Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
ES Design

Hardware components

Software Components

Verification and Validation

Hardware
Models, Languages and Tools

Language $L_1$ → Model $M_1$ → Tool → Language $L_2$ → Model $M_2$

State machine → Sequent. program → Data-flow → Concurrent processes

Verilog → C/C++ → Java → VHDL

Implementation A → Implementation B → Implementation C
Elements of model of computation

- **Language:**
  - Synchronous vs. asynchronous languages

- **State**

- **Decidability**
  - Can a property be determined in a finite amount of time?
### Modes of Communication

<table>
<thead>
<tr>
<th></th>
<th>Transmitters</th>
<th>Receivers</th>
<th>Buffer Size</th>
<th>Blocking Reads</th>
<th>Blocking Writes</th>
<th>Single Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsynchronized</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Read-Modify-write</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Unbounded FIFO</td>
<td>one</td>
<td>one</td>
<td>unbounded</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bounded FIFO</td>
<td>one</td>
<td>one</td>
<td>bounded</td>
<td>no</td>
<td>maybe</td>
<td>yes</td>
</tr>
<tr>
<td>Single Rendezvous</td>
<td>one</td>
<td>one</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Multiple Rendezvous</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Communication

- Message Passing
  - Non-blocking
  - Blocking
  - Extended rendezvous

- Shared memory

```
process a {
  ..
  P(S) //obtain lock
  ..  // critical section
  V(S) //release lock
}

process b {
  ..
  P(S) //obtain lock
  ..  // critical section
  V(S) //release lock
}
```
Implantable Cardioverter Defibrillator

- **Function:**
  - helps stop dangerously fast heart rhythms in the heart's lower chambers
  - treats sudden cardiac arrest and restores a normal heartbeat.

- **Guarantee device works 100% of the time for at least 7-10yrs**
Models of Computation

- **State Machine Models**
  - FSM, StateCharts, CFSM, SDL
- **Petri nets**
- **Communicating Processes**
  - Kahn processes, Communicating Sequential Processes
- **Ada**
- **Dataflow models**
  - DFG, SDFG
- **Discrete Event Systems**
  - VHDL, Verilog, SystemC, SpecC
- **Synchronous languages**
  - Cycle based models, Esterel, Lustre
“Move the elevator either up or down to reach the requested floor. Once at the requested floor, open the door for at least 10 seconds, and keep it open until the requested floor changes. Ensure the door is never open while moving. Don’t change directions unless there are no higher requests when moving up or no lower requests when moving down…”
Elevator controller using a sequential program model

You might have come up with something having even more if statements.

Sequential program model

```c
void UnitControl()
{
    up = down = 0; open = 1;
    while (1)
    {
        while (req == floor);
        open = 0;
        if (req > floor) { up = 1; }
        else { down = 1; }
        while (req != floor);
        up = down = 0;
        open = 1;
        delay(10);
    }
}
```

```c
void RequestResolver()
{
    while (1)
    {
        req = ...
    }
}
```

```c
void main()
{
    Call concurrently: UnitControl() and RequestResolver()
}
```

Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

System interface
Classical automata

- **Moore-automata:**
  \[ O = H(S); \quad S^+ = f(l, S) \]

- **Mealy-automata**
  \[ O = H(l, S); \quad S^+ = f(l, S) \]
Finite-state machine (FSM) model

*UnitControl* process using a state machine

```
req > floor
GoingUp
!
!(req > floor)
!
!(timer < 10)

req < floor
Idle
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!

DoorOpen

u,d,o,t = 0,0,1,1

u is up, d is down, o is open

t is timer_start
```

```
req > floor
GoingDn
req < floor

Idle
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
![Image: Finite-state machine model]
Models of Computation

- State Machine Models
  - FSM, StateCharts, CFSM, SDL
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
## Models of computation comparison

<table>
<thead>
<tr>
<th>Communication/local computations</th>
<th>Shared memory</th>
<th>Message passing</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Communicating finite state machines</a></td>
<td>StateCharts</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Data flow</td>
<td>(Not useful)</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Petri nets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete event (DE) model</td>
<td><a href="#">VHDL*</a>, <a href="#">Verilog*</a>, <a href="#">SystemC*</a>, …</td>
<td>Only experimental systems, e.g. distributed DE in Ptolemy</td>
</tr>
<tr>
<td>Von Neumann model</td>
<td>C, C++, Java</td>
<td>C, C++, Java &amp; libraries, ADA</td>
</tr>
</tbody>
</table>

* Classification based on implementation with centralized data structures
Languages in Chap 2 of Text

- SDL
  FSM+asynchronous message passing

- StateCharts
  FSM+shared memory

- ADA
  von Neumann execution+synchronous message passing

See also
- Work by Edward A. Lee on Ptolomey: environment for simulating multiple models of computation http://ptolemy.berkeley.edu/
**UnitControl with FireMode**

- **FireMode**
  - When fire is true, move elevator to 1\textsuperscript{st} floor and open door
StateCharts: Hierarchy
Back to Elevator Example
StateCharts: Default state
StateCharts: History
History & default state

same meaning
StateCharts: Concurrency

![Statechart Diagram]

- **State Machines**:
  - **Lwait**: on the line-monitoring.
  - **Kwait**: on the key-monitoring (excl. on/off).

- **Transitions**:
  - **Ring**: Lwait to Lproc.
  - **Hangup** (caller): Lwait to Lwait.
  - **Key Pressed**: Kwait to Kproc.
  - **Done**: Kproc to Kproc.

- **Events**:
  - **Key-On**: Lproc to Lwait.
  - **Key-Off**: Lwait to Lproc.
  - **Off**: Kproc to Kproc.
Conditional Transitions
StateCharts: Timers

\[ \text{timeout} \rightarrow \text{20 ms} \rightarrow a \]
Example: Answering machine

Lproc

4 s

play text

timeout

lift off

talk

return (callee)

dead

8 s record

timeout

beep

silent
StateCharts: edge labels

- **Events:**
  - Exist only until the next evaluation of the model
  - Can be either internally or externally generated

- **Conditions:**
  - Refer to values of variables that keep their value until they are reassigned

- **Reactions:**
  - Can either be assignments for variables
  - or creation of events

- **Example:**
  - `service-off [not in Lproc] / service:=0`
Propagations and Broadcasts
StateCharts: Simulation

Status = values of all variables + set of events + current time
Step = execution of the three phases

Three phases:
1. Effects of external changes on events and conditions are evaluated,
2. The set of transitions to be made in the current step and right hand sides of assignments are computed,
3. Transitions become effective, variables obtain new values.
StateCharts Simulation Example

\[ e/a := b \]

\[ e/b := a \]

\[ /a := 1; \ b := 0 \]
In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.
Statecharts – Example 1

Statechart Example

Equivalent FSM Representation
Statecharts – Example 2

Statechart Example

Equivalent FSM Representation

Tajana Simunic Rosing
StateCharts: Application Examples

- Power converter system for trams, metros & trains
  - System-level modeling and automated code generation
  - Guarantee latencies less than 10 microseconds
  - Cut development time by 50%
  - Time from design completion to first prototype down to 1 hr from 3 months
  - Defect free code automatically generated for a number of RTOS implementations
StateCharts: Application Examples

- Development of defibrillator and pacemaker technology
  - Model system behavior before requirements are finalized
  - Check that SW provides mathematically consistent representation of the product’s system – correct and unambiguous representation of model behavior
  - More accurate and extensive verification – guarantee device works 100% of the time for at least 7-10 yrs
  - Cut product verification costs by 20%
  - 15-20% overall cost reduction per projects on future development
StateCharts: Application Examples

- Jet engine electronic controller design
  - System level specification and consistency check
  - Construction of on-screen simulation of the cockpit display
  - Evaluate correctness in normal and faulty operation modes
- Project so successful that now StateCharts are used for:
  - Independent overspeed protection system
  - Thrust reverse control system
  - Engine fuel control system

By BMW
StateCharts: Summary

- Hierarchy
- AND- and OR-super states
- Default state, History
- Timing behavior
- State oriented behavior
- Edge labels
- Concurrency
- Synchronization & communication
  - Broadcast, shared memory
- Simulation
- Cross compiling
Models of Computation

- State Machine Models
  - FSM, StateCharts, CFSM, SDL
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
Co-design Finite State Machines (CFSMs)

- CFSM is FSM extended with:
  - Data handling
  - Asynchronous communication

- CFSM has
  - FSM part
  - Data computation part

- Interacting CFSMs communicate through broadcast events
Example CFSM Specification

- State of a CFSM includes those events that are at the same time input and output for it
  - non-zero reaction time implies storage capability

```
*Key = On → *Start

*Key = Off or
*Belt = On →

*End=5 → *Alarm = On

*End=10 or
*Belt = On or
*Key = Off →
*Alarm = Off

*End=10 or
*Belt = On or
*Key = Off →
*Alarm = Off
```
Network of CFSMs

- GALS model, no hierarchy
Data modem design

- What are the design goals?
- What should the design process be like?
SDL is an example of a model of computation based on asynchronous message passing communication.

appropriate for distributed systems
SDL

- Designed for specification of distributed systems.
  - Dates back to early 70s, formal semantics defined in the late 80s, updates from 1984 to 1999 by International Telecommunication Union (ITU)
  - Provides textual and graphical formats
- Just like StateCharts, it is based on the CFSM model of computation; each FSM is called a process, but uses message passing instead of shared memory for communication
- Supports operations on data
SDL-representation of FSMs/processes

Process P1

state
input
output
Communication among SDL-FSMs

- Communication between FSMs (or “processes“) is based on **message-passing**, assuming a potentially indefinitely large FIFO-queue.

- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).
Determinate?

Let tokens be arriving at FIFO at the same time:

Order in which they are stored is unknown:

All orders are legal so simulators can show different behaviors for the same input, all of which are correct.
Operations on data

- Variables can be declared locally for processes.
- Their type can be predefined or defined in SDL itself.
- SDL supports abstract data types (ADTs)

```
DCL
Counter Integer;
Date String;
```

```
Counter := Counter + 3;
```

```
(1:10) (11:30) ELSE
```
Process interaction diagrams

- Interaction between processes can be described in process interaction diagrams, which are a special case of block diagrams.
- In addition to processes, these diagrams contain channels and declarations of local signals.

```
BLOCK B1
Signal A::B;
```

```
process B1
[B]
Sw1
[A]
Sw2

[A,B]

process P2
```
Hierarchy in SDL

- Process interaction diagrams can be included in **blocks**. The root block is called **system**.

Processes cannot contain other processes, unlike in StateCharts.
Timers

- Timers can be declared locally
- Elapsed timers put signal into queue, but are not necessarily processed immediately
- RESET removes a timer also from FIFO-queue.
Description of network protocols
Larger example: vending machine

Machine sells pretzels, chips, cookies, & doughnuts

Accepts nickels, dime, quarters, and half-dollars

It is not a distributed application.

Overall view of vending machine

System VendingMachine

Ccoins

[nickel, dime, quarter, half]

CoinInterface

Cadd

[add]

Ccoinctrl

[rej_further_coins, accept_coins]

Crequest

[pur_pretzel, pur_chip, pur_cookie, pur_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]

Creject

[reject_coin]

DecodeRequests

CamontDisplay

[amount_entered]

Cemptydisplay

[pretzel_empty, chip_empty, cookie_empty, doughnut_empty]

CspitPurchased

[spit_pretzel, spit_chip, spit_cookie, spit_doughnut]

CexaktDisplay

[exact_only]

CspitChange

[spit_nickel, spit_dime]

ChangeInterface

[spit_change]

Cchange

SIGNAL

[dime, nickel, quarter, half, pur_pretzel, pur_cookie, pur_doughnut, pur_chip, add(int), spit_change(int), amount_entered(int), reject_further_coins, exact_only, accept_coins, reject_coins, spit_dime, spit_nickel, pretzel_empty, spit_pretzel, chip_empty, spit_chip, cookie_empty, spit_cookie, doughnut_empty, spit_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]

SYNTAXE items=INTEGER

CONSTANTS 0:7

ENDSYNTAXE items;

SYNTAXE int=INTEGER

CONSTANTS 0:127

ENDSYNTAXE int;
Block DecodeRequests

CONNECT Cadd AND Radd;
CONNECT Ccoinctrl AND Rcoinctrl;
CONNECT Cchange AND Rchange;
CONNECT CAmountDisplay AND RamountDisplay;
CONNECT Crequest AND Rpretzel,Rchip,Rcookie,
    Rdoughnut;
CONNECT CemptyDisplay AND Rpretzel_e,Rchip_e,
    Rcookie_e,Rdoughnut_e;
CONNECT CspitPurchased AND Rpretzel_s,
    Rchip_s,Rcookie_s,Rdoughnut_s;

SYNONYM PRETZEL int=50
SYNONYM PCHIP int=15;
SYNONYM PCOOKIE int=55;
SYNONYM PDOUGHNUT int=60;
SYNONYM PMAX int=60;
SYNONYM NITEMS items=7;

SIGNAL sub(int);
Process ChipHandler

DCL nchip items:=NITEMS;

VIEWED current int;

pur_wait

pur_chip

VIEW(current) >= PCHIP

no

pur_wait

yes

sub(PCHIP)

nchip:= nchip-1;

spit_chip

nchip=0

yes

no

pur_wait

chip_empty

empty

reload_chip

nchip:=NITEMS

pur_wait
SDL: Real World Example

- ADSL design
  - Ideal language for telecom design; communication between components and their different states of operation can be easily modeled with SDL
  - Object orientation and automatic code generation significantly simplified system design verification
  - Early testing done on SDL – significant savings in the cost of expensive test equipment
SDL summary

- FSM model for the components,
- Non-blocking message passing for communication,
- Implementation requires bound for the maximum length of FIFOs; may be very difficult to compute,
- Not necessarily determinate
- Timer concept adequate just for soft deadlines,
- Limited way of using hierarchies,
- Limited programming language support,
- No description of non-functional properties,
- Excellent for distributed applications (used for ISDN),
- Commercial tools available (see http://www.sdl-forum.org)
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
Petri net definitions

A Petri net is a 5-tuple, \( PN = (P, T, F, W, M_0) \) where:

- \( P = \{p_1, p_2, \cdots, p_m\} \) is a finite set of places,
- \( T = \{t_1, t_2, \cdots, t_n\} \) is a finite set of transitions,
- \( F \subseteq (P \times T) \cup (T \times P) \) is a set of arcs (flow relation),
- \( W: F \rightarrow \{1, 2, 3, \cdots\} \) is a weight function,
- \( M_0: P \rightarrow \{0, 1, 2, 3, \cdots\} \) is the initial marking,
- \( P \cap T = \emptyset \) and \( P \cup T \neq \emptyset \).

A Petri net structure \( N = (P, T, F, W) \) without any specific initial marking is denoted by \( N \).

A Petri net with the given initial marking is denoted by \( (N, M_0) \).

---

<table>
<thead>
<tr>
<th>Input Places</th>
<th>Transition</th>
<th>Output Places</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Event</td>
<td>Postconditions</td>
</tr>
<tr>
<td>Input data</td>
<td>Computation step</td>
<td>Output data</td>
</tr>
<tr>
<td>Input signals</td>
<td>Signal processor</td>
<td>Output signals</td>
</tr>
<tr>
<td>Resources needed</td>
<td>Task or job</td>
<td>Resources released</td>
</tr>
<tr>
<td>Conditions</td>
<td>Clause in logic</td>
<td>Conclusion(s)</td>
</tr>
<tr>
<td>Buffers</td>
<td>Processor</td>
<td>Buffers</td>
</tr>
</tbody>
</table>

Source: Murata’89
Petri net – Train tracks model

- Train wanting to go right
- Train going to the right
- Track available
- Train going to the left
Concurrency, Causality, Choice
Concurrency, Causality, Choice
Concurrency, Causality, Choice

Causality, sequencing

\[ t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_6 \]
Concurrency, Causality, Choice
Concurrency, Causality, Choice

[Diagram showing concurrency and causality with transitions t1, t2, t3, t4, t5, and t6 with a note on choice and conflict]
Petri nets: confusion 😊

- Concurrency & conflict lead to confusion....
  - Symmetric
  - Asymmetric

Source: Murata’89
Conflict for resource „track“

- Train wanting to go right
- Train going to the right
- Track available
- Train going to the left
Communication Protocol
Communication Protocol
Communication Protocol
Communication Protocol
Communication Protocol

![Communication Protocol Diagram]

- P1
  - Receive Ack
  - Send msg
  - Receive msg

- P2
  - Send Ack
  - Receive Ack
Communication Protocol
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer Problem
Producer-Consumer with Priority

- Modeling synchronization control when sharing resources
- Multiprocessor CPUs, distributed systems
Petri net definitions

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $W: F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function,
- $M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking,
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by $N$.

A Petri net with the given initial marking is denoted by $(N, M_0)$.

<table>
<thead>
<tr>
<th>Input Places</th>
<th>Transition</th>
<th>Output Places</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Event</td>
<td>Postconditions</td>
</tr>
<tr>
<td>Input data</td>
<td>Computation step</td>
<td>Output data</td>
</tr>
<tr>
<td>Input signals</td>
<td>Signal processor</td>
<td>Output signals</td>
</tr>
<tr>
<td>Resources needed</td>
<td>Task or job</td>
<td>Resources released</td>
</tr>
<tr>
<td>Conditions</td>
<td>Clause in logic</td>
<td>Conclusion(s)</td>
</tr>
<tr>
<td>Buffers</td>
<td>Processor</td>
<td>Buffers</td>
</tr>
</tbody>
</table>

Tajana Simunic Rosing

Source: Murata’89
Petri Net Properties

- **Behavioral**
  - Reachability
    - Marking $M$ reachable from marking $M_0$
  - $k$ - Boundedness
    - Number of tokens in each place does not exceed finite number $k$
    - Safe if 1-bounded
  - Liveness
    - Can fire any transition of the net – related to absence of deadlocks

- **Structural**
  - Controlability
    - Any marking can be reached from any other marking
  - Structural boundedness
  - Conservativeness – weighted sum of tokens constant
PN Properties - Reachability

\[ M_0 = (1,0,1,0) \]
\[ M = (1,1,0,0) \]

\[ M_0 = (1,0,1,0) \]
\[ M_1 = (1,0,0,1) \]
\[ t_2 \]
\[ M = (1,1,0,0) \]
PN Properties – Deadlock-free
PN Properties – Deadlock-free
Petri Net Liveness

- **L0-Live (dead):** a particular transition can never fire
  - In the figure below, $t_0$ can never fire (see reachability graph)
Petri Net Liveness

- **L1-Live**: A particular transition can fire at least once for any firing sequence.
  - In the figure below, $t_1$ can only fire once, for any firing sequence (see reachability graph).
Petri Net Liveness

- **L2-Live**: a particular transition can fire $k$ times for a particular firing sequence, for any $k$.
  - In the figure below, $t_2$ can only fire once, twice, thrice, etc, for different firing sequences.
Petri Net Liveness

- **L3-Live**: a particular transition can fire infinitely in a particular firing sequence.
  - In the figure below, $t_3$ can fire infinitely for the firing sequence $t_3, t_3, t_3, t_3,...$
  - Note that the number of times $t_1$ and $t_2$, fire is finite for any firing sequence.
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Boundedness

Unbounded
PN Properties - Boundedness
PN Properties - Boundedness
PN Properties - Conservation

Not conservative
PN Properties - Conservation
PN Properties - Conservation
Petri Nets - Analysis

- Structural
  - Incidence matrix
- State Space Analysis techniques
  - Coverability Tree
  - Reachability Graph
PN Properties – Analysis

Incident Matrix

State Equations
Petri Nets – Coverability Tree
Example

- Assume
  - $e = 6$
  - $Mo = [0 \ 12]$
- Can we reach $M = [6 \ 0]$ from $Mo$?
- Can it be statically scheduled?
- What size buffers are needed at $P1$ & $P2$?
Petri nets: Applications

- Model, simulate and analyze networking protocols (e.g. TCP, Ethernet, etc)
- Model, simulate and analyze complex network elements (e.g. router, switch, optical mux); check their logical behavior
- Design and analyze network performance and AoS with logical models of traffic generators, protocols and network elements
- Study network behavior characteristics (e.g. throughput, blocking probability etc)
Petri nets in practice

Example apps:
- TCP performance
- Security system design and automated code generation
- MAC design
- ....

Tajana Simunic Rosing
Petri Nets - Summary

- PN Graph
  - places (buffers), transitions (action), tokens (data)
- Firing rule
  - Transition enabled if enough tokens in a place
- Properties
  - Structural (consistency, structural boundedness)
  - Behavioral (reachability, boundedness, etc.)
- Analysis techniques
- Applications
  - Modeling of resources, mutual exclusion, synchronization
Modeling Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.
Where we are now

- Models:
  - FSMs, CFSM, StateCharts, SDL, Petri nets
- HW#1 assigned
- Project part 1 due today
  - Each student uploads their test.o
- Plan for today:
  - Modeling continued: DFGs, SDFs, etc.
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Communicating Sequential Processes, Ada, Kahn processes
- Dataflow models
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- HW Models
- Unified Modeling Language (UML)
Process

- A sequential program
  - Executes concurrently with other processes
- Basic operations on processes
  - Create & terminate, suspend & resume, join
- Process communication
  - Shared memory (and mutexes)
  - Message passing
  - Rendezvous

![Diagram of elevator system interface with buttons and request resolver.]

System interface

- Unit Control
- Request Resolver
- Buttons inside elevator: up1, up2, dn2, up3, dn3, ..., dnN
- Up/down buttons on each floor: b1, b2, bN
- System interface connections: req, floor, open, up, down
Concurrent processes & implementation

Heartbeat Monitoring System

Task 1: Read pulse
If pulse < Lo then
  Activate Siren
If pulse > Hi then
  Activate Siren
Sleep 1 second
Repeat

Task 2: If B1/B2 pressed then
  Lo = Lo +/- 1
If B3/B4 pressed then
  Hi = Hi +/- 1
Sleep 500 ms
Repeat

Set-top Box

Task 1: Read Signal
  Separate Audio/Video
  Send Audio to Task 2
  Send Video to Task 3
Repeat

Task 2: Wait on Task 1
  Decode/output Audio
Repeat

Task 3: Wait on Task 1
  Decode/output Video
Repeat

Input Signal

Video

Audio

Processor A
Processor B
Processor C
Processor D
General Purpose Processor
General Purpose Processor

Communication Bus

(a) (b) (c)
ADA-rendez-vous

```ada
task screen_output is
  entry call_ch(val:character; x, y: integer);
  entry call_int(z, x, y: integer);
end screen_out;
task body  screen_output is
...
  select
    accept call_ch ... do ..
  end call_ch;
or
  accept call_int ... do ..
  end call_int;
end select;

Sending a message:
begin
  screen_out.call_ch('Z', 10, 20);
exception
  when tasking_error =>
    (exception handling)
end;
```
Task graphs

Nodes are a "program" described in some programming language.

Sequence constraint

Diagram:
- Nodes labeled $T_1$, $T_2$, $T_3$, $T_4$, $T_5$ connected in a sequence.

121
Task graphs - Timing

Arrival time \( (0,7] \)  

Deadline \( T_1 \)

Arrival time \( (1,8] \)  

Deadline \( T_2 \)

Arrival time \( (3,10] \)  

Deadline \( T_3 \)
Task graphs - I/O

\[
\begin{align*}
T_1 & \rightarrow T_2 & T_2 & \rightarrow T_3 & T_3 & \rightarrow T_4 & T_4 & \rightarrow T_5 & T_5 & \rightarrow \text{output} \\
T_1 & \rightarrow \text{input} & & & & & & & & & & \end{align*}
\]
Task graphs - Shared resources
Task graphs - Periodic schedules

\[ \ldots J_{n-1} \rightarrow J_n \rightarrow J_{n+1} \]

.. infinite task graphs
Task graphs - Hierarchy

\[
\begin{align*}
T_1 & \rightarrow T_2 & T_2 & \rightarrow T_5 \\
T_3 & \rightarrow T_4 & T_3 & \rightarrow T_1 & T_4 & \rightarrow T_5
\end{align*}
\]
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Communicating Sequential Processes, Ada
- Dataflow models
  - Kahn processes
  - DFG, SDFG
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Petri nets
- HW Models
- Unified Modeling Language (UML)
Kahn process network

KPN - executable task graphs
Communication is via infinitely large FIFOs
Nonblocking write, blocking read => DETERMINATE

Important properties of KPN:
- Continuous: Output signals can be gradually produced; never have to consume all input to produce some output
- Monotonic: Output depends on input but doesn’t change previous output

Continuous monotonic processes => ITERATIVE
Petri net model of a Kahn process

- KPNs are deterministic:
  - Output determined by
    - Process, network, initial tokens
Dataflow Process Networks

- Dataflow:
  - Maps input tokens to output tokens
  - Outputs function of current inputs
    - No need to keep state on suspend/resume
  - Scheduling for resource sharing

Z = (A + B) * (C - D)

Nodes with arithmetic transformations

Nodes with more complex transformations
Dataflow process examples

- HW resource scheduling
  - Constraints: 2 mult, 2 ALU
  - List scheduler used
  - Additional constraints
    - Performance
    - Power
    - Area etc.
Synchronous dataflow
SDF notation

- Nodes have rates of data production or consumption.
- Edges have delays.
  - Delays do not change rates, only the amount of data stored in the system at startup.
What Latency & Sample Period can a SDFG achieve?

- $T_L = \max(I-O \text{ path}, D-O \text{ path})$
- $T_S = \max(D-D \text{ path}, I-D \text{ path})$
SDFG can be Transformed to Affect $T_L$ & $T_S$
SDF examples
SDF Scheduling

- By building a set of “flow and conservation” equations

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

Solution: \(a = 2c; \ b = 3c; \ d = 4c\)

Possible schedules:
- BBBCDDDDDDAA
- BDBDBBCADDA
- BBDDDBDCAA
- …
Dataflow process examples

- Design of a first 802.11b WLAN card
  - New product – combines RF, MAC and PHY; lots of DSP
  - Implemented on an ASIC
  - Previous design hand coded VHDL

- System level design with COSSAP by Synopsys
  - Explore architectural trade-offs – what in gates, what on DSP
  - Scheduling trade-offs: latency, area, propagation delay between clocks, and vendor library

- Results:
  - Reduced time to market by a factor of TWO!
  - Architectural exploration within 10% of actual measurements in terms of timing and area
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous reactive languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Reactive Synchronous Languages

- Assumptions
  - Instantaneous reactions
  - Discrete event
  - Static

- Cycle based models
  - Excellent for (single) clocked synchronous circuits

- Control flow oriented (imperative) languages
  - Esterel

- Data flow languages
  - Lustre, Signal

- Deterministic behavior

- Simulation, software and hardware synthesis, verification
Lustre example
Reactive Synchronous Models: Esterel Statements

- Emit S
- Present S then p else q end
- Pause
- P; Q, P||Q
- Loop p end
- Await S
- Abort p when S
- Suspend p when S
- Sustain S = (loop emit S; pause end)
Abort Statement

```
abort
  pause;
  pause;
  emit A
when B;
emit C
```

- **Normal Termination**
  - B not checked in first cycle (like await)

- **Aborted termination**
  - emit A preempted

- **Aborted termination**
Esterel Examples

emit A;
emit B;
pause;
loop
  present C then emit D end;
  present E then emit F end;
pause;
end

C   C
E   E
A   D   D
B   F   F
Esterel Examples

```
[  
    await A; emit C  
||  
    await B; emit D  
];  
emit E
```

Diagram:

```
A ---- B
  |
  |
C ---- D ---- E
```
Time in Esterel

- Global clock with precise control over when events appear
  - At every tick: read inputs, compute, output

- Statements
  - A bounded number in one cycle
    - Emit, present, loop
  - Take multiple cycles:
    - Pause, await, sustain

- Causality analysis
  - Deterministic & non-contradictory
Esterel Application Examples

- TI used Esterel to automatically synthesize full coverage tests for a safety-critical design
  - Showed functional coverage covered only 30% of the design, with Esterl 100% covered

- Airbus
  - Significant decrease in errors due to increase in automated code generation (40-70%)
    - e.g. fly by wire controls & automatic flight control 70%, display computer 50%, warning & maintenance computer 40%
  - Major increase in productivity

Computer Aided Specifications

<table>
<thead>
<tr>
<th>Aircraft</th>
<th>A310 (70')</th>
<th>A320 (80')</th>
<th>A340 (90') *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of digital units</td>
<td>77</td>
<td>102</td>
<td>115</td>
</tr>
<tr>
<td>Volume of on-board software in MB 4</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Errors found per 100 KB</td>
<td>A few hundred</td>
<td>A few dozen</td>
<td>Less than 10</td>
</tr>
</tbody>
</table>
Esterel Summary

- Reactive synchronous language
- Control flow oriented
- Imperative syntax
- Synchrony assumption useful for safety critical embedded systems
  - Convert timing relations to causal ordering
  - Used in verification
    - e.g. TI, Airbus
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- **Discrete Event Systems**
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Discrete Events

- Notion of time is fundamental: global order
  - events are objects which carry ordered time info
  - there is a casual relationship between events
- DE simulator maintains global event queue
  - Verilog, VHDL
- Expensive - ordering tame stamps can be time consuming
- Large state & Low Activity => Effective simulation
- Simultaneous events lead to non-determinacy
  - require complex conflict resolution schemes
  - e.g. delta delays
Simultaneous Events in the Discrete Event Model

B has 0 delay

B has delta delay
VHDL: Entities

entity full_adder is
  port(a, b, carry_in: in Bit; -- input ports
       sum, carry_out: out Bit); -- output ports
end full_adder;
architecture structure of full_adder is
    component half_adder
        port (in1, in2: in Bit; carry, sum: out Bit);
    end component;
    component or_gate
        port (in1, in2: in Bit; o: out Bit);
    end component;
signal x, y, z: Bit;  -- local signals
begin  -- port map section
    i1: half_adder port map (a, b, x, y);
    i2: half_adder port map (y, carry_in, z, sum);
    i3: or_gate port map (x, z, carry_out);
end structure;

architecture behavior of full_adder is
begin
    sum <= (a xor b) xor carry_in after 10 Ns;
    carry_out <= (a and b) or (a and carry_in) or (b and carry_in) after 10 Ns;
end behavior;
VHDL: signal strengths

- 'X'
- '0'
- '1'
- 'W'
- 'L'
- 'H'
- 'W'
- 'I'
- 'h'
- 'Z'

- strongest
- medium strength
- pre-charged
- weakest
Processes model HW parallelism

```vhdl
process
begin
  a <= b after 10 ns
end
```

Waits:

```vhdl
wait until signal list;
wait until a;
wait until condition;
  wait until c='1';
wait for duration;
  wait for 10 ns;
wait; suspend indefinitely
wait on = sensitivity list
```

```vhdl
process (x, y)
begin
  prod <= x and y ;
  wait on x,y;
end process;
```

```vhdl
process
begin
  prod <= x and y ;
  wait on x, y;
end process;
```
VHDL Simulation

Start of simulation

Future values for signal drivers

Assign new values to signals

Evaluate processes

Activate all processes sensitive to signal changes
### VHDL Simulation of an RS FF

Architecture `one` of `RS_Flipflop` is

```vhdl
architecture one of RS_Flipflop is
begin
    process: (R,S,Q,nQ)
    begin
        Q <= R nor nQ;
        nQ <= S nor Q;
    end process;
end one;
```

<table>
<thead>
<tr>
<th></th>
<th>0ns</th>
<th>0ns+δ</th>
<th>0ns+2δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nQ</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

δ cycles reflect the fact that no real gate comes with zero delay.
VHDL Summary

- Entities and architectures
- Multiple-valued logic
- Modeling hardware parallelism by processes
  - Wait statements and sensitivity lists
- VHDL simulation cycle
  - $\delta$ cycles
SystemC: Motivation
SystemC: Methodology
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Ada
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
Levels of hardware modeling

1. System level
2. Algorithmic level
3. Instruction set level
4. Register-transfer level (RTL)
5. Gate-level models
6. Switch-level models
7. Circuit-level models
8. Device-level models
9. Layout models
10. Process and device models
## Instruction level

<table>
<thead>
<tr>
<th>Assembler (MIPS)</th>
<th>Simulated semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>andi $1,$2,100</td>
<td>Reg[1] := Reg[2] ∧ 100</td>
</tr>
<tr>
<td>sll $1,$2,10</td>
<td>Reg[1] := Reg[2] &lt;&lt; 10</td>
</tr>
<tr>
<td>srl $1,$2,10</td>
<td>Reg[1] := Reg[2] &gt;&gt; 10</td>
</tr>
</tbody>
</table>
Register transfer level: MIPS
Gate-level model
Switch level model
Circuit level model

.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
Device level

Measured and simulated currents

$I_D$ [mA]

$2 \quad 4 \quad 6 \quad -V_{DS}$
Layout model
Process model

Simulated

Measured
Models of Computation

- State Machine Models
  - FSM, StateCharts, SDL, CFSM
- Petri nets
- Ada
- Communicating Processes
  - Kahn processes, Communicating Sequential Processes
- Dataflow models
  - DFG, SDFG
- Synchronous languages
  - Cycle based models, Esterel, Lustre
- Discrete Event Systems
  - VHDL, Verilog, SystemC, SpecC
- HW Models
- Unified Modeling Language (UML)
UML (Unified modeling language)
UML - StateCharts
UML – Extended Petri Nets

```
Begin

Develop technology specification

Issue RFP

RFP [issued]

Submit specification draft

Collaborate w competitive submitters [optional]

Evaluate initial submissions

Finalize specification

Evaluate final submissions

Vote to recommend

Specification [final proposal]

Specification [initial proposal]

[if YES] [if NO]

Revise specification

activity
control flow

fork of control

join of control

object flow

input value

join and fork of control

guard

branch

```

"swimlane"
UML Summary

- UML
  - State machine diagram (StateChart-like)
  - Activity diagram (extended Petri nets)
  - Deployment diagram (exec. arch.)
  - Use case diagram
  - Package diagram (hierarchy)
  - Class diagrams
  - Timing diagrams (UML 2.0), UML for real-time
Models and Languages Summary

- Multiple models and languages are essential for high-level design
  - Managing complexity by abstraction
  - Formality ensures refinement correctness
  - Model choice depends on
    - Class of applications
    - Required operations (synthesis, scheduling, ...)

- Multiple MOCs can co-exist during all phases of design
  - Specification
  - Architectural mapping and simulation
  - Synthesis, code generation, scheduling
  - Detailed design and implementation
  - Co-simulation
Sources and References

- Alberto Sangiovanni-Vincentelli @ UCB
- Mani Srivastava @ UCLA
- Rajesh Gupta @ UCSD
- Nikil Dutt @ UCI