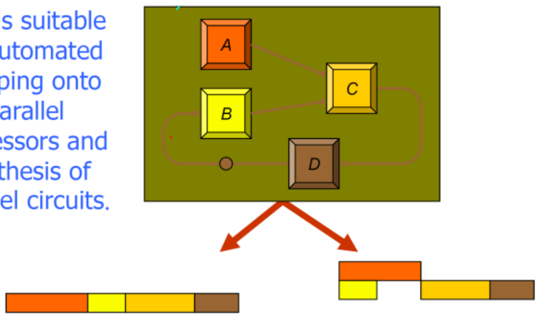
CD-to-DAT rate converter

- Converts a 44.1 kHz sampling rate to 48 kHz



Scheduling SDF models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.

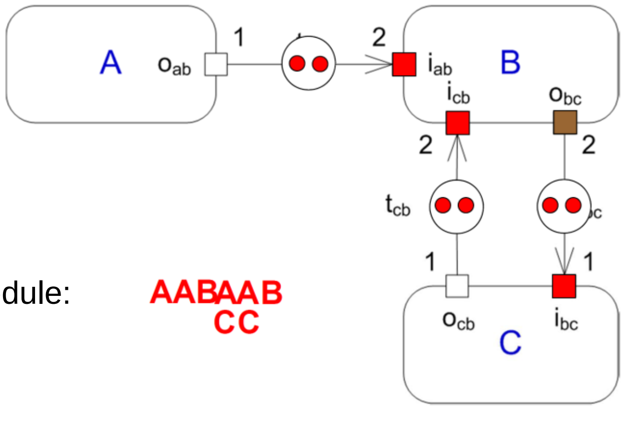


Sequential
periodic admissible sequential schedule (PASS)

Parallel
periodic admissible parallel schedule (PAPS)

(admissible = correct schedule, finite amount of memory required)

SDF example



static schedule: **AABAAB CC**

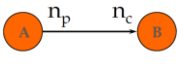
SDF Scheduling Algorithm
Lee/Messerschmitt 1987

1. Establish **relative execution rates**
   - Generate balance equations
   - Solve for smallest positive integer vector **c**
2. Determine **periodic schedule**
   - Form an arbitrarily ordered list of all nodes in the system
   - Repeat:
     - For each node in the list, schedule it if it is runnable, trying each node once
     - If each node has been scheduled $\mathbf{c}_n$ times, stop.
     - If no node can be scheduled, indicate deadlock.

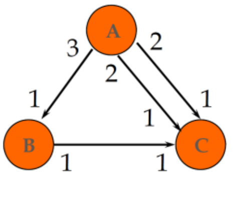
Source: Lee/Messerschmitt, Synchronous Data Flow (1987)

Balance equations

- Number of produced tokens must equal number of consumed tokens on every edge

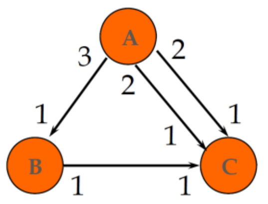


- Firing vector v_S of schedule S: number of firings of each actor in S
- $v_S(A)\ n_p = v_S(B)\ n_c$ must be satisfied on each edge



- $3\ v_S(A) - v_S(B) = 0$
- $v_S(B) - v_S(C) = 0$
- $2\ v_S(A) - v_S(C) = 0$
- $2\ v_S(A) - v_S(C) = 0$

Balance equations



topology matrix

$$M = \begin{vmatrix} 3 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \\ 2 & 0 & -1 \end{vmatrix}$$

the (c, r)th entry in the matrix is the amount of data produced by node c on arc r each time it is involved

- $M\, v_S = 0$
  iff S is periodic
- Full rank (as in this case)
  - no non-zero solution
  - no periodic schedule

Rank of a matrix

The rank of a matrix Γ is the number of linearly independent rows or columns.

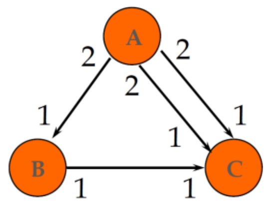
The equation

$$\Gamma q = \vec{0}$$

forms a linear combination of the columns of Γ. Such a linear combination can only yield the zero vector if the columns are linearly dependent.

If Γ has a columns and b rows, the rank cannot exceed $\min(a, b)$.

Balance equations



$$M = \begin{vmatrix} 2 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \\ 2 & 0 & -1 \end{vmatrix}$$

- Non-full rank
- Infinite number of solutions exist:
  any multiple of $| 1 \; 2 \; 2 |^T$ satisfies the balance equations
- ABCBC and ABBCC are valid schedules

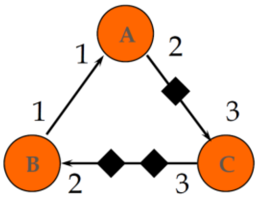
Static SDF scheduling

SDF scheduling theorem (Lee ´86)

- A connected SDF graph with n actors has a periodic schedule iff its topology matrix M has rank n-1
- If M has rank n-1 then there exsts a unique smallest integer solution v_S to
  $M \, v_S = 0$

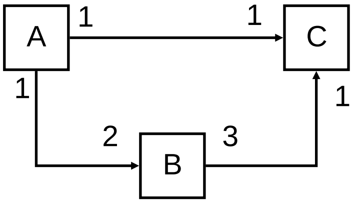
- Rank must be at least n-1 because we need at least n-1 edges (connectedness), each providing a linearly independent row
- Rank is at most n because there are n actors
- Admissibility is not guaranteed, depends on initial tokens on cycles

Admissibility



- No admissible schedule: BACBA, then deadlock…
- Adding one token on A->C makes BACBACBA valid
- Making a periodic schedule admissible is always possible, but changes specification...

An inconsistent system

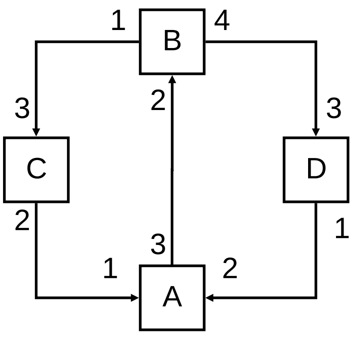


a - c = 0
a – 2b = 0
3b – c = 0

3a – 2c = 0

- No way to execute without an unbounded accumulation of tokens
- Only consistent solution is „do nothing“

PASS example: 1) firing rates



d(AB)=6

$3a - 2b = 0$

$4b - 3d = 0$

$b - 3c = 0$

$2c - a = 0$

$d - 2a = 0$

Solution:

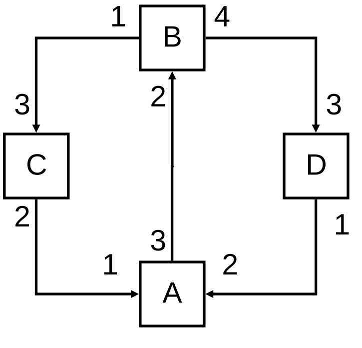
$a = 2c$

$b = 3c$

$d = 4c$

Smallest solution: a=2; b=3; d=4; c=1

PASS example: 2) Simulation



d(AB)=6

Smallest solution:
a=2; b=3; d=4; c=1

Possible schedules:
BBBCDDDDAA
BDBDBCADDA
BBDDBDDCAA
(and many more)

BC... not valid

Completeness theorem

Given an SDF graph with topology matrix M and a positive integer vector v s.t. M v = 0, a PASS of period p = $\mathbf{1}^T$ q exists iff a pass of period N p exists for any integer N.

(proof on blackboard)