Lecture 3: System specifications and models of computation

I. Verbauwhede,
Acknowledgements: H.DeMan
05-06
KULeuven

HJ94 goal: Skiing down a mountain

Specification
Algorithm Transformations
Memory Transformations and Optimizations
Floating-point to Fixed-point

ASIC Special Purpose Retargetable coprocessor DSP processor DSP-RISC RISC

pipelining, unrolling
loop merging, compaction
40 bit accumulator

SPW, Matlab, C++
Overview

Lecture 1: what is a system-on-chip
Lecture 2: terminology for the different steps
Lecture 3 – today: models of computation

Lecture 2: 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...

Polycon

ADT = Abstract Data Type, DT = Discrete Time
System Specs and Models of Computation

Outline:
- What are specifications?
- The tagged signal framework
- Taxonomy of models
- Languages (C++, GEZEL, VHDL, MATLAB…)
- Computational model implementations
  - Extended Control Flow and MATLAB-STATEFLOW
  - Data-Flow and OCAPI (C++)/Gezel, MATLAB-SIMULINK

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 AL):
   Most important ones: Power, Area, Timing, Price,
   Estimators: E(P), E(A), E(T), E(GBW)...

4. A set of constraints:
   \[ E(X) < C \]

   \[
   \begin{array}{c}
   \text{E: Estimator} \\
   \text{AL : abstraction level}
   \end{array}
   \]

Specs and Occam’s principle

- Needed at all abstraction layers (AL)
- Design is top-down refinement of specifications
- Specs are logic, not physics…->semantics
- 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

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...
Typical embedded system model (cont.)

3 main parts:
- Environment: embedded signal reacts to the environment = imposes real-time!
- Signal part
  ADC = sampling + quantization + coding
  Stream of (multi-dimensional) data = sample rate
  Data flow model of computation
- Control part
  React to irregular events
  Basic model = FSM, extended FSM

Design and Modeling issues:
- Concurrent Processes
- Real-Time and Time-tyranny
- Heterogeneous
System specification is heterogeneous

- Many representations of time and values
- Many types of functionality
- Many methods of communication

How to create order in this chaos?

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

What is Time?

Salvador Dalí, *The Persistence of Memory*, 1931

- continuous time
- discrete time
- totally-ordered discrete events
- multirate discrete time
- synchronous/reactive
- partially-ordered discrete events
Many different functionalities

- Environment, Transducers, Signal conditioning
  - Ordinary Differential Equations (time and frequency domain)

- Analog digital interfacing
  - Mix of ODE’s, Continuous time, value and discrete time, value

- Digital signal processing
  - Single- and Multirate Streams, repetitive (time-invariant) algorithms, high throughput -> Data-Flow processes

- User interface, status observation and mode setting, data acquisition and synchronisation, i/o protocols...
  - Complex and usually slowly evolving Control-Flow (Extended Control Flow processes) driven by discrete events. Reactive system

Two schools of thought

One model does all

Dream inefficient

Realistic
But requires understanding
System Specs and Models of Computation

Outline:
- What are specifications?
- The tagged signal framework: classify
  - Taxonomy of models
  - Languages (C++, VHDL, MATLAB, GEZEL, …)
  - Computational model implementations

Tagged Signal Framework

What? Order in the chaos...
- A mathematical framework to understand the essential properties of models of computation
- Compare based on representation of time, signals, state, communication, …
- Here only basics. For more formal treatment see:

Events and Signals as sets

Set of values $V, v \in V$
Set of tags $T, t \in T$
Event $e = (v,t), e \in V \times T$
Signal is a set of events $s = \{e\} \subset V \times T$
Functional Signal is function $s : T \rightarrow V$
Set of all Signals $S = 2^{(V \times T)}$ (powerset)
$N$ - tuples of signals $s = (s_1, s_2, ..., s_N) \in S^N$

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 ($t < t' \Leftrightarrow e < e'$)

- 3 representations:
  - Absolute time
    $T = \mathbb{R}$ (T totally ordered closed connected set)
  - Discrete time
    $T$ is totally ordered discrete set
    $t \neq t' \Leftrightarrow (t < t') \oplus (t' < t) = 1$
  - Precedences
    $T$ is partially ordered discrete set
Processes and Systems

- Process $P$ with $N$ signals is a set of possible behaviours
  \[ P \subseteq S^N \]
- A behaviour $s$ satisfies a process $s = (s_3, s_4, s_5) \in P$
- $s$ can be partitioned into inputs $S_i$ and outputs $S_o$
  \[ s = (S_i, S_o) = (s_3, (s_4, s_5)) \]
- A process $P$ is functional or deterministic if it is a single valued mapping:
  \[ P : S_i \rightarrow S_o \text{ or } (s_4, s_5) = F(s_3) \]
- A system $Q$ is a composition of processes
  $Q(s_1, s_5) = (s_4, P = s_6, P_1)(s_2, P_1 = s_3, P)$

?? Is $Q$ Deterministic if $P$ and $P_1$ are????

Causality definitions

Process is:

- Causal: Delay $\Delta$ is nonnegative
  \[ \Delta \geq 0 \]
- Strictly causal: delay is positive (can be arbitrarily small)
  \[ \Delta > 0 \]
- Delta causal: at least larger than a given positive $\delta$
  \[ \Delta \geq \delta > 0 \]

VHDL, MATLAB processes are strictly causal

* Distance metric on tags
System Specs and Models of Computation

Outline:
- What are specifications?
- The tagged signal framework
- Taxonomy of models of computation
- Languages (C++, VHDL, MATLAB…)
- Computational models

Models and processes

System Behaviour

Functional Behaviour

Processes

Communication Behaviour

Values around a tag

Delay

State-less processes

State processes

Tag synchronisation between events in pairs of processes

Communication processes
Functional Behaviour

- **Timed Models of Computation** = total order
  - Continuous time-
  - Discrete event-
  - Synchronous-
  - Discrete time-(Synchronous/Reactive)
  - ...

- **Untimed Models of Computation** = partial order
  - Sequential Processes with Rendez-Vous
  - Kahn Networks
  - Data-Flow networks
  - ...

Timed Models of Computation

A *timed model of computation* has a tag system where $T$ is a *totally ordered set*.

That is, for any distinct $t$ and $t'$ in $T$, either $t' < t$ or $t < t'$
Continuous time systems (1)

Let \( T(s) \) be the set of tags of signal \( s \) in a timed system (\( T(s) \) has total order)

- A continuous time (CT) system is a timed system where \( T \) is a continuum* and \( T(s) = T \) for each \( s \) in the tuple \( s \) that satisfies the system.

The model for a CT system is a set of ODE’s

* closed connected set \( S \): no two disjoint sets \( O_1 \) and \( O_2 \) s.t.

\[
O_1 \cup O_2 = S. \text{ Example: } \mathbb{R}.
\]

Continuous Time Systems (2)

ODE’s cannot be solved exactly on a computer which in essence operates in discrete time.

Simulation requires time discretization and numerical integration.
(\text{SPICE, MATLAB SIMULINK, VHDL-AMS})

Three problems:

- **Accuracy**: truncation error = function time step \( h \)
- **Stability**: stability region different from ODE itself
- **Convergence** if non-linear
Discrete Event System

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

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

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 $e_i$; Sorting!!!
     Remove $e_i$ from slot $t$;
     Project $e_o$ on $Q$ at $t+\Delta_{oi}$ slot ($\Delta_{oi}$: delay of $e_o$ w.r.t $e_i$);}
  }
}

Simulation mechanism of VHDL, MATLAB-STATEFLOW

$e_o$'s are $e_i$'s of fan-out processes
Queue = Linked List

Discrete Event Simulation

- Useful for systems with high degree of inactivity. Only active parts are computed (digital systems at gate level, concurrent processes with delta causal delay). Basis of VHDL simulator.

- But sorting events in Q is very costly in CPU time

- Ok for delta causal delay loops. Problems with zero delay loops. Can be non-deterministic. Cft VHDL “delta” delay to handle simultaneity.
Be careful with DE simulation

- If system contains loops with not delta causal processes.
- VHDL allows “zero delay” simulations but simulates them as strictly causal with $\delta=0^+$ $\delta$ is not the delta of delta causal!

Discrete fixed point problem with no solution!
Result is no longer a discrete event system!

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 \( \perp \).

Input signals are synchronous to clock signal.

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

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) Example = GEZEL.
- 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)
Extension: Synchronous/Reactive Systems

Idem as clocked discrete-time systems but any input event is considered to be a clock tick. Loop delay by master-slave memory. Basic simulation mechanism of MATLAB-STATEFLOW, SDL, ESTEREL extended state machine simulation (see later). Time between two consecutive events = called cycle. Formally provable properties (see http://www-sop.inria.fr/meije/esterel/esterel-eng.html)

Very powerful model. But need also “asynchronous” communication. “locally synchronous, globally asynchronous” needed for implementation. Also synchronicity to check.

Untimed Models of Computation

- Model $Q$ in which $T(s)$ with $s \in Q$ is a poset. 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
Sequential processes with Rendez-Vous

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

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

Sequential processes with rendez-vous

Concurrent Sequential Processes:
- CSP: C.A.R Hoare 1978

Calculus of Communicating Systems
- CCS Milner 1980

Languages: Occam, Lotos, Ada

Implementations:
- Interleaving on single processor
- Multiprocessor with rendez-vous handshake (Transputer)
- vhdl: communication=discrete event shared memory and code interleaving (cut = wait on or wait until)
VHDL rendez-vous

channel chan: integer;
P1:
  ...
  send(chan,msg);
  ...
P2:
  ...
  receive(chan,m);
  ...

abstract

signal chan: integer;
signal chan_req,chan_ack: boolean;
P1:
  ...
  chan<=msg;
  chan_req<=true;
  wait until chan_ack;
  chan_req<=false;
  ...
P2:
  wait until chan_req;
  m<=chan;
  chan_ack<=true;
  wait until not chan_req;
  chan_ack<=false;
  ...

Impl.

Other Implementation: process network (Kahn\(^1\))

Disadvantage CSP: blocking send stops sender;
slows down computing

Kahn network: channel is unbounded FIFO (totally ordered stream)
process is continuous, i.e. adding more tokens to the input stream
may not result in a change or removal of output tokens already
produced. Way to do it: use blocking read: reading process
suspends if no input tokens available. Provably deterministic
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.**
- More details in the next lectures.
Communicating Processes

- Model communication. Impose order on tags of communicating processes.
- Two actions;
  - Transfer of values on shared time stamps (communication)
  - Impose (partial) ordering relationship between T's of processes (synchronisation)
- Taxonomy
  - Unsynchronised
  - Read-modify-write
  - Unbounded FIFO
  - Bounded FIFO
  - Rendez-Vous

Languages

- Languages are syntactic sugar to describe MoC’s
- Not language, but MoC is important
- People like standard languages (VHDL, C++, Java, MATLAB, SDL…)
- Languages \(\supseteq\) Models. Model = style of programming
**Language Spectra**

<table>
<thead>
<tr>
<th>Requirem.</th>
<th>System</th>
<th>Algor.</th>
<th>RT</th>
<th>Logic</th>
<th>Circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MATLAB/SIMULINK/STATE-FLOW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSTEM-C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HJ91</td>
<td>GEZEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VHDL, Verilog</td>
<td></td>
<td></td>
<td>-AMS, SPICE</td>
</tr>
</tbody>
</table>

**MATLAB...**

MATLAB: array programming, file I/O, plots

*Static Workspace*

SIMULINK:
Structural programming: blocks=behaviour & testbench

- Continuous time: ODE, Num Int. (smallest time step)
- Discrete time: Multirate streams: diff. Eq. DF
- Events: switches, thresholds, steps
- Float and bit-true

STATEFLOW:
- Synchronous/reactive extended state machines
- Concurrency
- Preemption

Graphical Programming -> C
Conclusion

Models of computation

• Associated with levels of abstraction
• Allow reasoning without details
• Need to know the boundaries (where applicable, where not)

We will use a lot:

• Data flow representation
• Control flow