Embedded Systems
Message Sequence Charts
Message Sequence Charts

- Message Sequence Charts (MSC) is a language to describe the interaction between a number of independent message-passing instances.
- Defined by ITU (International Telecommunication Union) - Z.120 recommendation
- MSC is
  - a scenario language
  - graphical
  - formal
  - practical
  - widely applicable
In telecommunication industry, MSCs are the first choice to describe example traces of the system under development. MSCs are used throughout the whole protocol life cycle from requirements analysis to testing.

To define longer traces hierarchically, simple MSCs can be composed by operators in high-level MSC (HMSC).

Message Sequence Charts may be used for requirement specification, simulation and validation, test-case specification and documentation of real-time systems.
Message sequence charts (MSC)

- **Graphical means** for representing schedules; time used vertically, “geographical” distribution horizontally.

![Diagram of Message Sequence Charts](image.png)
Basic MSC in a nutshell

MSC diagram

MSC heading

Condition
no predicate logic, merely a label

Output event

Input event

REVIEW

Instance

Message to the environment

Instance end

**Instance**

**Message to the environment**

**Instance end**

**Condition**
no predicate logic, merely a label

**Output event**

**Input event**

**MSC diagram**

**MSC heading**

**Condition**
no predicate logic, merely a label

**Output event**

**Input event**
Timer set and timeout

- User is accepted → forget to push the door

- AC system will detect this through the expiration of the timer → Lock
Preferred situation
• In almost all description/programming/specification languages there is a way to \textit{isolate subparts} of the description in a separate named construct (procedures, functions, classes, packages)

• In MSC there are MSCs which can be \textit{referred} from other MSCs.
• Assume that the scenario where the user is accepted is part of a larger context where there is an automatic door. When the door is unlocked it automatically opens.

• The MSC reference symbol is a box with rounded corners.
**HMSC (High Level MSC)**

```
msc ACsystemOverview
```

**HMSC Start**

**MSC Reference**

**Loop**

\[ \text{User accepted} \]

\[ \text{User rejected} \]

**Connection Point**

\[ \text{Door unlocked} \]

**Condition**

\[ \text{Unlocked_reset} \]

\[ \text{Unlocked_timeout} \]

\[ \text{Unlocked_unclosed} \]

**Alternative**

REVIEW
Data in MSC-2000

- MSC has no data language of its own!

- MSC has parameterized data languages such that
  - fragments of your favorite (data) language can be used
    - C, C++, SDL, Java, ...
  - MSC can be parsed without knowing the details of the chosen data language
  - the interface between MSC and the chosen data language is given in a set of interface functions
Data Flow Models
Data flow modeling

- **Def.**: The process of identifying, modeling and documenting how data moves around an information system.

Data flow 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 model

- Nodes represent transformations
  - May execute concurrently

- Edges represent flow of tokens (data) from one node to another
  - May or may not have token at any given time

- When all of node’s input edges have at least one token, node may fire

- When node fires, it consumes input tokens processes transformation and generates output token

- Nodes may fire simultaneously

- Several commercial tools support graphical languages for capture of dataflow model
  - Can automatically translate to concurrent process model for implementation
  - Each node becomes a process
Philosophy of Dataflow Languages

- Drastically different way of looking at computation
- Von Neumann imperative language style: program counter controls everything
- Dataflow language: movement of data the priority
- Scheduling responsibility of the system, not the programmer
Applications of Dataflow

- signal-processing applications
- Anything that deals with a continuous stream of data
- Becomes easy to parallelize
- Buffers typically used for signal processing applications anyway
Kahn Process Networks

- Proposed by Kahn in 1974 as a general-purpose scheme for parallel programming
- Theoretical foundation for dataflow
- Unique attribute: deterministic
Properties of Kahn process networks (2) REVIEW

- 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.

This is the key beauty of KPNs!
Kahn Process Networks

- Key idea:
  
  Reading an empty channel blocks until data is available

- No other mechanism for sampling communication channel’s contents
- Can’t check to see whether buffer is empty
- Can’t wait on multiple channels at once
Sample parallel program S

Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer in U,V; integer out W);
    Begin integer I ; logical B ;
    B := true ;
    Repeat Begin
    I := if B then wait(U) else wait(V) ;
    print (I) ;
    send I on W ;
    B := not B ;
    End ;
    End ;

(3) Process g(integer in U; integer out V, W);
    Begin integer I ; logical B ;
    B := true ;
    Repeat Begin
    I := wait(U) ;
    if B then send I on V else send I on W ;
    B := not B ;
    End ;
    End ;

(4) Process h(integer in U; integer out V, W);
    Begin integer I ;
    send INIT on V ;
    Repeat Begin
    I := wait(U) ;
    send I on V ;
    End ;
    End ;
Comment : body of mainprogram ;
(5) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,1) ;
    End ;

(1) ... channel declation

processes f, g, h are declared
A Kahn Process

- From Kahn’s original 1974 paper

```
process f(in int u, in int v, out int w)
{
    int i; bool b = true;
    for (;;) {
        i = b ? wait(u) : wait(v);
        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.

What does this do?
A Kahn Process

- From Kahn’s original 1974 paper:

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

- From Kahn’s original 1974 paper:

```c
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 \texttt{u} and alternately copies it to \texttt{v} and \texttt{w}
A Kahn Process

- From Kahn’s original 1974 paper:

```c
process h(in int u, out int v, int init)
{
  int i = init;
  send(i, v);
  for(;;) {
    i = wait(u);
    send(i, v);
  }
}
```

What does this do?

Process sends initial value, then passes through values.
Sample parallel program S

(1) ... channel declaration

processes f, g, h are declared

(6) ... body of the main program:

- calling instances of the processes
- actual names of the channels are bound to the formal parameters
- infix operator \( \text{par} \) \( \rightarrow \) concurrent activation of the processes
**A Kahn System**

- What does this do?

**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

---

**REVIEW**
Determinism

- Process: “continuous mapping” of input sequence to output sequences
- Continuity: process uses prefix of input sequences to produce prefix of output sequences. Adding more tokens does not change the tokens already produced
- The state of each process depends on token values rather than their arrival time
- Unbounded FIFO: the speed of the two processes does not affect the sequence of data values
Synchronous Dataflow (SDF)

Edward Lee and David Messerchmitt, Berkeley, 1987

Ptolemy System

Restriction of Kahn Networks to allow compile-time scheduling

Basic idea: each process reads and writes a fixed number of tokens each time it fires:

```
loop
    read 3 A, 5 B, 1 C …compute…write 2 D, 1 E, 7 F
end loop
```
Synchronous dataflow

- With digital signal-processors (DSPs), data flows at fixed rate.
Synchronous dataflow

- Multiple tokens consumed and produced per firing

- Synchronous dataflow model takes advantage of this
  - Each edge labeled with number of tokens consumed/produced each firing
  - Can statically schedule nodes, so can easily use sequential program model
    - Don’t need real-time operating system and its overhead

- Algorithms developed for scheduling nodes into “single-appearance” schedules
  - Only one statement needed to call each node’s associated procedure
    - Allows procedure inlining without code explosion, thus reducing overhead even more
SDF and Signal Processing

- Restriction natural for multirate signal processing

- Typical signal-processing processes:
  - Unit-rate
    - Adders, multipliers
  - Upsamplers (1 in, n out)
  - Downsamplers (n in, 1 out)
Asynchronous message passing: Synchronous data flow (SDF)

- 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.
Synchronous DataFlow

SDF firing rules:

- **Actor enabling** = each incoming arc carries at least *weight* tokens
- **Actor execution** = atomic consumption/production of tokens by an enabled actor
  - i.e., consume *weight* tokens on each incoming arcs and produce *weight* tokens on each outgoing arc
- **Delay** is an initial token load on an arc.
SDF Example

Static schedule:

AABAAAABCC
Parallel Scheduling of 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 Scheduling Algorithm
Lee/Messerschmitt 1987

1. Establish relative execution rates
   ▪ Generate balance equations
   ▪ Solve for smallest positive integer vector \( \mathbf{q} \)

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 \( q_n \) times, stop.
     • If no node can be scheduled, indicate deadlock.

Source: Lee/Messerschmitt, Synchronous Data Flow (1987)
Multi-rate SDF System

- DAT (digital audio tape) -to-CD rate converter
- Converts a 44.1 kHz sampling rate to 48 kHz
SDF: restriction of Kahn networks

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

- \(V\) is a set of nodes (activities)
- \(E\) is a set of edges (buffers)
- \(\text{cons}: E \to \mathbb{N}\) number of tokens consumed
- \(\text{prod}: E \to \mathbb{N}\) number of tokens produced
- \(d: E \to \mathbb{N}\) number of initial tokens

\(d: \text{”delay“}\) (sample offset between input and output)
Delays

- Kahn processes often have an initialization phase.

- SDF doesn’t allow this because rates are not always constant.

- Alternative: an SDF system may start with tokens in its buffers.

- These behave like delays (signal-processing).

- Delays are sometimes necessary to avoid deadlock.
Example SDF System

- **Finite Impulse Response (FIR) Filter (all single-rate)**

\[
Y_n = x_n c_0 + x_{n-1} c_1 + \ldots + x_{n-(N-1)} c_{(N-1)}
\]
SDF Scheduling

- Schedule can be determined completely before the system runs

- Two steps:

  1. Establish relative execution rates by solving a system of linear equations

  2. Determine periodic schedule by simulating system for a single round
SDF Scheduling

- Goal: a sequence of process firings that:
  - Runs each process at least once in proportion to its rate
  - Avoids underflow
    - no process fired unless all tokens it consumes are available
  - Returns the number of tokens in each buffer to their initial state

- Result: the schedule can be executed repeatedly without accumulating tokens in buffers
Balance equations

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

- Repetitions (or 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 for each edge
Balance equations

- Balance for each edge:
  - $3 \cdot v_S(A) - v_S(B) = 0$
  - $v_S(B) - v_S(C) = 0$
  - $2 \cdot v_S(A) - v_S(C) = 0$
  - $2 \cdot v_S(A) - v_S(C) = 0$
Balance equations

- \( M \cdot v_S = 0 \)
  - iff \( S \) is periodic
- Full rank (as in this case)
  - no non-zero solution
  - no periodic schedule
  - (too many tokens accumulate on A->B or B->C)

![Diagram of network with nodes A, B, C and arcs](image)

**topology matrix**

\[
M = \begin{bmatrix}
3 & -1 & 0 \\
0 & 1 & -1 \\
2 & 0 & -1 \\
2 & 0 & -1
\end{bmatrix}
\]

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
Balance equations

- Non-full rank
  - infinite solutions exist
- Any multiple of \( v_s = [1 \ 2 \ 2]^T \) satisfies the balance equations
- ABCBC and ABBCC are minimal valid schedules
Static SDF scheduling

- Main 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 exists 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 (connected-ness), providing each a linearly independent row
  - Admissibility is not guaranteed, and depends on initial tokens on cycles
Admissibility of schedules

- 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

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

\[
\begin{align*}
    a - c &= 0 \\
    a - 2b &= 0 \\
    3b - c &= 0 \\
    3a - 2c &= 0
\end{align*}
\]
Calculating Rates

- Each arc imposes a constraint

\[
\begin{align*}
3a - 2b &= 0 \\
4b - 3d &= 0 \\
b - 3c &= 0 \\
2c - a &= 0 \\
d - 2a &= 0
\end{align*}
\]

Solution:
\[
\begin{align*}
a &= 2c \\
b &= 3c \\
d &= 4c
\end{align*}
\]
Scheduling Example

- Theorem guarantees any valid simulation will produce a schedule

Possible schedules:

BBBCDDDDAA
BDBDVCADDA
BBDBDDBDCAA

BC … is not valid
SDF Compiler

Task for an SDF compiler:

- Allocation of memory for the passing of data between nodes
- Scheduling of nodes onto processors in such a way that data is available for a block when it is invoked

Assumptions on the SDF graph:

- The SDF graph is nonterminating and does not deadlock
- The SDF graph is connected

Goal:

- Development of a periodic admissible parallel schedule (PAPS)
- or a periodic admissible sequential schedule (PASS)

(admissible = correct schedule, finite amount of memory required)
Does a PASS exist?

- The SDF graph is described by the topology matrix

\[
M = \begin{bmatrix}
c & -e & 0 \\
d & 0 & -f \\
0 & -i & g
\end{bmatrix}
\]

- The entry of row \( r \) and column \( c \) is the number of tokens produced (positive number) or consumed (negative number) by node \( c \) on arc \( r \).

- Connections to the outside world are not considered.
Does a PASS exist?

- A PSS (periodic sequential schedule) exists if rank \((M) = s -1\), where \(s\) is the number if nodes in a graph.

- There is a \(v\) such that \(Mv = O\) where \(O\) is a vector full of zeros. \(v\) describes the number of firings in each scheduling period.
A PASS exists

- The rank of the matrix $M = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$ is $s - 1 = 2$ and $v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

- A valid schedule is $\Phi = \{a, b, c, c\}$, but not $\Phi = \{b, a, c, c\}$

- The maximum buffer sizes for the arcs are $b = \langle 1, 2, 2 \rangle$
A PASS does not exist

- The graph has sample rate inconsistencies.
- A schedule for the graph will result in unbounded buffer sizes.
- No PASS can be found (\( \text{rank } (M) = s = 3 \)).
Assumption: Block 1 : 1 time unit
Block 2 : 2 time units
Block 3 : 3 time units

Trivial Case - All computations are scheduled on same processor
PAPS

- The performance can be improved, if a schedule is constructed that exploits the potential parallelism in the SDF-graph. Here the schedule covers one single period.

![Single Period Schedule](image)
The performance can be further improved, if the schedule is constructed over two periods.