SMUG04
Shlaer-Mellor User Group

Experiences from the first step in designing an Architecture executing Executable UML semantics in Programmable Logic using VHDL

Anders Eriksson
Senior Software Systems Engineer
Saab Bofors Dynamics, Linköping, Sweden

Contents

- Background
- FMV Funded Study
- VHDL
- Design – The First Step
- Experiences

Background

- Experiences from the Software Development with xUML presented in earlier presentation
- Software/Hardware co-design – good and bad experiences
- Electronic and Software Engineers are working (reasonably) close together in projects
- Engineers should have an interest in both disciplines to a certain level ⇒System-wide perspective
- Experiences from System IV
  - Communication handling – timing requirements, …
  - Digital and analog decoding – complex algorithms, timing, …

  ⇒ complex software design and processor loading
- When the above functionality is moved, the processor is free to concentrate on the calculation of the algorithms
- Difficult when no direct translation rules from xUML model to a VHDL implementation exists

FMV Funded Study

- Technical studies can be funded by FMV to stimulate usage of new techniques in products from the defence industries in Sweden
- The department Centre of Expertise Sensors & Telecom within FMV where interested in finding a way of expressing the system without knowing the realisation. They were looking for Software Hardware Co-Design.
- During a visit at FMV, a presentation was held regarding our work with the Ada95 model compiler and how we can specify the system regarding software without thinking of the realisation
- A proposal for a study regarding “Open code translation from Executable UML models to VHDL” was then put together

FMV = Swedish Defence Materiel Administration

FMV Funded Study

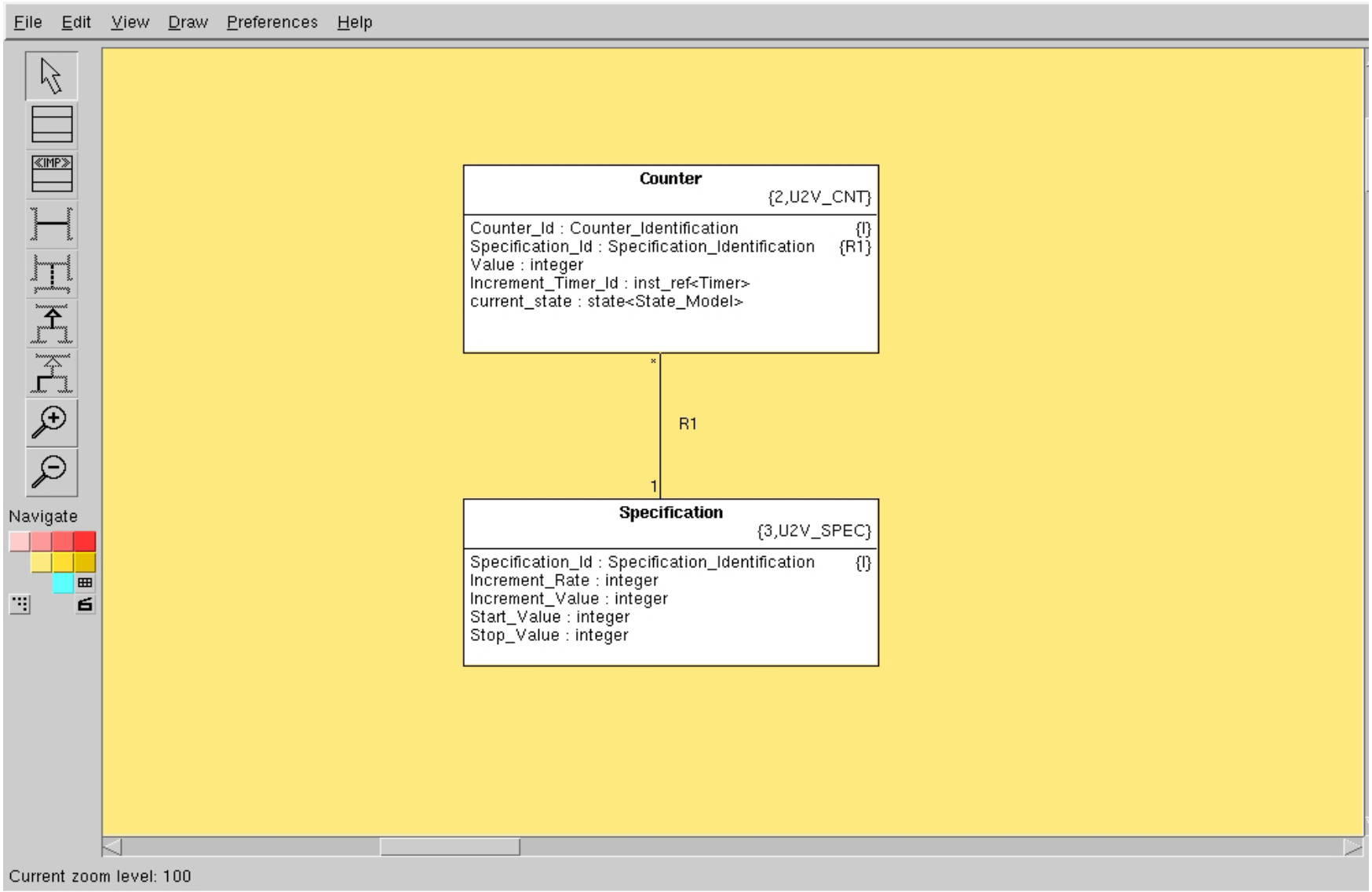
- The primary goal was to investigate how to use the VHDL as a language to build an architecture that can take the xUML model to *construction* and not only to the *simulation* level

- First step
  - Design the mechanisms necessary to fulfil the xUML model semantics
  - Identify implementation patterns used when hand coding an xUML example model

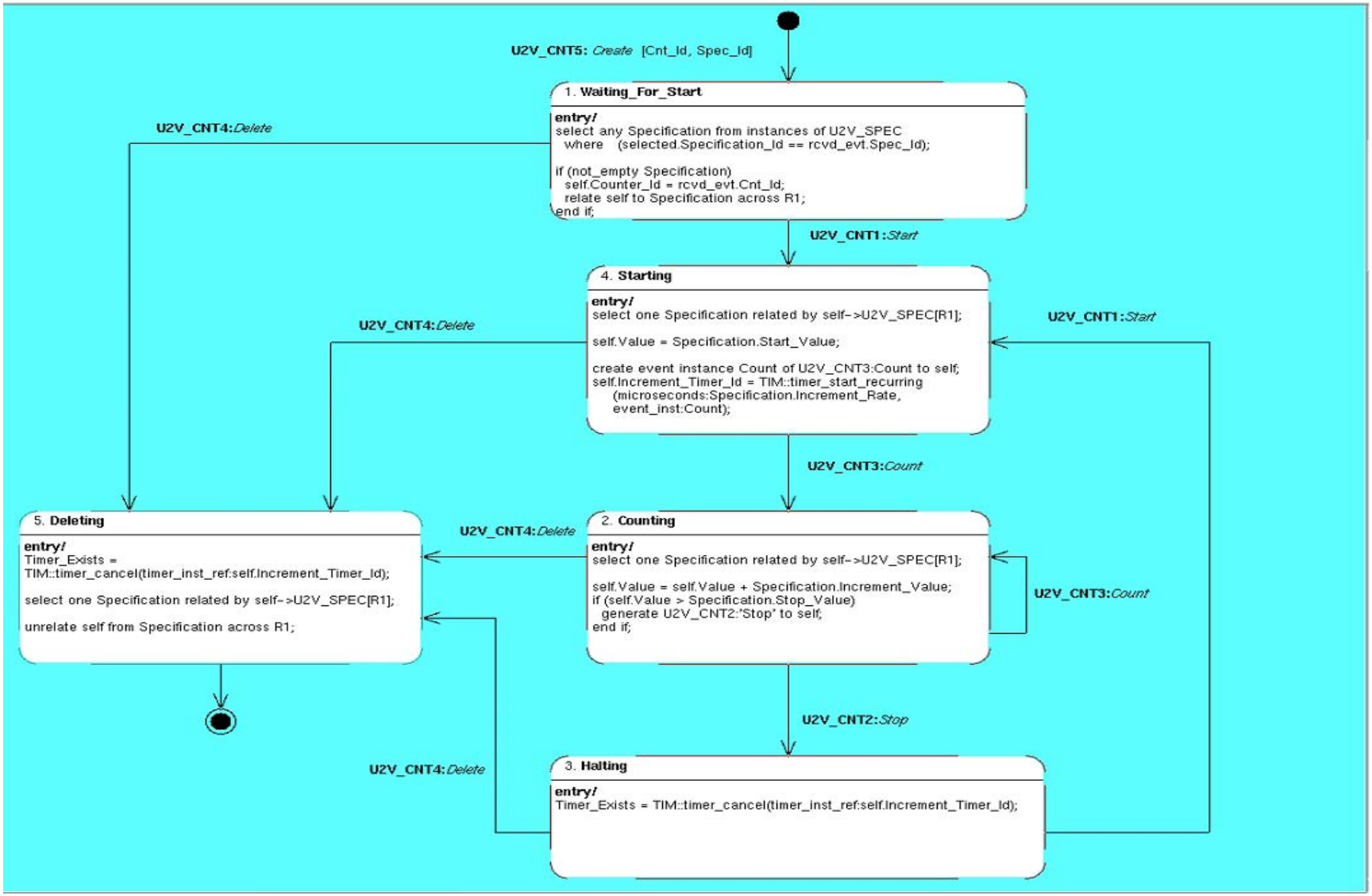
- Investigate which constructs that are *synthesisable* in VHDL concerning the
  - VHDL Language
  - xUML model semantic

- How to partition the architecture
  - Access
  - Execution
  - Libraries
  - …

Example Model



Example Model



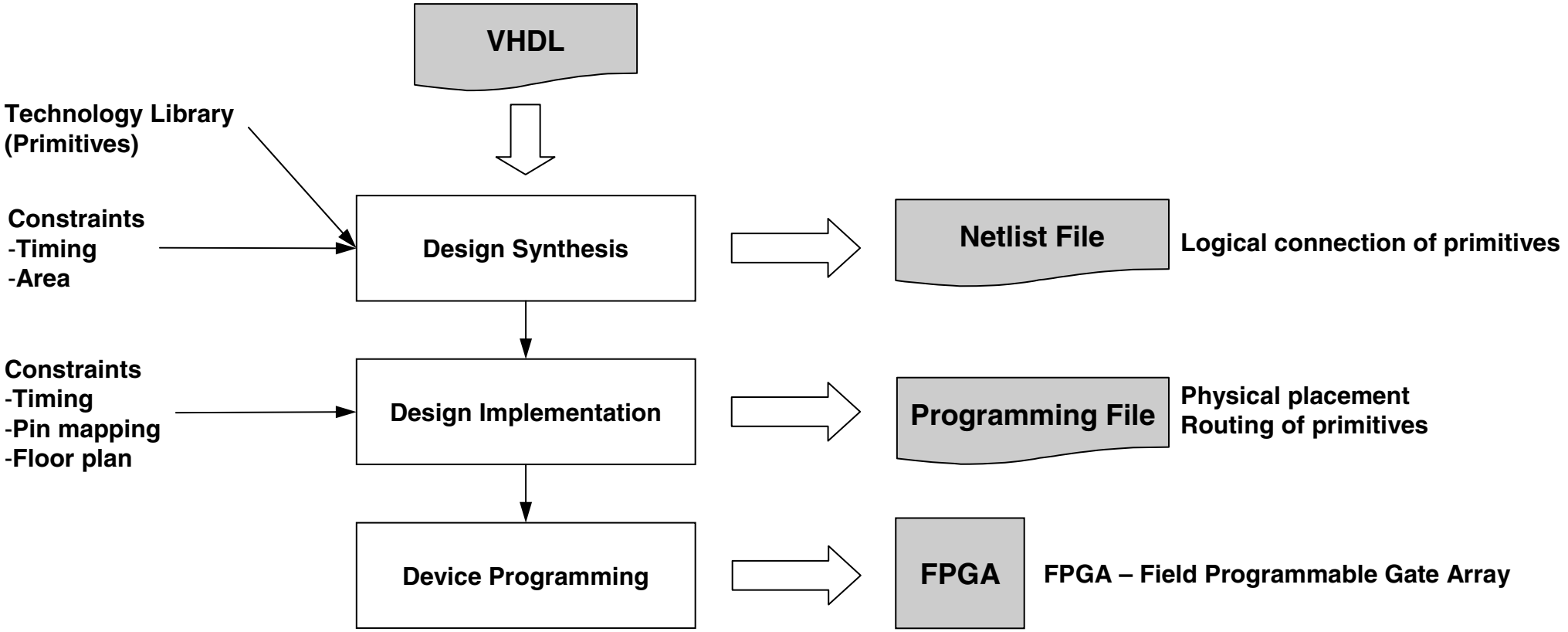
RTPLA-AE Anders Eriksson

2004-11-09

VHDL – History

- VHDL can be used to describe a digital system at many levels of abstraction ranging from the algorithmic level to the gate level
  - Can be seen as a super-set of Ada and with an extension called *signals*
- VHDL is an acronym for VHSIC Hardware Description Language. VHSIC is an acronym for Very High-Speed Integrated Circuits
- The main focus was to be able to write *executable* specifications for simulation and not for construction
- The construction was made by manually converting the models into schematics using gates and building blocks from a target library
- In the beginning of the 1990's, VHDL *synthesis* tools that could convert VHDL models directly to a technology netlist started to emerge
- Synthesis can be compared to a compiler that generates machine code out of Ada95 code

VHDL – FPGA Implementation Flow



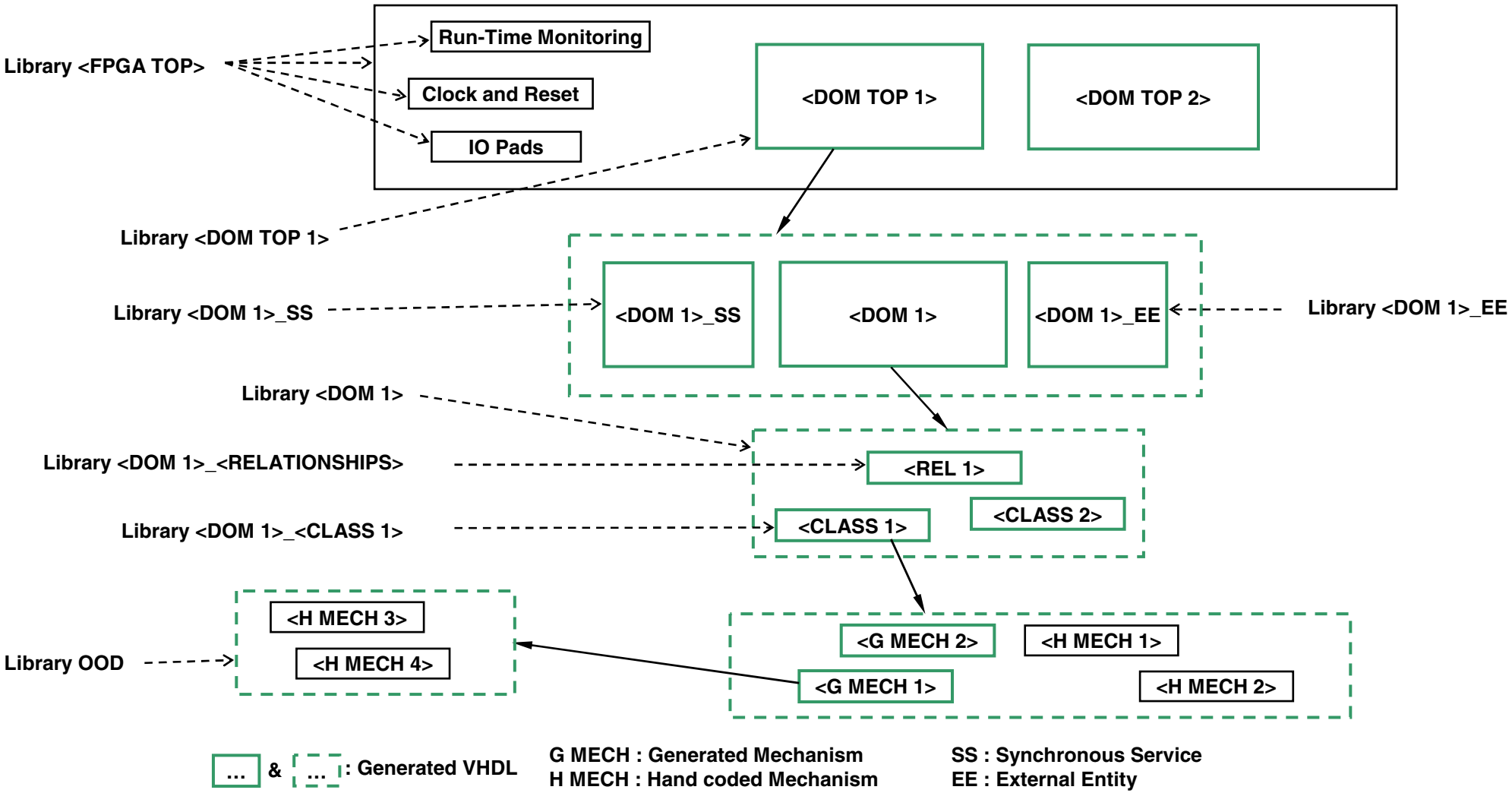
VHDL – The Language

- Design units in VHDL
  - Entity
  - Architecture
  - Package Header
  - Package Body
  - Configuration (Simulation)
- Design partitioning with VHDL
  - Block
  - Package, Header + Body (optional)
  - Component, Entity + Architecture
  - Library
  - Configuration (Simulation)
- Design modularity with VHDL
  - Procedure
  - Function
- Data types in VHDL
  - Scalar (Enumeration, Integer, Real, …)
  - Composite (Record, Array)
  - …

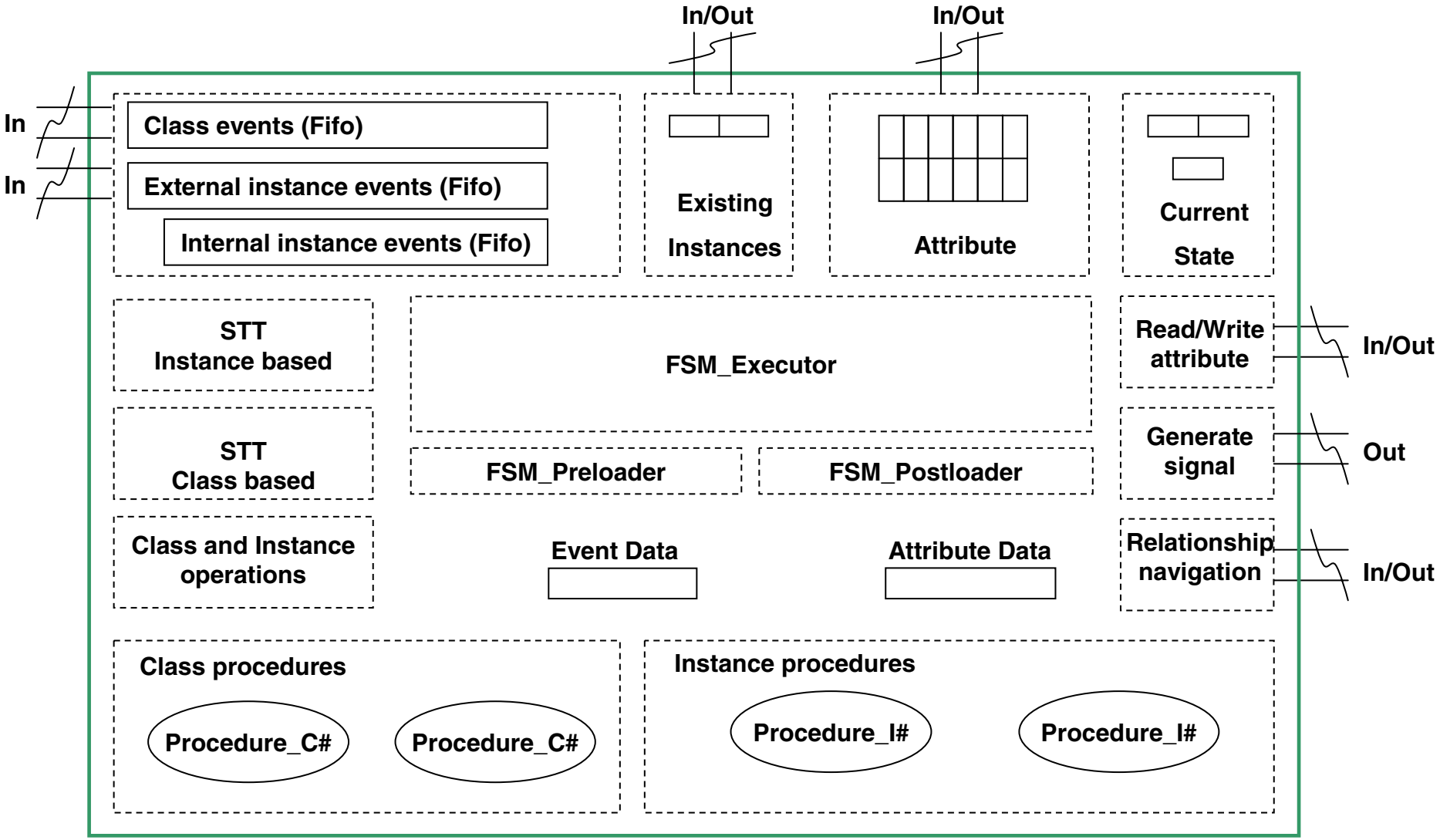
Design – The First Step

- Architecture structure, partitioning with VHDL
- Execution
- Communication between mechanisms
- Run-time monitoring
- Implementation issues

Design – Architecture Structure



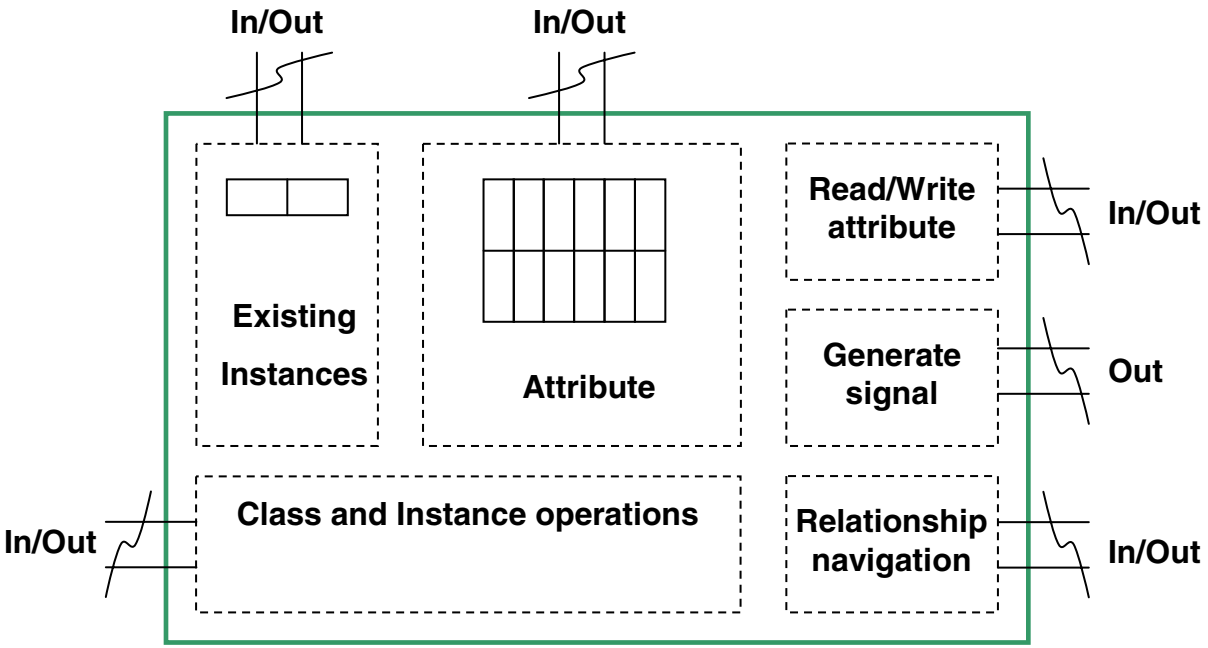
Active Class



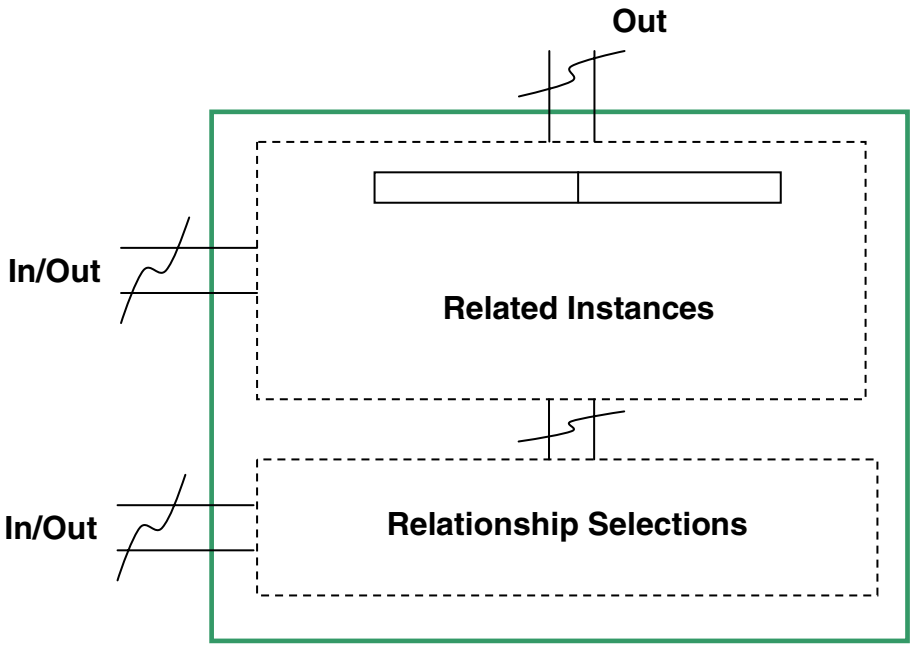
RTPLA-AE Anders Eriksson

2004-11-09

Passive Class



Relationships

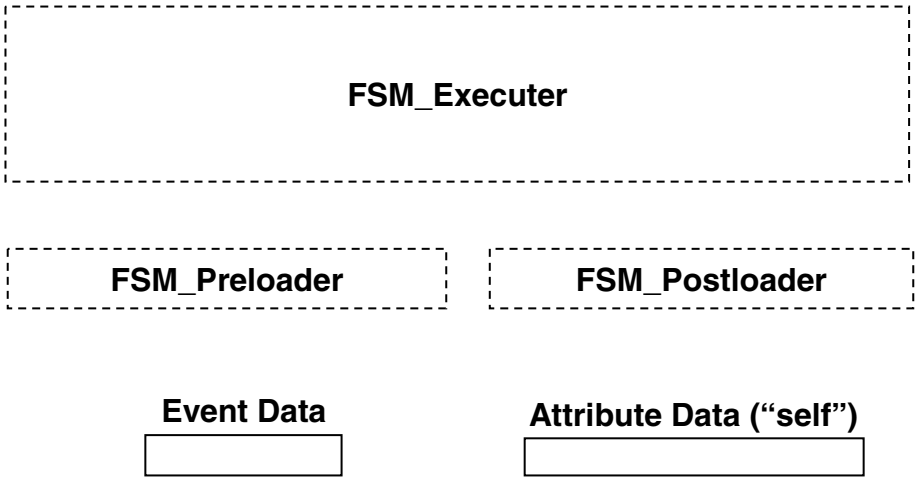


Design – Execution

- One Instance per Class is executing at a time
- Instances from different classes are executing concurrently
- All instances within a class share the same state Procedures
- The following steps is performed during execution

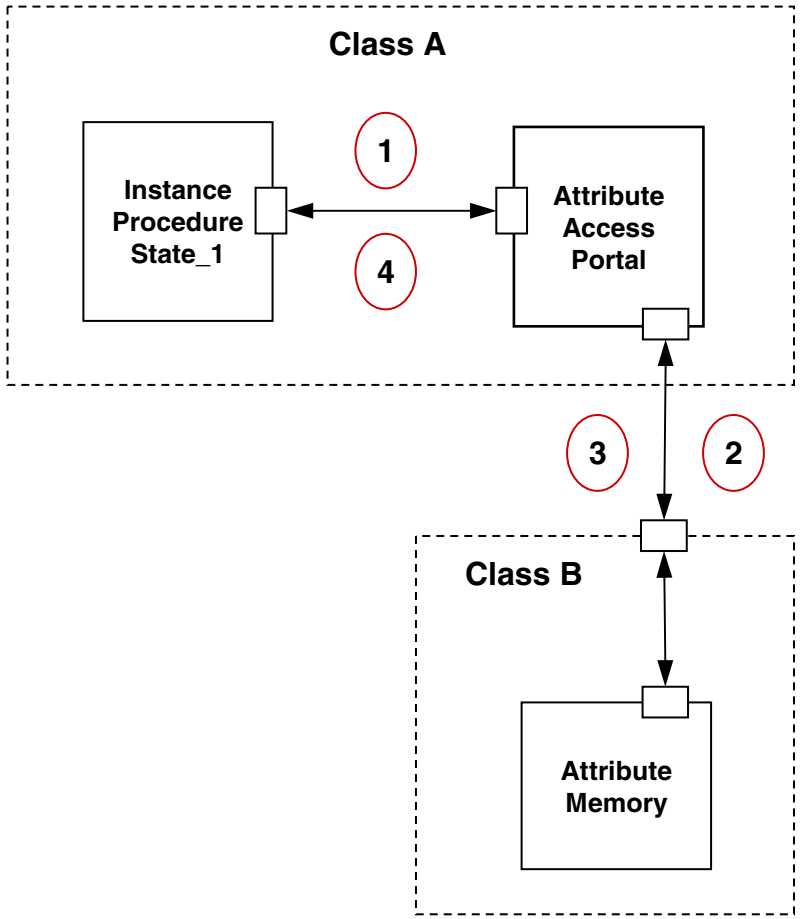
1. Check Internal, External and Class Event Queue
2. If creation event, create instance
3. If instance based, check if instance exists

3.1 Get current state and evaluate STT
3.2 If transition
3.2.1 Update current state with the new state
3.2.2 FSM_Preloader
3.2.3 Run the procedure for the new state
3.2.4 FSM_Postloader



Design – Communication Between Mechanisms

- Every mechanism will follow the handshaking rules as follows
  - Start_<Command>, Done_<Command>
  - Request, Grant
    - when more than one resource-user is attached
- Inside the procedures and operations the data types from the xUML model are used
- Every external communication goes through a “Portal” where the typed data will be converted to a general address and data bus
- Every Class has several subprograms to support the address and data conversions



Design – Run-Time Monitoring

- All mechanisms and implementation patterns are prepared for run-time monitoring

- Different kind of monitoring categories
  - Protocol Error – hand shaking between mechanisms, …
  - Sizing – event queue, instance table, ...
  - Inconsistent model – navigating an association that is unconditional and no instance is related, …
  - Logging – class state transitions, …

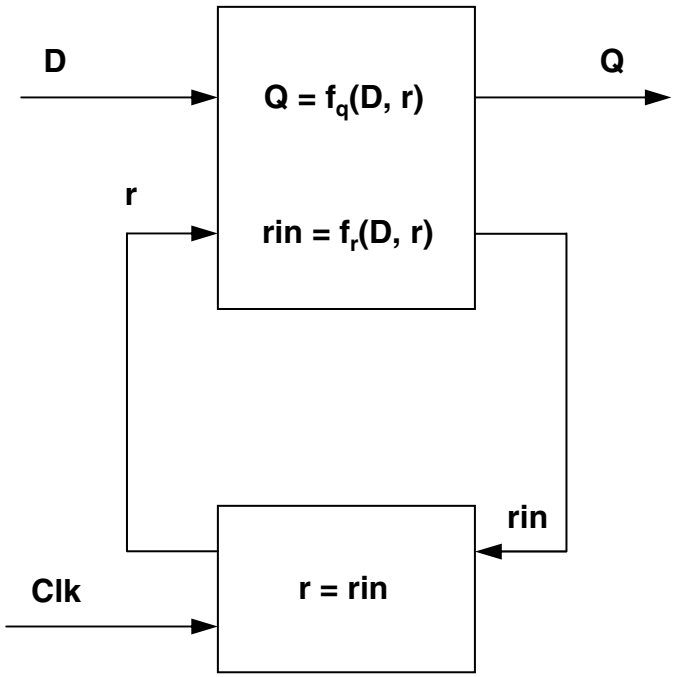
- The run-time monitoring is configurable

Design – Implementation Issues

- Data type Real is not supported
- Inspired by “A Structured VHDL Design Method” written by Jiri Gaisler
  - http://www.gaisler.com/doc/vhdl2proc.pdf
  - Named ‘two-process’ method
- Prescribes how to use the VHDL language to construct/code on a higher level than the traditional ‘dataflow’ style
- The method is used in LEON3
  - A synthesisable VHDL model of a 32-bit processor compliant with the SPARC* V8 architecture
- Two Process Schema
  - Combinational (asynchronous) logic
  - Sequential logic (register)
  - State Machines
    - Moore, Mealy, Lean (mix of Mealy and Moore)
  - Increases abstraction level
  - Improves readability
  - Improves simulation speed

** SPARC is a registered trademark of SPARC International*

Combinational Process, the algorithm

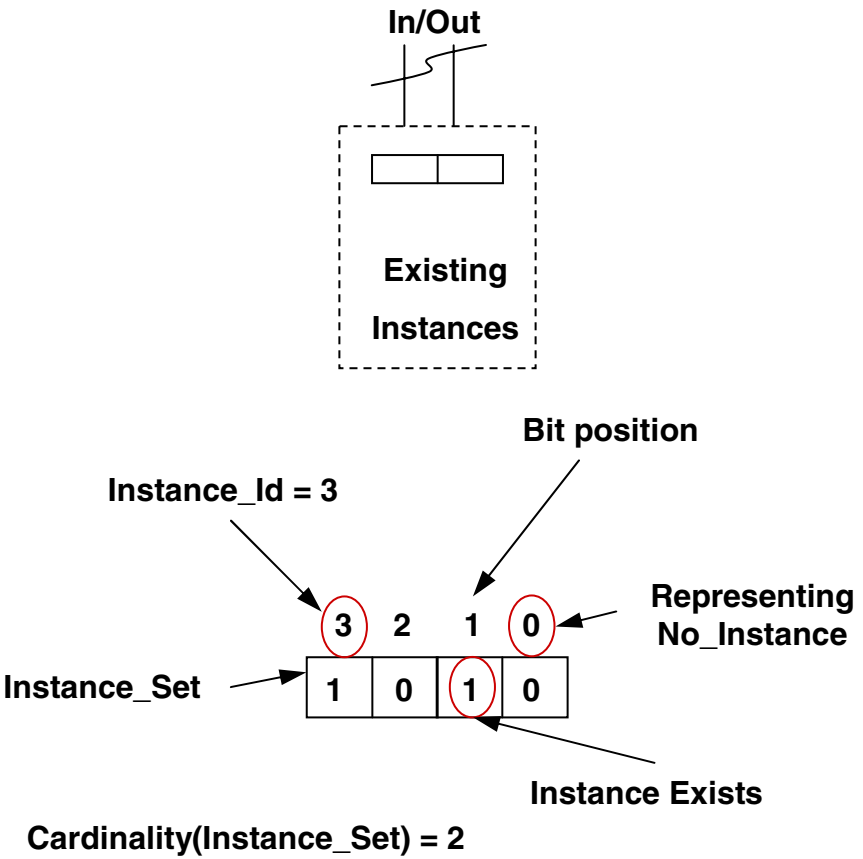


Sequential Process, the state

The figure origins from the document http://www.gaisler.com/doc/vhdl2proc.pdf

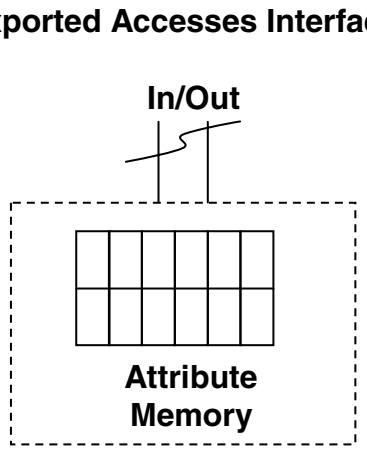
Design – Class Manipulation

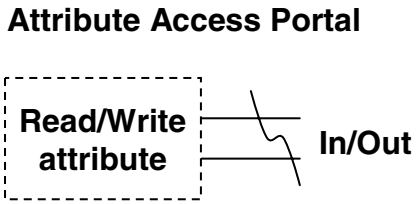
- Creation and Deletion of Instances can be done
  - Synchronously
  - Asynchronously
- Only a fixed number of instances are available
- The information about existing instances is always available as read-only data for other mechanisms e.g. Relationships, FSM, …
- Data type Instance_Set is coded as “One-Hot”, i.e. bit position corresponds to Instance_Id
- Selecting Instances
  - Select Many - already available
  - Select Any - first bit position set to one
  - Where clause - executed by the receiver



Design – Class Manipulation

- The Attribute memory is pre-created and the size is dependent on
  - Total number of instances
  - Bit-width of the data type of each attribute
- Reading and Writing attributes can be done simultaneously if not to same Address
- The Address is composed of Instance_Id and Attribute_Name
- Mathematically-dependent attributes are treated as ordinary instance-based operations at the moment



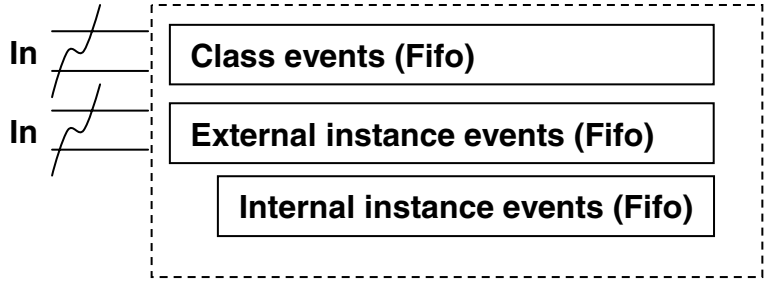


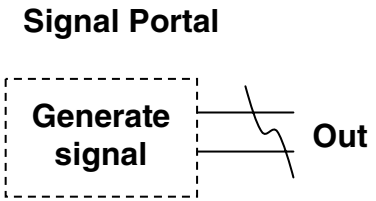
Design – Association

- Creation and deletion of associations has been designed
- Selections are designed
- All instances related are visible as read-only data and used by the Instance_Table
- Navigation is handled by the classes via a “Navigator_Portal”

Design – Event and Signal

- All instances of the same class share the same Event Queues
- Creation events are handled as ordinary events with an extra bit indicating creation
- Event data is coded and decoded by subprograms converting to and from a general data bus
- The bit-width of the data bus is worst-case
  - Maximum size of event data combinations
- Generating signals can be done immediately or delayed
- Signalling to other classes is handled by a “Signal_Portal” and follows the same principles as for attribute access and relationship navigation





Design – State Procedure

- Each action i.e. attribute access, relationship navigation, … is divided into one or several VHDL state machines
- State transitions between actions are also following the handshaking principles with Start and Done
- Example for an attribute read access
  1. Wait for Start
  2. Use the subprogram to produce the address from Instance_Id and Attribute_Name
  3. Wait for Done_Read
  4. Assign the variable with the result from the subprogram converting the general data into the specific attribute type
  5. Signal Done
- When working on “self” a local copy is used with exactly the attributes the Procedure is using
- The scope for variables is managed by the Block construction in VHDL

Experiences

- Design principles and patterns are finished
  - Some of them are also implemented and tested
- Execution principles are designed as presented
- The workload is more on structural parts in a VHDL architecture than compared to the Ada95 architecture
- The effort has been to keep the coding of the Procedures as simple as possible, this is achieved via the different kinds of “Portals”
  - Accesses
  - Relationship Navigation
  - …

Experiences

- These “Portals” can then be generated directly from the Interaction metamodel created from the first pass in the model compiler
  - The VHDL model compiler uses the same information as the Ada95 model compiler
- Two worlds meeting – “Software” and “Hardware”
  - Same logical view but very different implementation issues
  - Hardware engineers must understand the xUML semantics
- There is no information regarding Timing and Area usage at the moment

Questions & Answers?