



Hardware Description Languages & System Description Languages – Properties

There is a need for executable *specification language* that is capable of capturing the functionality of the system in a machine-readable and simulatable form

- **Simulation**
- **Input to synthesis tools**
- **Serve as comprehensive documentation**
- **Medium for the exchange of design information**

The goal of any language is to capture the conceptual view of the system with the minimum of effort on the part of the designer



Properties of Conceptual Models

- **Concurrency: data-driven, control-driven**
- **State transitions**
- **Hierarchy: structural, behavior**
- **Programming constructs**
- **Behavioral completion**
- **Communication: shared memory, message passing**
- **Synchronization: control-dependent, data-dependent**
- **Exception handling**
- **Non-determinism**
- **Timing: functional, timing constraints**

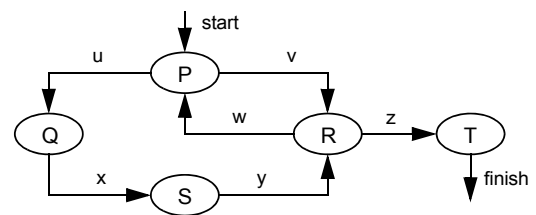


Concurrency

- In many cases, the functionality of a system is most easily conceptualized as a set of concurrent behaviors
- The result of concurrent behaviors F_1 and F_2 in sequential representation is cross product: $F_1 \times F_2$
- Data-driven concurrency => data dependencies
- Control-driven concurrency => threads
 - job level / task level (fork-join) / statement level / operation level / Bit level

State Transitions

- A system can be conceptualized as having various *modes* or states, of behavior
- A machine with N states can have N^2 possible transitions



Hierarchy

- Large systems can be too complex to be considered in their entirety
- Hierarchical models allow a system to be conceptualized as a set of smaller subsystems (*divide and conquer*)
- Hierarchical model provides a mechanism for objects scope
- Two types of hierarchy
 - Structural hierarchy - set of interconnected components
 - Behavioral hierarchy - behavior decomposed into sub-behaviors

Programming constructs

- The behaviors can be described as sequential algorithms, described in the best with *programming language constructs*
 - Data types
 - Branching - `if`, `case`
 - Iteration - `while`, `for`, `repeat`
 - Subroutines - `function`, `procedure`



Behavioral Completion

- That is, behavior's ability to indicate that it is completed, as well to the ability of other behaviors to detect this completion
 - FSM model – *final state*
 - Program-state machine model – *completion point*
- Advantages of specification of behavioral completion:
 - Helps designers to conceptualize each hierarchical level, to view it as an independent module
 - Allows the natural decomposition of a behavior into sub-behaviors which are then sequenced by the “completion” transitions

Communication

- Shared memory communication model, includes *broadcast* mechanism
 - *Persistent* medium – memory, value retains until overwritten
 - *Non-persistent* medium – buses
- Message passing communication model – abstract *channel* over which *messages* are sent; data is transferred by using *send* and *receive* primitives
 - uni-directional; bi-directional
 - point-to-point; multiway
 - blocking; non-blocking



Synchronization

- Processes are not completely independent of each other and have to be synchronized for certain actions (data exchange)
- **Control dependent synchronization**
Using fork-join statements
- **Data dependent synchronization**
 - Shared memory based synchronization
 - Synchronization by common event
 - Synchronization by common variable
 - Synchronization by status detection
 - Synchronization by message passing (blocking communication)



Exception Handling

- An event can require that a behavior or mode be *terminated immediately*
- Example: computer interrupts
- FSM model - transition to certain state from each other state

Non-determinism

Selection

```
if(x) then
  do EITHER a OR b
end if
```

Ordering

```
if(x)
  do BOTH a AND b
end if
```



Timing

- Timing is a way to reflect the real world implementation
- **Functional timing** – timing information which affect the simulation output of the system specification
wait statement in VHDL
- **Timing constraints** – intended for use by synthesis and verification tools
 - Performance
 - Data rate
 - Min/max timing
 - etc.



Language Support for Conceptual Model Characteristics of Embedded Systems

Language	State transitions	Behavioral hierarchy	Concurrence	Program constructs	Exceptions	Behavioral completion
VHDL	-	~	+	+	-	+
Verilog	-	+	+	+	+	+
Hardware-C	-	~	+	+	-	+
CSP	-	+	+	+	-	+
StateChart	+	+	+	-	+	-
SDL	+	~	+	-	-	+
Silage	n/a	n/a	+	n/a	n/a	n/a
Esterel	-	+	+	+	+	+
SpecCharts	+	+	+	+	+	+
SystemC	-	+	+	+	+	+

CSP - Communicating Sequential Processes

SDL - Specification and Description Language

[based on Gajsky et. al. 1994]



VHDL

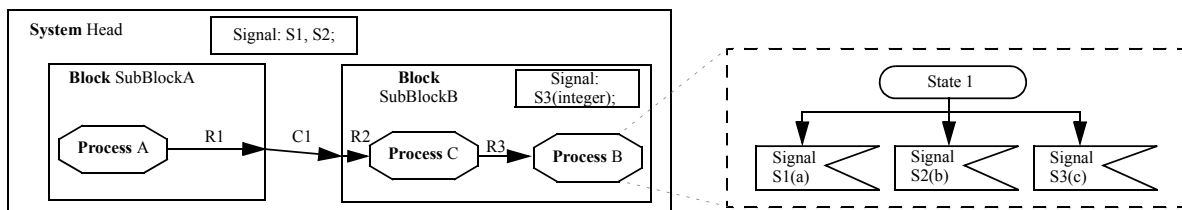
- **VHDL supports:**
 - Structural & behavioral hierarchy
 - Statement level concurrency
 - Full support of programming constructs
 - Communication between processes adopting shared memory model (common/global signals)
 - Synchronization by means of process's *sensitivity list* or *wait* statement
- **VHDL does not support exceptions and state transitions**

Verilog

- **Verilog supports:**
 - Structural & behavioral hierarchy
 - C-like programming constructs
 - Communication using shared-memory model (wires connecting ports on modules)
 - Synchronization (fork-join, event control)
 - Exceptions (*disable* statement in named blocks)
- **Verilog does not support state transitions**

SDL - Specification and Description Language

- SDL is standardized by *ITU* (International Telecommunication Union).
- SDL is an object-oriented language with concepts for describing the logical structure, data and behavioral aspects of systems.
- SDL is widely used in the telecommunications field (reactive and discrete systems).
- The basis for description of behavior is communicating *Extended State Machines* that are represented by processes.
- Communication is represented by signals and can take place between processes or between processes and the environment of the system model.
- Some aspects of communication between processes are closely related to the description of system structure.



- <http://www.sdl-forum.org>

CSP - Communicating Sequential Processes

- Proposed by C.A.R. Hoare in 1978
- The basic idea of CSP is that multiple concurrent (parallel) processes can synchronize with each other most easily by synchronizing their I/O
- The purposed way to do this is to allow I/O to occur only when:
 - process A states specifically that it is ready to output to process B, and process B states specifically that it is ready for input from process A
 - if one of these happens without the other being true, the process is put on a wait queue until the other process is ready
- **CSP Advantages**
 - conceptually simple
 - flexible
- **CSP Disadvantages (Problems)**
 - must know the *specific* name of any process you communicate with
 - no I/O buffering (programmer must add)



Occam

- William Occam (c. 1280-1349)
- It's a parallel computing language based on the CSP programming model designed by Tony Hoare
- The main implementations of the language are currently on INMOS Transputers which were designed with the CSP programming model
- Two versions of the language, occam and occam2, have been produced so far with occam3 in the pipeline
- A portable and free occam compiler, called SPOC, has been released by the University of Southampton

Esterel

- *Esterel* is an imperative synchronous parallel programming language dedicated to reactive systems
 - Control-handling dominant
 - Input-driven
 - Zero-delay reaction
- <http://www.esterel.org/>



Erlang

- *Erlang* is a functional programming language with processes that is suitable for implementing large systems with soft real time demands
- The main advantages of Erlang are robustness, speed of development and reduced maintenance
- Most of the concepts in Erlang have been taken from the modern programming languages that have proven to be good
- The languages Prolog, Strand, Parlog and Eri-Pascal have had an especially high influence
- Main application area – telecommunication
- <http://www.erlang.org/>

- A simple erlang program for finding factorial

```
-module(math1).
-export([factorial/1]).
```

```
factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).
```

```
>math1:factorial(20).
2432902008176640000
>
```



Java

- **Java is an object-oriented programming language developed by Sun**
 - Strong typing
 - No unsafe constructs
 - The language is small so its easy to become fluent
 - The language is easy to read and write
 - There are no undefined or architecture dependent constructs
 - Java is object oriented so reuse is easy
 - Java has concurrency

C / C++ based HDLs

- **Extending language itself and/or libraries**
 - HardwareC / HandelC / uC++ / SystemC
- **C/C++ based design flows**
 - Ocapì / Gezel



Hardware-C

- **Rajesh Gupta, David Ku, ~1990**
- **Extended C - features to describe hardware**
 - module types - process, function
 - i/o direction - module parameters
 - synchronization - wait, read, write, free
 - boolean type, bitwidth declaration, access to single bits

```
process i8251(ChipSelect, WriteEnable, /* ... */ control, status)
    in boolean ChipSelect;
    in boolean WriteEnable;
    /* ... */
    out boolean control[DataSize];
    in boolean status[DataSize];
{
    /* ... */
    switch (decode) {
        /* ... */
        case 0xD:
            dbuf = read ( data );
            control = dbuf;
            write control;
            break;
        /* ... */
    }
}
```




Handel-C

- Embedded Solutions, Ltd. – Spin-off company of Oxford University
- Now Celoxica, Ltd. – <http://www.celoxica.com>
- System-on-Chip design / Reconfigurable computing
- An assignment statement takes exactly one clock cycle to execute

```

void main(void)
{
    unsigned 8 x, y;
    unsigned 5 temp1;
    unsigned 4 temp2;
    ...
    par
    {
        temp1 = (0@(x <- 4)) + (0@(y <- 4));
        temp2 = (x \\ 4) + (y \\ 4);
    }
    x = (temp2 + (0@temp1[4])) @ temp1[3:0];
}

z = x <- 2; // Take LSBs
z = y \\ 2; // Drop LSBs
z = x @ y; // Concatenation
z = x[3]; // Bit selection
z = y[2:3]; // Bus selection
z = width(x); // Width of expression

// internal
rom unsigned int 8 program[] = {1,2,3,4};
// external
ram unsigned int 4 ExtRAM[8]
with { offchip = 1,
data = {"P01", "P02", "P03", "P04"},
addr = {"P05", "P06", "P07"},
we={"P08"}, oe={"P09"}, cs={"P10"} };

```



OCAPI

- C++ based design environment
 - abstraction of a complex problem
 - independence of implementation details
 - huge potential for IP reuse
 - gradual refinement to implementation possible
- OCAPI library
 - library primitives
 - simulation / VHDL/verilog code generation

GEZEL

- Successor of OCAPI
- Cycle-based hardware description language
- HW/SW cosimulation / VHDL code generation
- User-defined library-block extensions in C++
 - new cosimulation/cosynthesis interfaces
- <http://sourceforge.net/projects/gezel/>



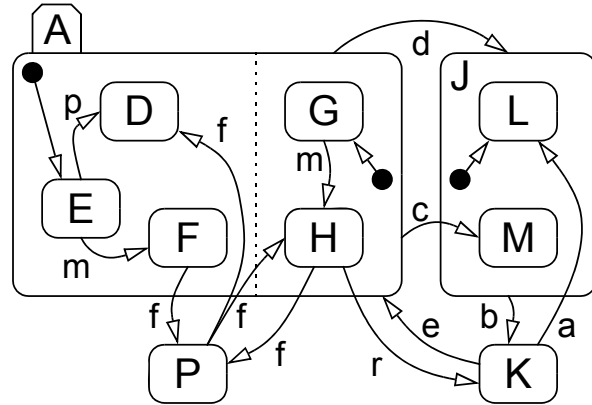
StateChart

David Harel, Israel, 1989.

Statechart - visual formalization for the behavioral description of complex system (extended FSM paradigm).

Properties:

- Hierarchy;
- State history;
- AND (concurrency) and OR clustering;
- *timeout* transitions.

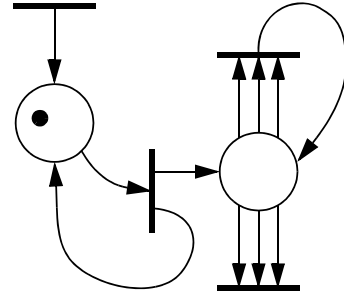


Formal Description Techniques

- Purposes:
 - unambiguous, clear, and concise specification
 - completeness of specifications
 - consistency of specifications, in isolation and relatively to each other
 - tractability of specifications
 - conformance of implementations to specifications
- ESTELLE – Extended Finite State Machine Language (ISO 1989e)
 - specification defines a system of hierarchically structured state machine
- LOTOS – Language of Temporal Ordering Specifications (ISO 1989)
 - can be used to specify all allowed behaviors of a system
- SDL – Specification and Description Language
 - provides constructs for representing structures, behaviors, interfaces, and communication links

Petri-Net

- A directed graph consisting of two types of nodes - *places & transitions*
- A place may or may not have a *token*
- A transition is *fired* when all places preceding the transition have at least one token that are removed
- Firing generates new tokens for every succeeding place of the transition
- Firing order of transition is *non-deterministic* if more than one transition can be fired
- The state of the system is represented by the distribution of tokens in the system
- The dynamic behavior of a system is described by flow of tokens in the system



Netlists

- Set of interconnected modules
- Netlist of gates - structural VHDL, Verilog, EDIF, etc.
 - cycle based
 - event based
- Netlist of devices - Spice (and alike), VHDL-AMS, etc.
 - frequency domain - Fourier
 - time domain - Laplace



What next – UML, SystemC or SystemVerilog?

- **UML - Universal Modeling Language**
- **UML is standardized, graphical notation used in the object-oriented analysis and the object-oriented design of systems. It is used for specifying, visualizing, and documenting the artifacts of an object-oriented system under development.**
- **UML defines a number of graphical diagrams that provide different perspectives of the system under development - characteristics, relations, processes, etc. *State charts* form another representation that aims to represent a behavioral view of a system.**
- **UML is standardized by Object Management Group in 1997.**
- **But it is for software development!**
- `http://www.uml.org/`



What next – UML, SystemC or SystemVerilog?

- **SystemC – language and modeling platform, based on C++**
- **SystemC 2.0 – enabling system-level modeling**
 - both software and hardware implementations
- `http://www.systemc.org`
- **SystemVerilog**
- **SystemVerilog is a Hardware design and Verification language having features inherited from Verilog and C++. SystemVerilog is a solution to decrease the gap between design and verification language. Already leading EDA companies like Synopsys, cadence, MentorGraphics have adapted SystemVerilog in their tools.**
 - `http://electrosofts.com/systemverilog/introduction.html`
- **Overview: SystemVerilog for RTL Designers**
 - `http://www.esperan.com/pdf/esperan_systemverilog_rtl.pdf`