6. European SystemC Users Group Meeting

Modelling Cycle-Accurate Hardware with Matlab/Simulink using SystemC

Frank Czerner, TechnoTeam Bildverarbeitung GmbH Ilmenau

Jens Zellmann, Institute of Microelectronic and Mechatronic Systems gGmbH Ilmenau

Modelling Cycle-Accurate Hardware with Matlab/Simulink using SystemC

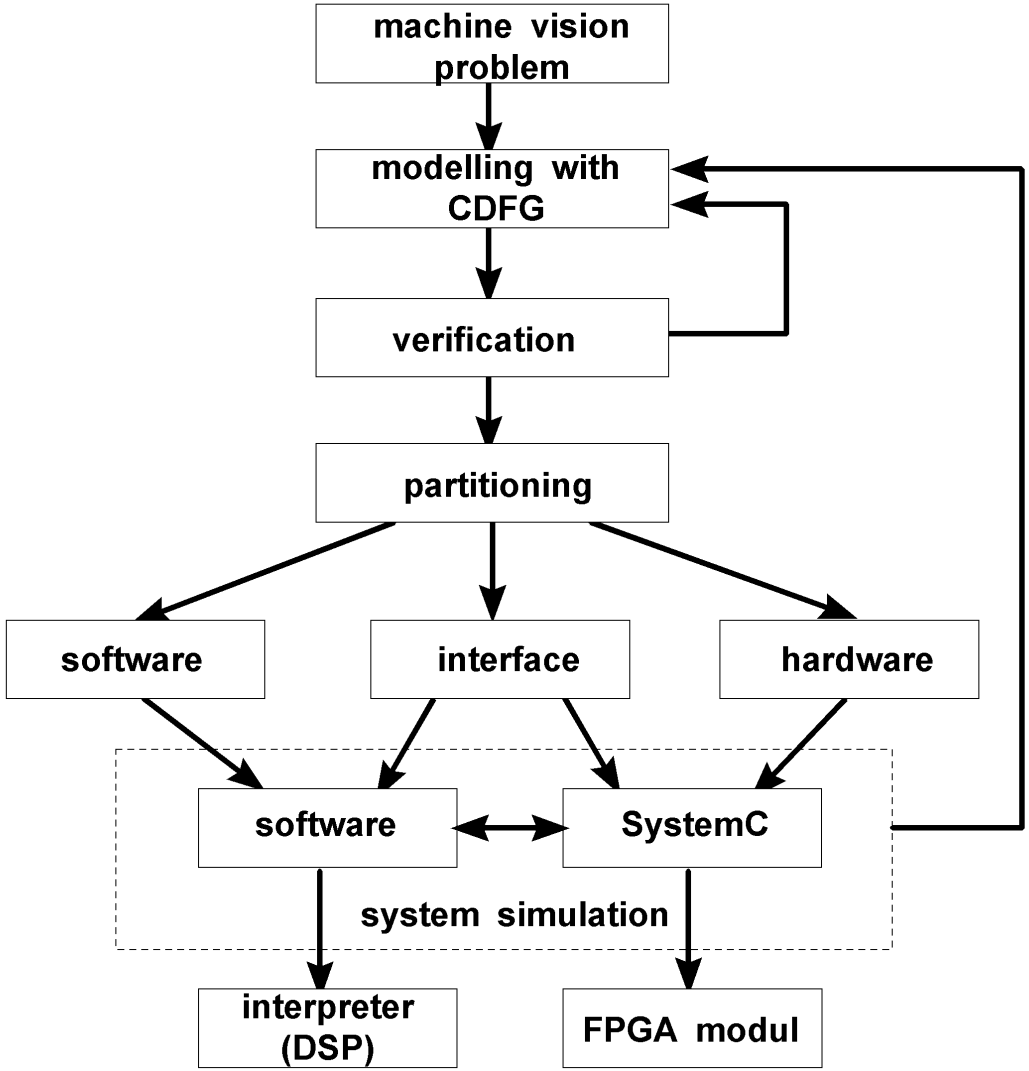
- Overwiew
- How to integrate SystemC Models in Simulink ?
- How to generate multible persistent Objects of a SystemC Model ?
- Required extensions of the SystemC-Kernel
- Examples
- Conclusions and Recommendations

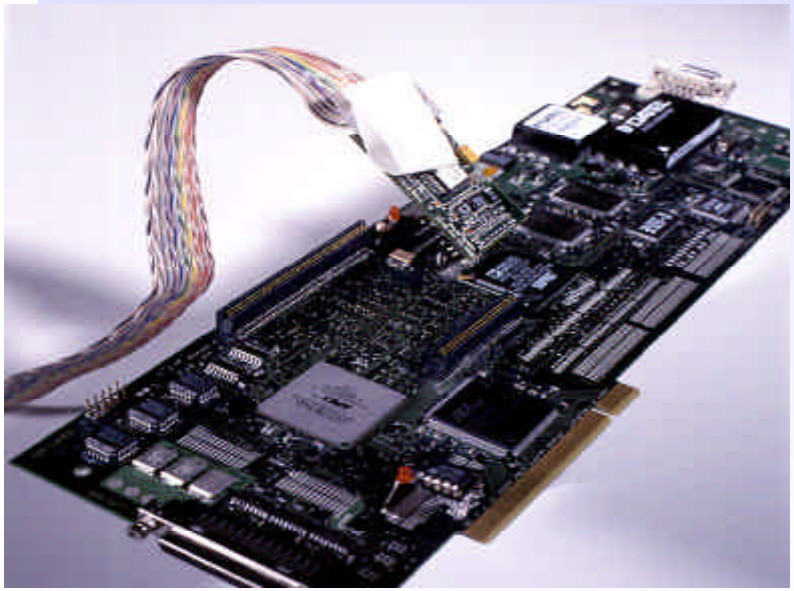
Overview

?

ycle-accurate hardware

machine vision



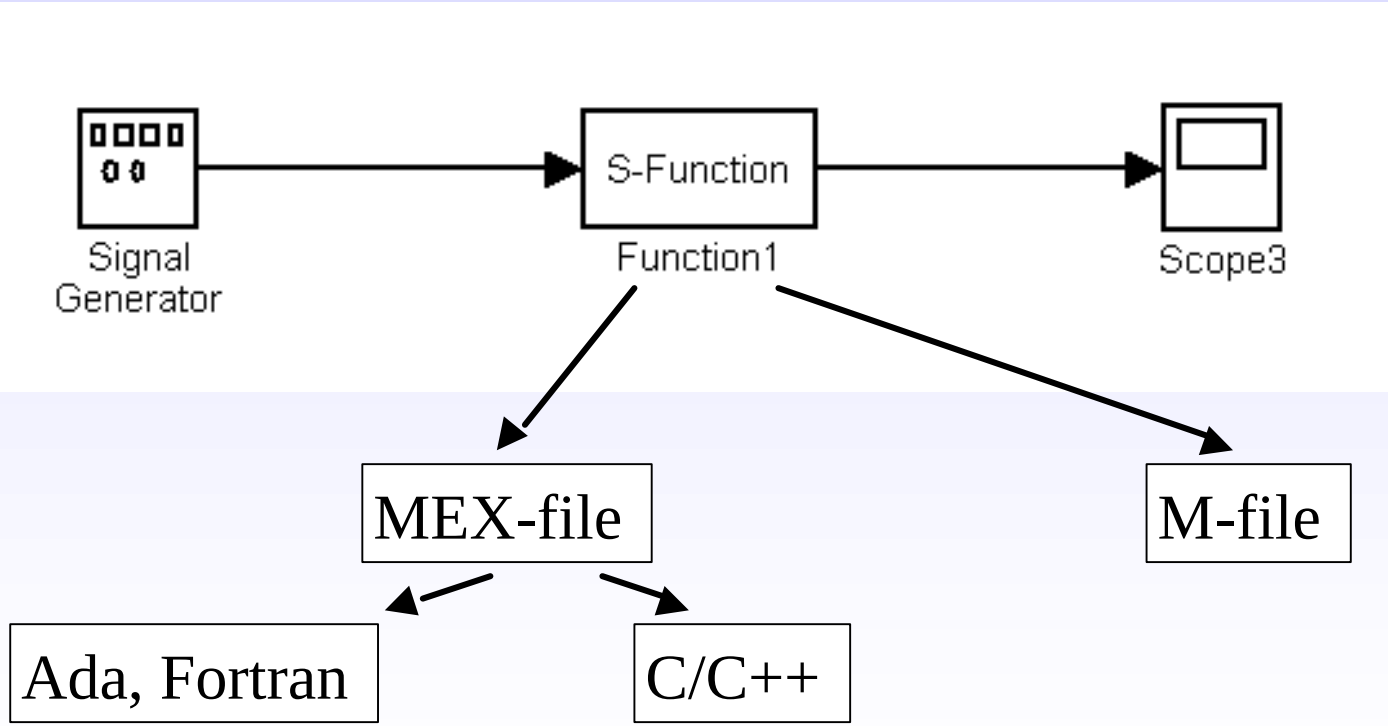


Overview

- **Where we are using SystemC ?**
  - **modelling and verification of cycle-accurate hardware components within codesigns**
  - **new methodology for building machine vision applications**
- **Why using Matlab/Simulink?**
  - **transfering this approach to a commonly used tool with a lot of libraries**
  - **addressing other signal processing applications (i.e. signal preprocessign in smart sensors)**
  - **basic support for hardware integration through third-party Tools**

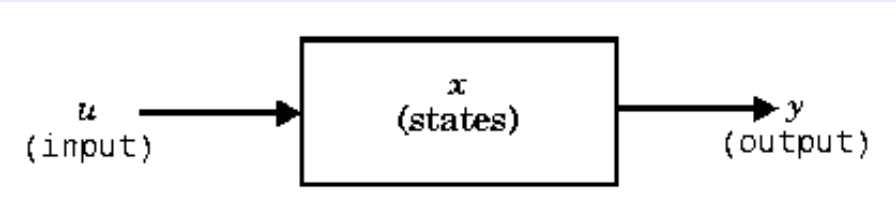
How to integrate SystemC Models in Simulink ?

- S-Functions provide a powerful mechanism for extending Simulink
- With S-Functions it is possible to add customized blocks to Simulink models



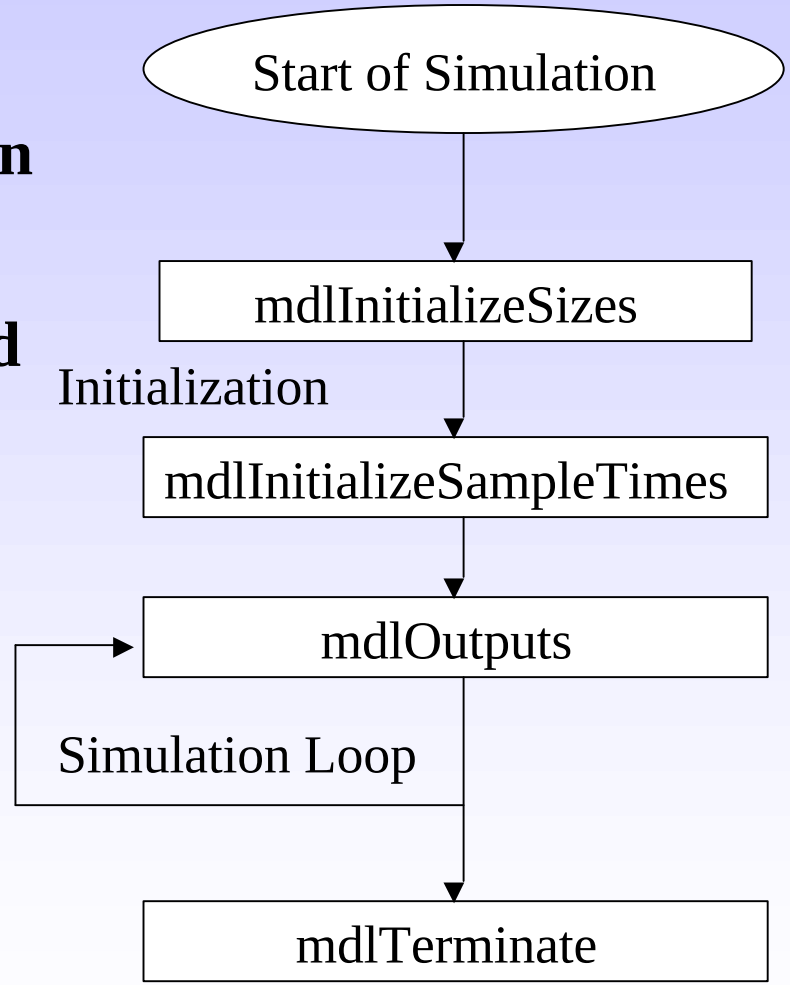
How to integrate SystemC Models in Simulink ?

- **using the fixed-step, discrete time solver**
- **simulation is controlled by duration time and step size**
- **output is calculated from input and from states at each time step**



$$y = f(t, x, u)$$

- **the mdlOutput function is called at each simulation step**



How to integrate SystemC Models in Simulink ?

- simulation of SystemC models is usually „stand alone“
- initialization and simulation loop is normally processed in one function

```
#include <systemc.h>
#include "stimulus.h"
#include "display.h"
#include "fir.h"

int sc_main (int argc , char *argv[]) {
  sc_clock         clock;
  sc_signal<bool>  reset;
  ...

  stimulus stimulus1("stimulus_block");
  stimulus1.reset(reset);
  stimulus1.clk(clock.signal());
  ...

  fir fir1( "process_body");
  fir1.reset(reset);
  fir1.clk(clock);
  ...

  display  display1 ( "display");
  display1.output_data_ready(output_data_ready);
  display1.result(result);

  sc_start(clock, -1);
  return 0;
}
```

How to integrate SystemC Models in Simulink ?

Integration in Simulink needs:

- a stepwise simulation cycle by cycle, which can be executed in mdlOutputs Function
- initialization of the SystemC kernel must be seperated from simulation

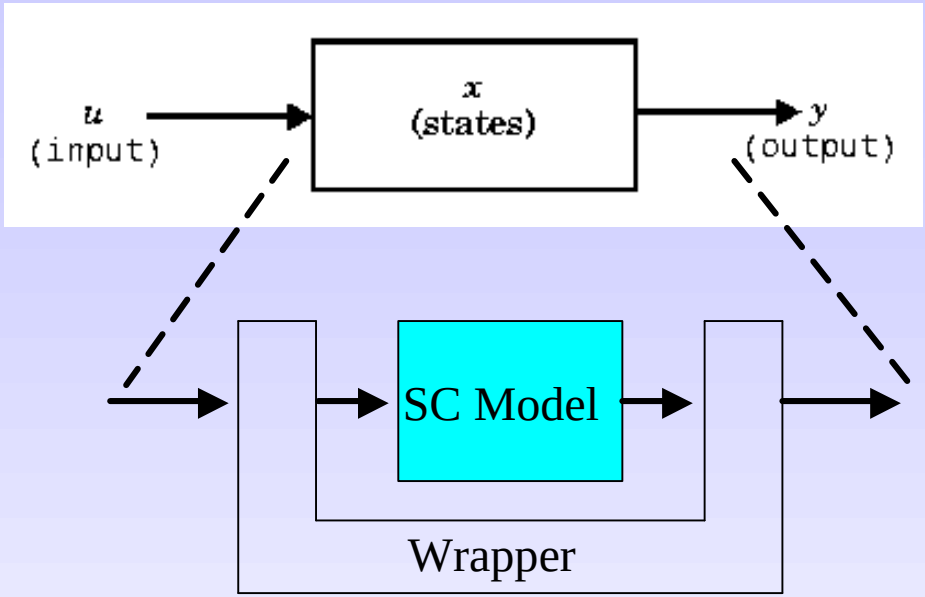
using the SystemC functions sc_initialize() and sc_cycle()

```
// init the kernel
pTmpSimContext = sc_free_curr_simcontext();
pModuleSimContext->initialize();

...

// simulate one cycle
(*clk).write(true);
pModuleSimContext->cycle(sc_time(10.0,true));
(*clk).write(false);
pModuleSimContext->cycle(sc_time(10.0,true));
```

How to integrate SystemC Models in Simulink ?



Functions:

- InitModule
- SimStep
- DestroyModule

- a wrapper is needed to connect Simulink ports to a SystemC-Block
- converting Simulink data types to SystemC signals and vice versa
- initializing of the SystemC-Kernel
- converting events (function call from Simulink to sc_cycle())
- provide a DLL interface to the Simulink S-Function

How to generate multible persistent Objects of a SystemC Model ?

- one object of the SystemC module is created dynamically
- a pointer to this object is stored in a vector

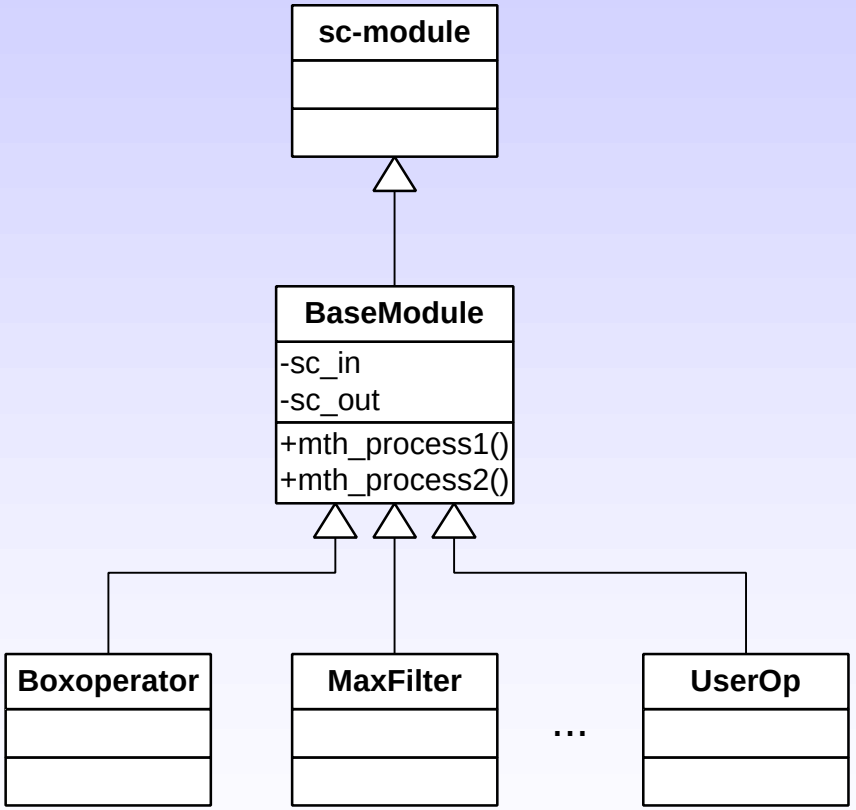
```
//build hardware module
ssGetPWork(S)[0]=BuildModule(chModuleName,DoTrace);
```

- the pointer can be retrieved in any callback method to access the object

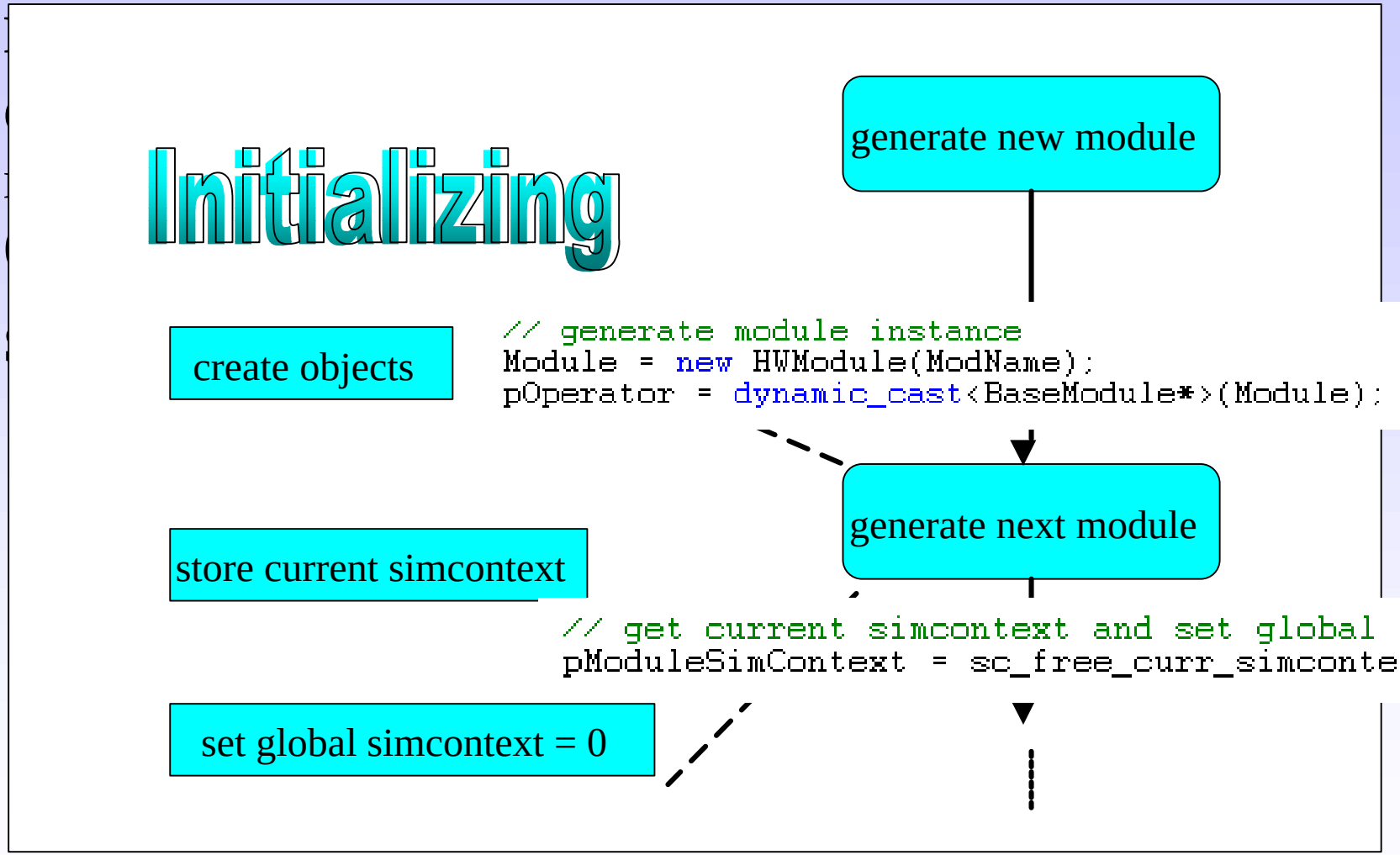
```
// get the pointer to the hardware module
void *pHWModule = ssGetPWork(S)[0];
//perform one simulation step
TSimStep(pHWModule,pBox8Inputs,pBox8Outputs);
```

Conditions to build operator libraries

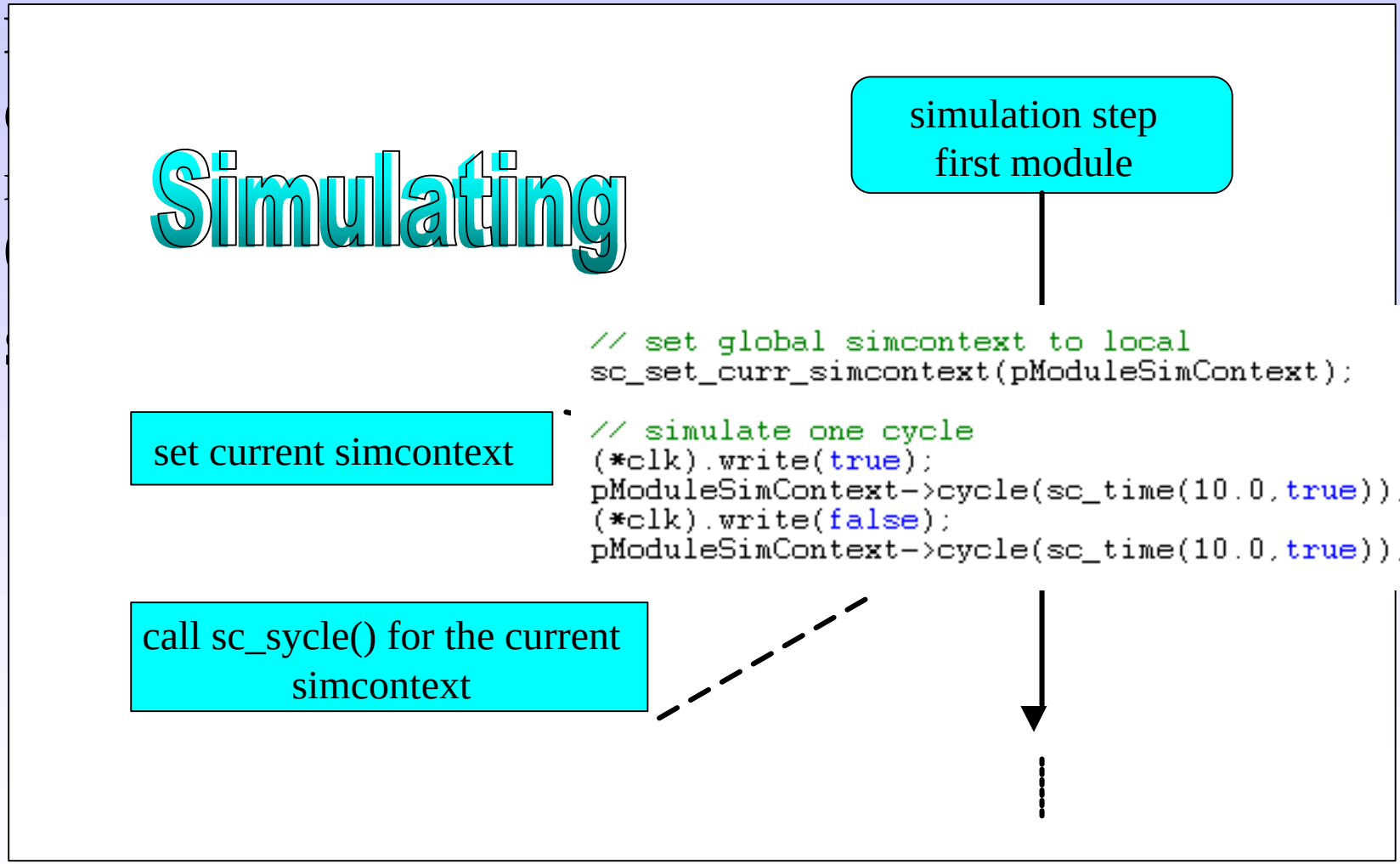
- using a module base class
- derive all modules from this base class
- wrapper works with base class objects



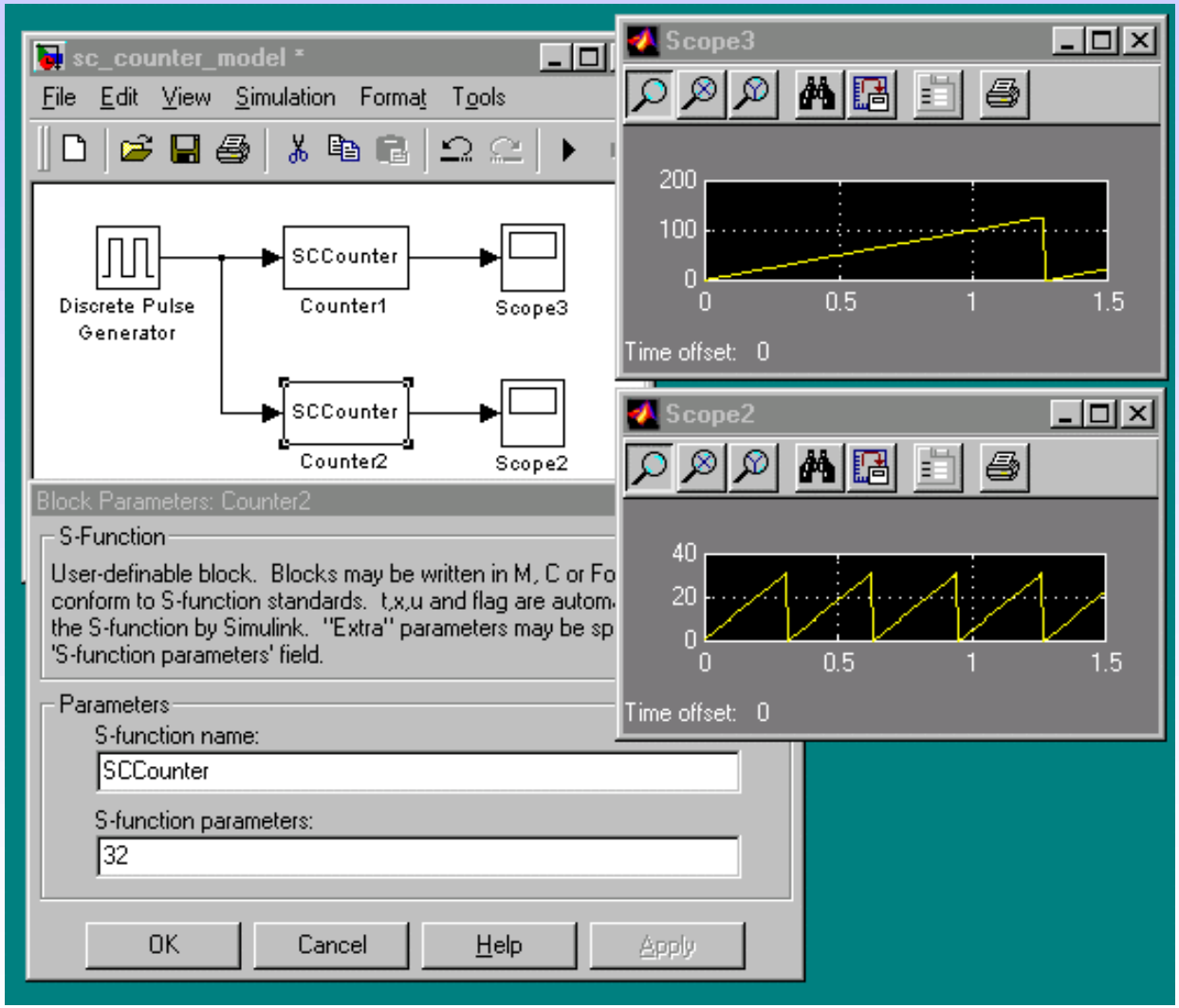
Required extensions of the SystemC-Kernel



Required extensions of the SystemC-Kernel

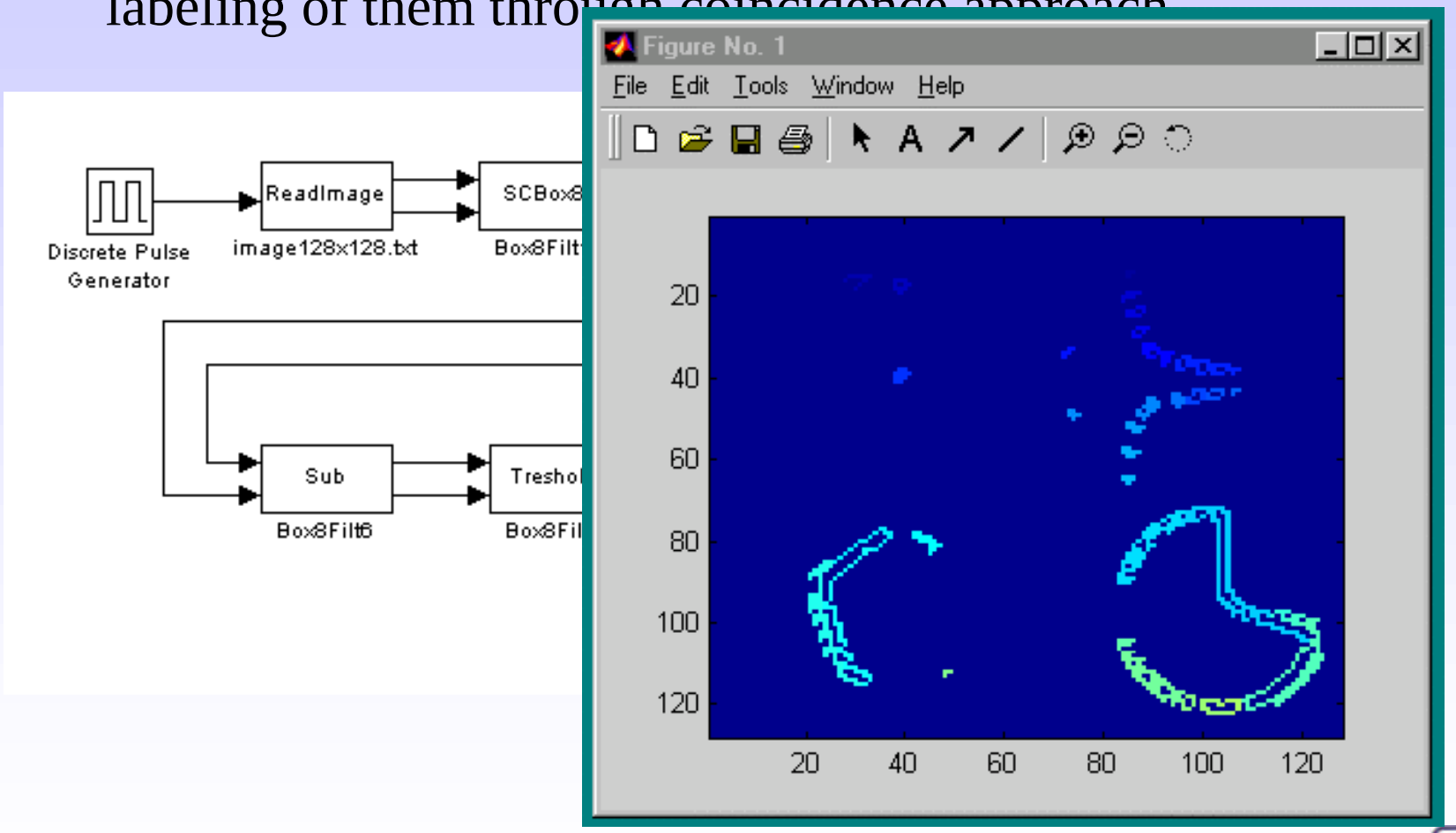


Example 1 - Multible Instances of a Counter Model



Example 2 - Object Pixel Detection

Model of a dataflow for detecting object pixels and labeling of them through coincidence approach



Conclusions and Recommendations

• models are reusable in the design environment for machine vision described before to build applications from tested operators

• one required functionality for hardware/software codesign in future projects is realized

• possibility of extending FPGA-Libraries from some vendors (XILINX, ALTERA)

Future Work

• modelling heterogenous systems

• hardware/software partitioning (possibly automated)

• code generation