

# UML in an Electronic System Level Design Methodology

Ananda Shankar Basu, Mauro Prevostini  
ALaRI, University of Lugano  
Lugano, Switzerland

Marcello Lajolo  
NEC Laboratories America, Inc.  
Princeton, NJ, USA  
*[lajolo@nec-labs.com](mailto:lajolo@nec-labs.com)*



# Agenda

- Motivation
- EDA for System-level Design
- The UML-based Design Flow
  - UML specifications
  - Exporting structural information
  - Code generation
  - Web-based interface
- Conclusions



# Motivation

- SOC integration is currently limited due to the missing link between system-level specification and design implementation
- There is a consensus toward the need to raise the level of abstraction of the system specification, but ...
- The EDA market still offers only a limited support for system-level design

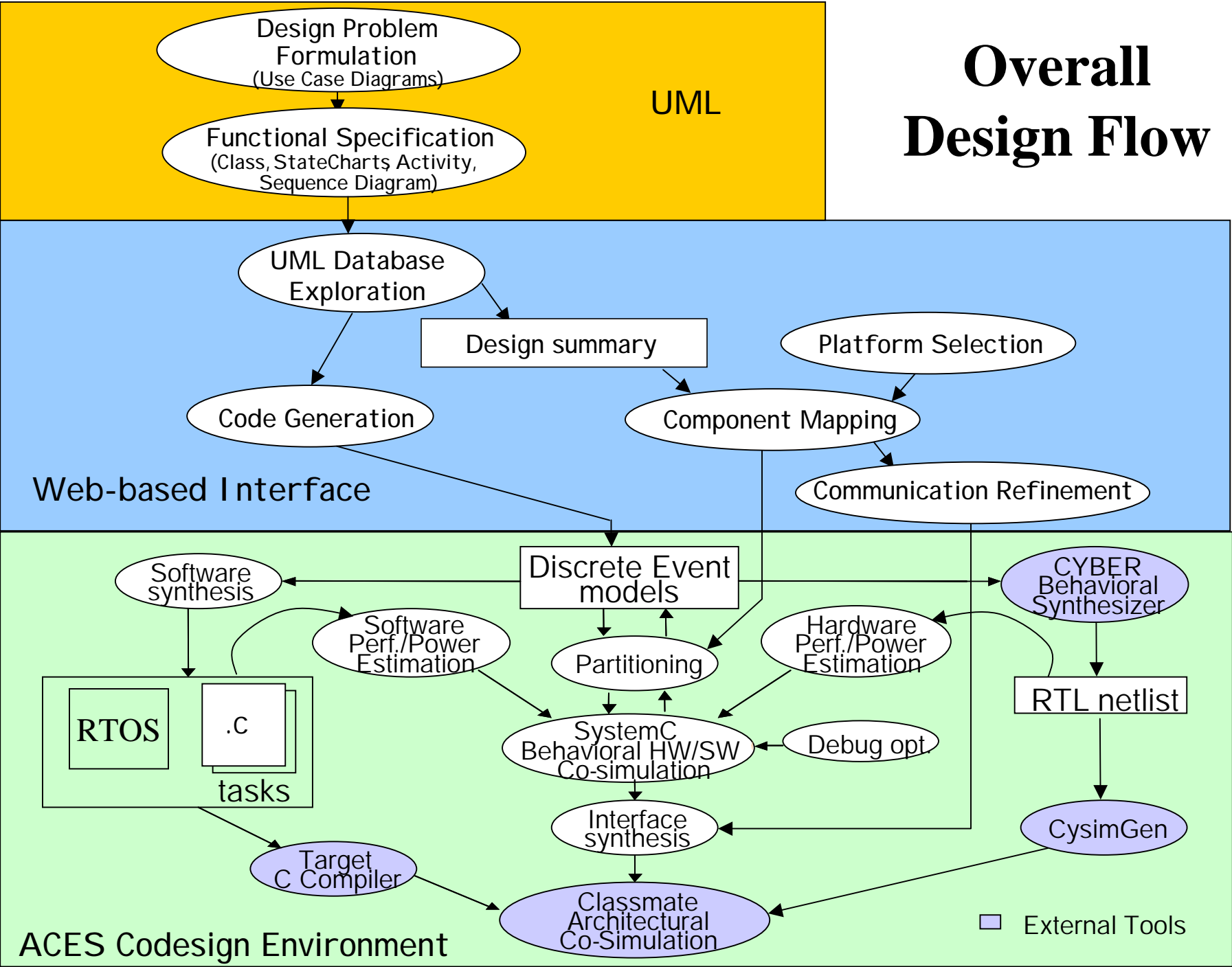


# EDA for System-level design

- Many commercial hw/sw co-verification tools are available (Mentor, CoWare, VAST, Virtio, Axys)
- The emerging area is the one of co-processor synthesis (Mentor-ASAP, CriticalBlue, Synfora)
- What is mostly missing is the support for initial analysis of hw/sw partitioning trade-offs



# Overall Design Flow



# Design Flow

- The application is modeled and analyzed in UML
- A web-based interface is used for browsing the UML repository and generates the input descriptions needed by the Codesign environment
- The Codesign phase uses behavioral hardware synthesis from SystemC generated from UML

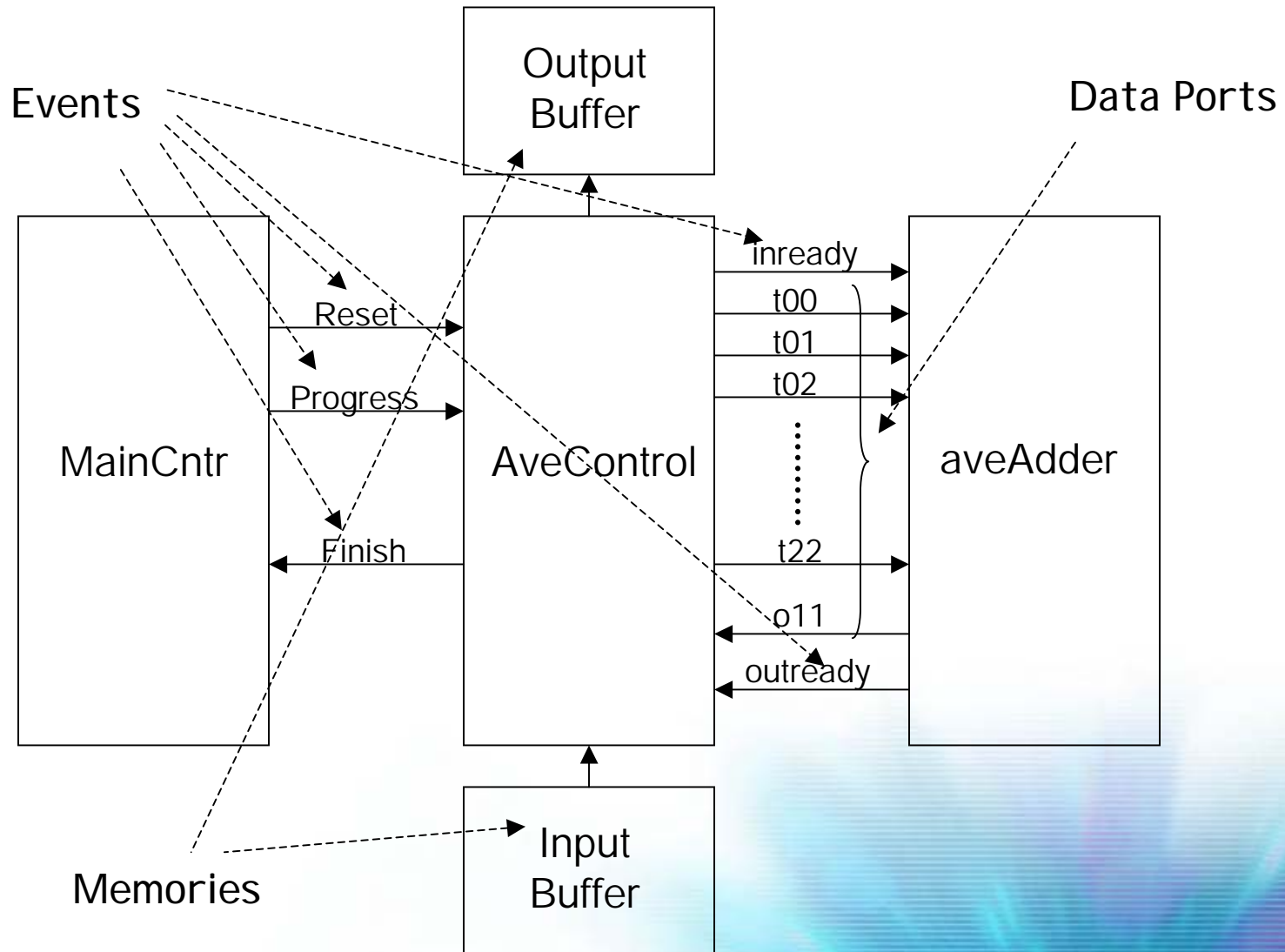


# UML Specifications

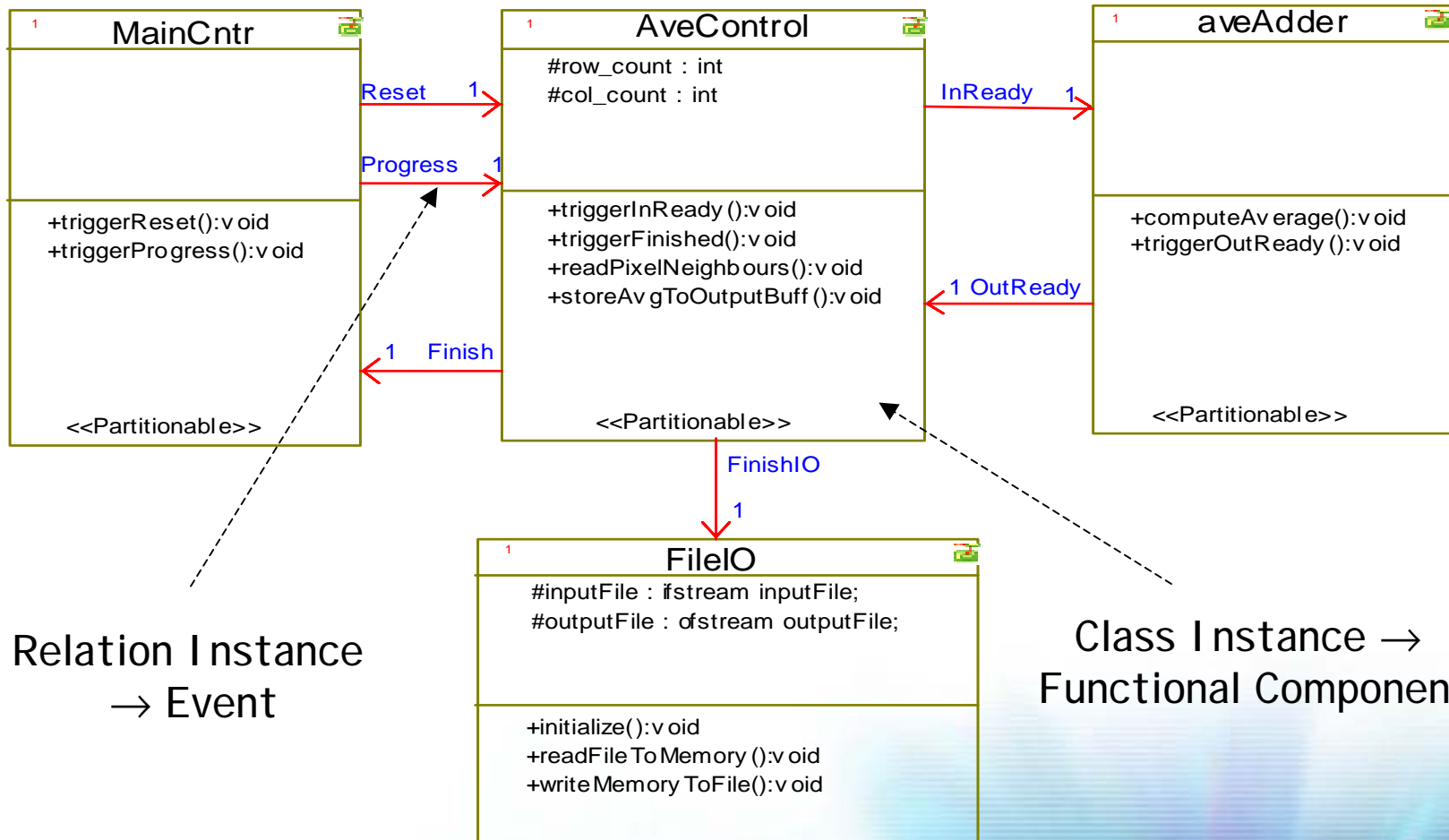
- Object model Diagrams
  - Collection of class instances used to describe the functional components in the system and their interconnections
- State Charts
  - Describe the behavior of a functional component
- Sequence Diagrams
  - Used to describe the expected behavior
  - Constraints express timing
  - Help to create testbenches



# An Example for Functional Specifications



# Object Model Diagram



Relation Instance  
→ Event

Class Instance →  
Functional Component

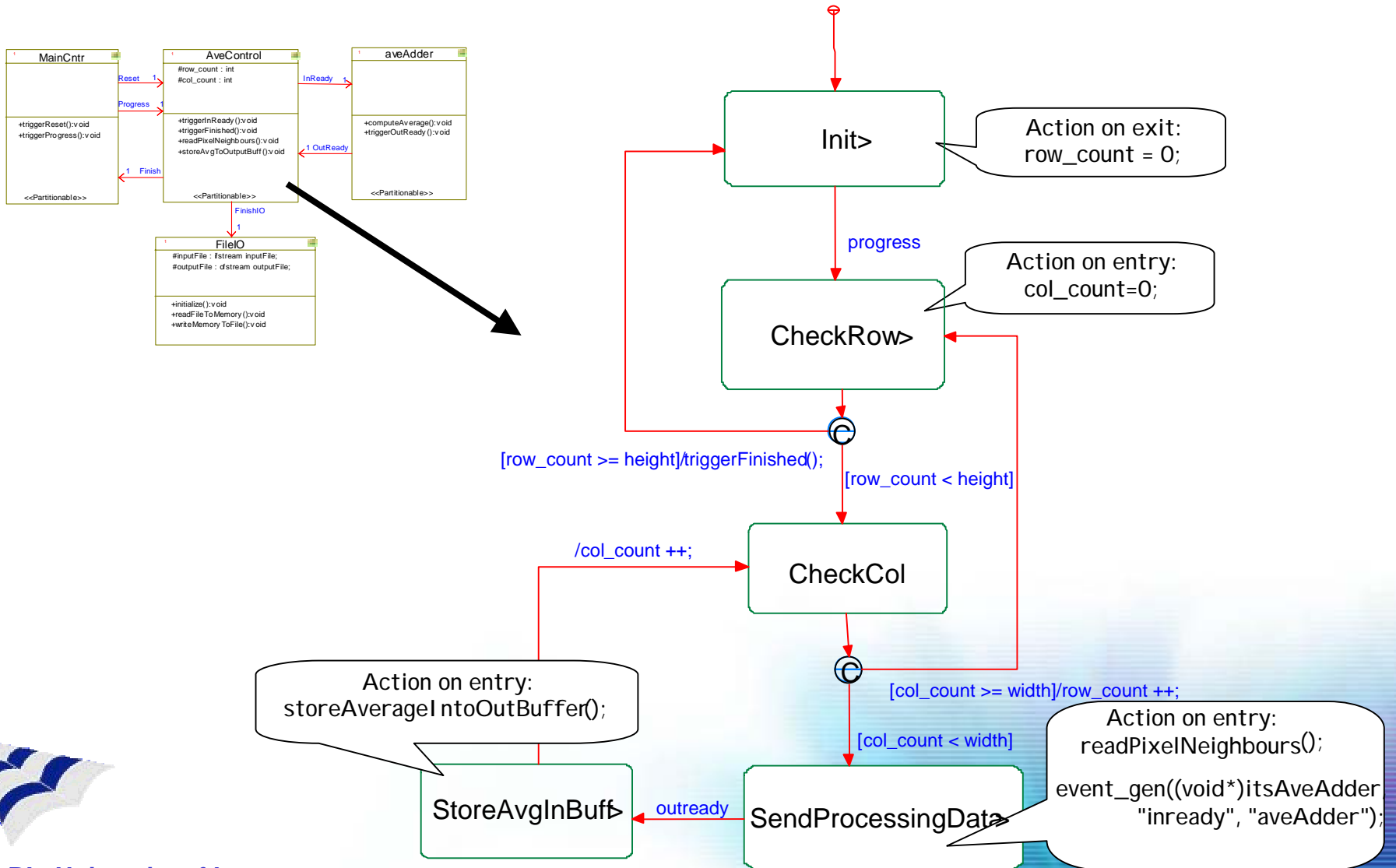


# Object Model Diagram - Properties

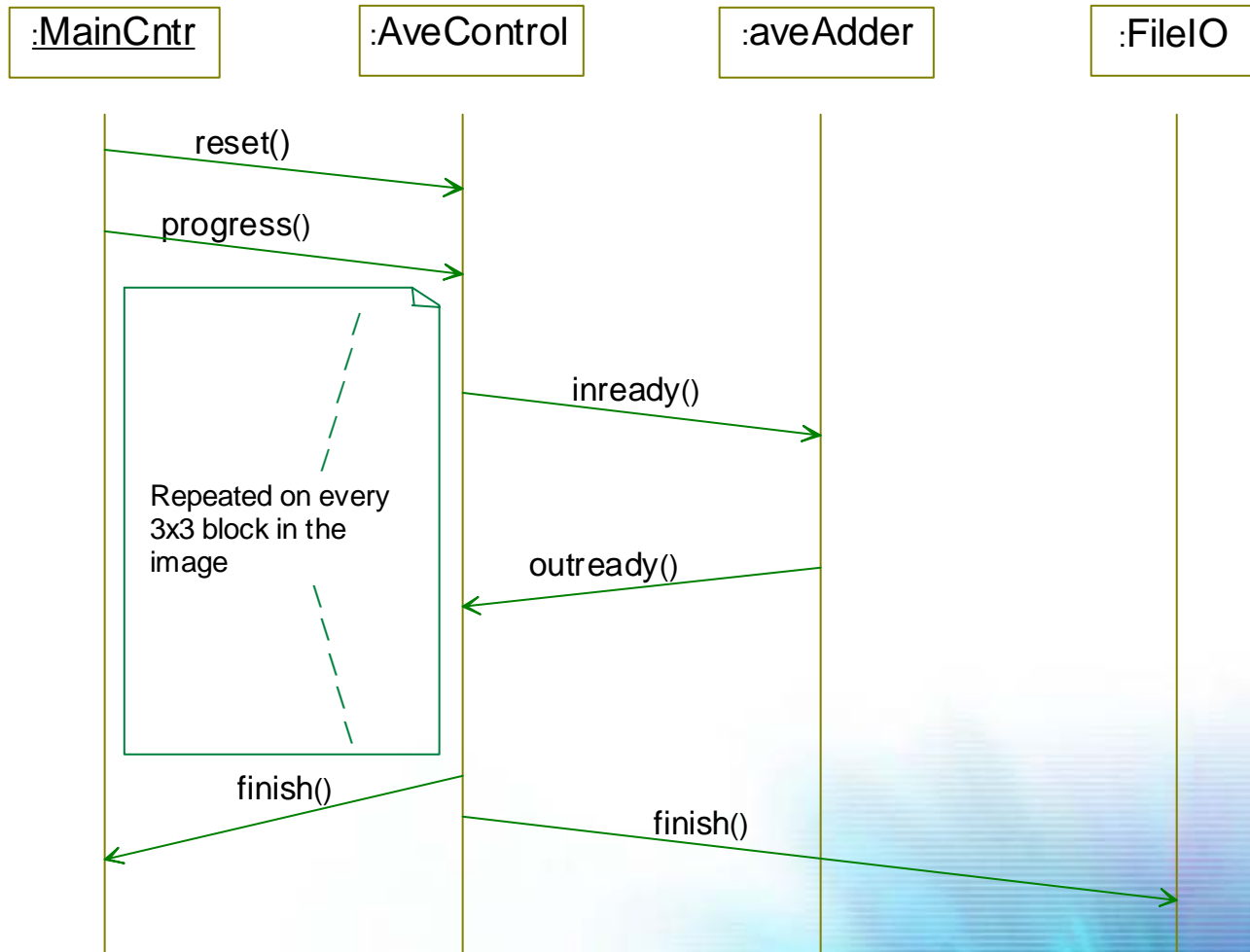
- One class instance corresponds to one functional component
- All functional components taking part in the hw/sw partitioning process must be identified with the *Partitionable* stereotype
- Communication among modules happens through:
  - Events (associated to graphical relationships)
  - Data ports (API calls in the StateCharts)
  - Global variables (API calls in the StateCharts)



# State Diagram for Module AveControl



# Sequence Diagram



# Extended UML API

- Used to provide a general UML semantics for managing events, data ports and shared data that can work across multiple UML platforms
- API functions can be called in:
  - Internal code of Class Instances of an OMD
  - Actions in StateCharts
  - User Defined Packages
- We have implemented and tested it using Rhapsody UML from I-Logix



# Extended UML API

API Macro Name	Description
<code>event_gen(<i>relation_name</i>, <i>event_name</i>)</code>	Generate an event through the specified relation instance
<code>write_dport(<i>port_name</i>, <i>value</i>)</code>	Write a value to a data port
<code>read_dport(<i>port_name</i>)</code>	Read from a data port
<code>write_shared_data(<i>var_name</i>, <i>value</i>)</code>	Write a value to a shared data
<code>read_shared_data(<i>var_name</i>)</code>	Read from a shared data
<code>lock_shared(<i>var_name</i>)</code>	Lock a shared data
<code>unlock_shared(<i>var_name</i>)</code>	Unlock a shared data
<code>check_shared_status(<i>var_name</i>)</code>	Check the status of a shared data



# Extended API Usage

- API called from user code
- For example, body of user function `storeAvgToOutputBuff()` uses APIs `read_dport` and `write_shared_data`

```
int result = read_dport(o11);  
write_shared_data(memory_out[row_count]  
                  [col_count], result);
```



# From UML to Codesign

- Code generation for synthesizable models
  - SystemC code is generated from *Partitionable* State Charts
- Export of structural information
  - Textual representation containing the set of modules and their interconnections



# Code Generation for AveControl

```
#include <AveControl.h>

// External memories
extern sc_int InputBuffer[height*width];
extern sc_int OutputBuffer[height*width];

SC_MODULE(AveControl) {
  sc_in_clk clk;
  sc_in<bool> rst;

  // Input terminals
  sc_in<sc_int<8>> o11; // input data
  sc_in<bool> outready; // input event
  sc_in<bool> Reset; // " "
  sc_in<bool> Progress; // " "

  // Output terminals
  sc_out<bool> Finish; // output event
  sc_out<bool> inready; // " "
  sc_out<sc_int<8>> t00; // output data
  ... Omitted ...

  SC_CTOR(AveControl) {
    SC_CTHREAD(main,clk.pos());
    watching(rst.delayed() == 0);
  }

  void main(void) {
Init:
    aces_wait(Progress);
    row_count=0;
    goto CheckRow;

CheckRow:
    col_count=0;
    if ( row_count < height ) {
        goto CheckCol;
    }
    else { // Send Finish
        event_gen((void*)itsMainCntr, "finish", "MainCntr");
        event_gen((void*)itsFileIO, "finish", "FileIO");
        goto Init;
    }

CheckCol:
    ... Omitted ...
  }
};
```

Event ports

External data

Data port

FSM behavior  
(unstructured style)

# Code Generation Algorithm

```
codeGenerate(state *S) {  
  1. If S is visited, return;  
  2. Mark S as visited.  
  3. Issue code specified in the action-on-entry section  
  4. Get out transitions {T} from state S;  
  5. {U} = empty;  
  6. for each out-transition 't' of {T} do {  
    if 't' is conditional {  
      issue code specified in the action-on-exit section;  
      s_t = target state if condition is true;  
      s_f = target state if condition is false;  
      issue if-then-else with goto label as 's_t' or 's_f' depending on condition;  
      insert 's_t', 's_f' in {U};  
    } else {  
      s = target state of 't', insert 's' in {U};  
      if 't' is triggered by event 'e' {  
        issue wait on event 'e';  
      }  
      issue code specified in the action-on-exit section;  
      issue goto with label as 's';  
    }  
    issue code specified in the action section of transition 't';  
  }  
  for each 'u' in {U} do  
    codeGenerate(u);  
}
```

# Event Semantics in UML

1. An event is created when it is sent by one object to another
2. It is then queued on the queue of the target object thread
3. An event that gets to the head of the queue is dispatched to the target object
4. The event is processed by the receiving module and then deleted by the execution framework



# Event Semantics: drawbacks

- The event semantics (SDL-like) is not adequate for hardware modeling
- A real Discrete Event engine would be needed in order to simulate real hardware:
  - Global event queue
  - Events ordered based on their timestamps
- For this reason we use UML only for Specifications but not for hw/sw co-simulation and validation



# HTML page generated from UML Specs

The screenshot displays the ACES Interface Synthesizer web application in a Netscape browser. The page features the NEC logo and the title "ACES Interface Synthesizer". Below the title, there is a section for "Overview of the System:" which contains a block diagram. This diagram shows a "MainCtrl" block connected to an "AveControl" block. The "AveControl" block is connected to an "Output Buffer" block and an "aveAdder" block. A "Software1 (VBS0)" block is connected to a "Bus<sub>1</sub>" block, which is in turn connected to an "M<sub>1</sub>" block and a "BusIntfc1" block. The "BusIntfc1" block is connected to the "aveAdder" block. To the right of the main diagram, there is a smaller diagram titled "Overview of the template:" which shows a "M<sub>1</sub>" block connected to a "Hardware1" block.

Below the diagrams, there is a section titled "Choose the implementation of the following modules:" which contains a table with the following data:

Module	Implementation	Link
MainCtrl:	Software1	<a href="#">source code</a>
AveControl:	Software1	<a href="#">source code</a>
ave2AdderII:	Software1	<a href="#">source code</a>

At the bottom of the page, there is a button labeled "GENERATE PARTITION".



# Mapping on a dual-processor architecture

The image displays the ACES Interface Synthesizer software interface, which is used for mapping a system onto a dual-processor architecture. The interface is divided into several sections:

- Overview of the project:** A block diagram showing the main components: MainCtrl, AveControl, aveAdder, Output Buffer, and Input Buffer. AveControl is connected to both the Output Buffer and the Input Buffer. aveAdder is connected to AveControl via multiple data paths.
- Overview of the template:** A diagram showing the mapping of the project components onto a dual-processor architecture. It features three buses (Bus<sub>1</sub>, Bus<sub>2</sub>, Bus<sub>3</sub>) and three modules (M<sub>1</sub>, M<sub>2</sub>, M<sub>3</sub>). Each module is connected to a bus and contains a Bridge and Hardware component. Software1 (V850) is connected to Bus<sub>1</sub>, Software2 (V850) to Bus<sub>2</sub>, and Hardware3 to Bus<sub>3</sub>.
- Configuration window:** A window titled "Choose the implementation of the following modules:" with a table for selecting implementations:

Module	Implementation	Generate Code
MainCtrl:	Software1	<a href="#">GENERATE CODE</a>
AveControl:	Software1	<a href="#">GENERATE CODE</a>
ave2AdderII:	Hardware1	<a href="#">GENERATE CODE</a>
	Hardware2	
	Hardware3	

At the bottom of the configuration window is a button labeled "GENERATE PARTITION".



# Communication Refinement

Choose the implementation for the following signals:

LIST OF SIGNALS				
CPU 1				
Type	Signal Name	Input for	Output of	Implementation
I	<i>MainCntr_e_RW_Reset</i>	AveControl	MainCntr	Memory Mapped
I	<i>ave2AdderII_e_011</i>	AveControl	ave2AdderII	Memory Mapped
I	<i>ave2AdderII_e_outready</i>	AveControl	ave2AdderII	Memory Mapped
I	<i>MainCntr_e_RW_Progress</i>	AveControl	MainCntr	Memory Mapped
I	<i>Finish</i>	MainCntr	OUT	Memory Mapped
I	<i>net_MainCntr0e_selftrigger</i>	MainCntr	MainCntr	Memory Mapped
O	<i>MainCntr_e_RW_Reset</i>	AveControl	MainCntr	Memory Mapped
O	<i>MainCntr_e_RW_Progress</i>	AveControl	MainCntr	Memory Mapped
O	<i>ave2AdderII_e_100</i>	ave2AdderII	AveControl	Memory Mapped

# UML Pros & Cons in a Platform Based Design Context

- Pros
  - It is platform independent
  - It helps team members to share relevant information during the various design phases
- Cons
  - The event semantics in State Charts (SDL-like) is not adequate for hardware modeling
  - Each tool vendor supports his own customized enhancements of UML



# Conclusions

- We have presented a methodology using UML for modeling embedded systems at the system level
- An extended and portable UML API has been proposed for describing functional communication
- We have shown an integration with an existing hardware/software codesign technology

