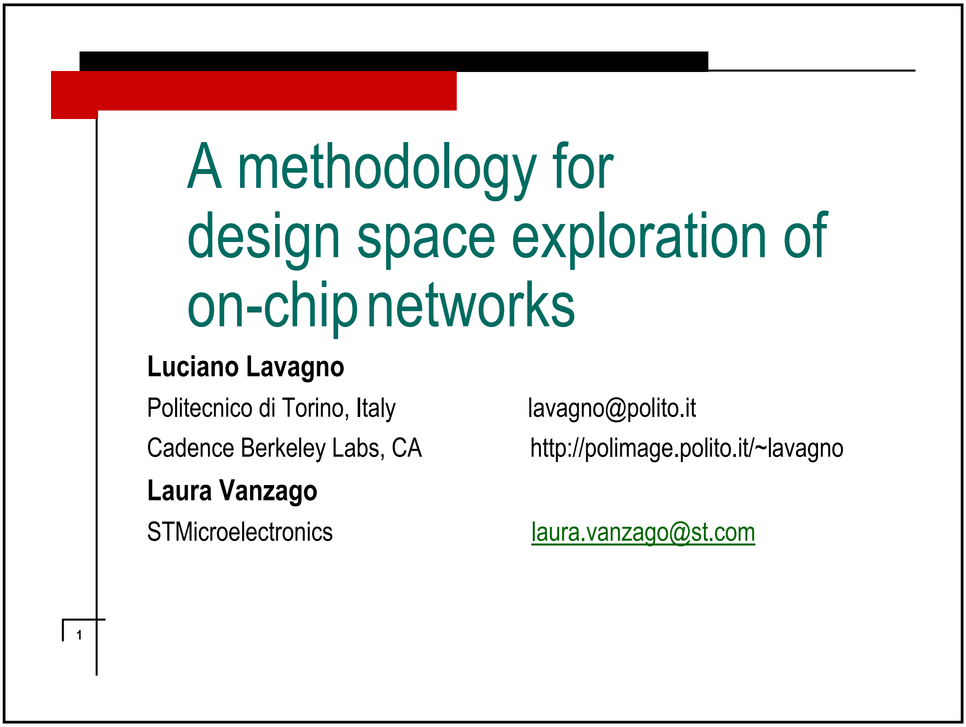
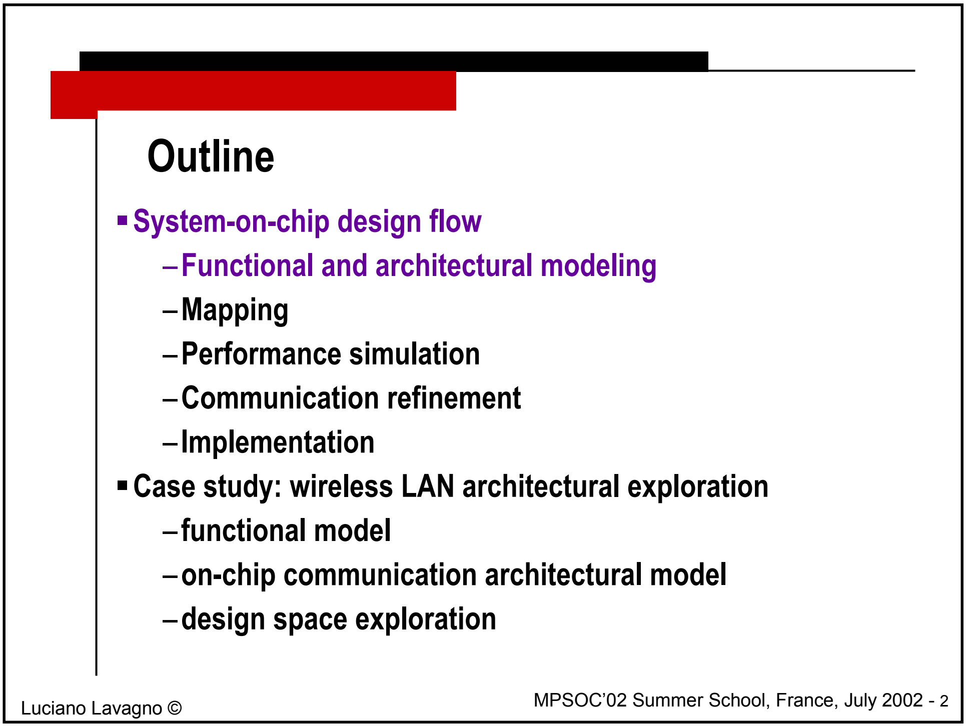


A methodology for design space exploration of on-chip networks

Luciano Lavagno
Politecnico di Torino, Italy lavagno@polito.it
Cadence Berkeley Labs, CA http://polimage.polito.it/~lavagno

Laura Vanzago
STMicroelectronics laura.vanzago@st.com



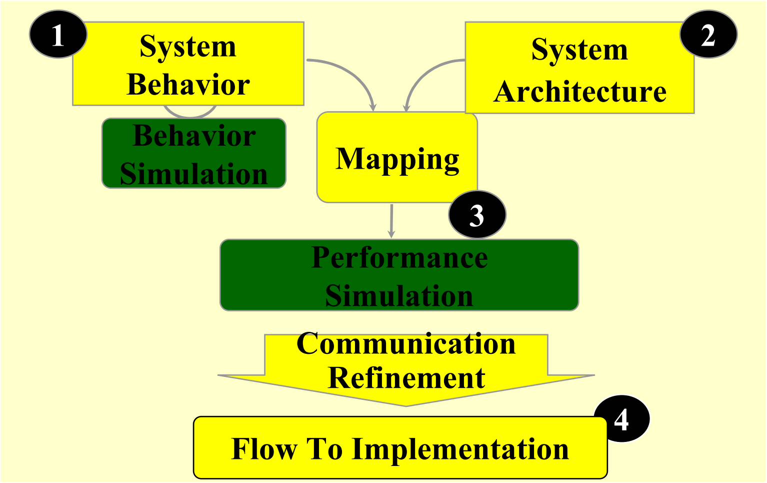
Outline

- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration

The System-On-Chip Design Flow

- **Specify:**
  - *What does the customer really want?*
- **Architect:**
  - *What is the most cost and performance effective architecture to implement it?*
  - *What existing components can I adapt and re-use?*
- **Evaluate:**
  - *What is the performance impact of a cheaper architecture?*
- **Implement:**
  - *What can I generate automatically from libraries and customization?*
- *Idea: separate computation, communication and performance*

The System-On-Chip Design Flow



1 System Behavior

Behavior Simulation

2 System Architecture

Mapping

3

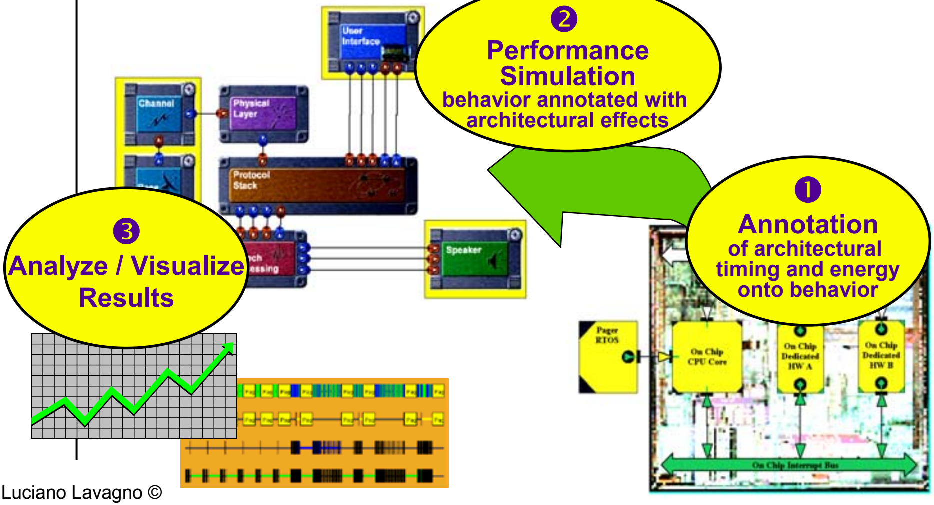
Performance Simulation

Communication Refinement

4

Flow To Implementation

The System-On-Chip Design Flow

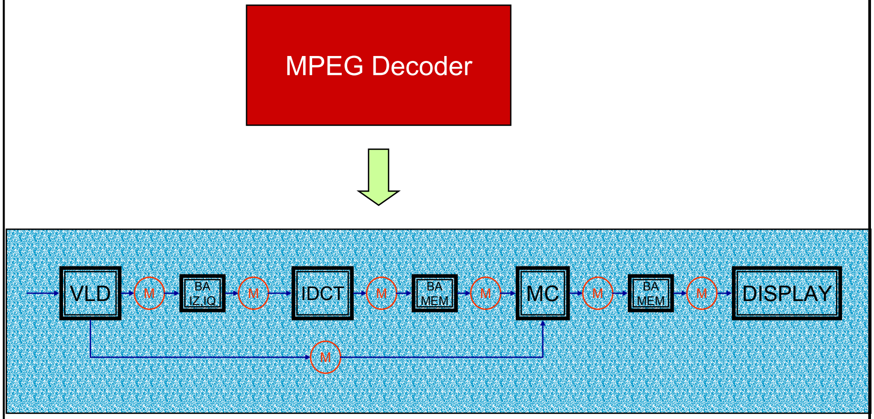


➋ Performance Simulation
behavior annotated with architectural effects

➊ Annotation
of architectural timing and energy onto behavior

➌ Analyze / Visualize Results

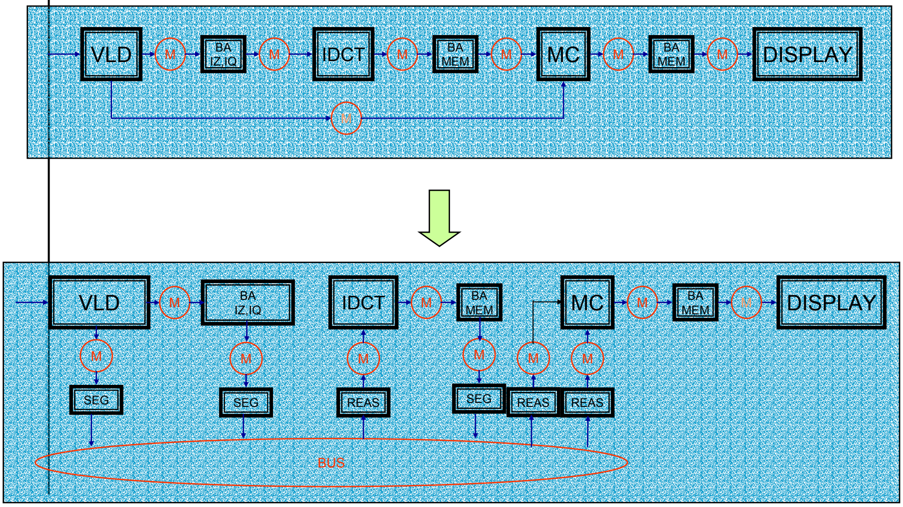
Functional Modeling



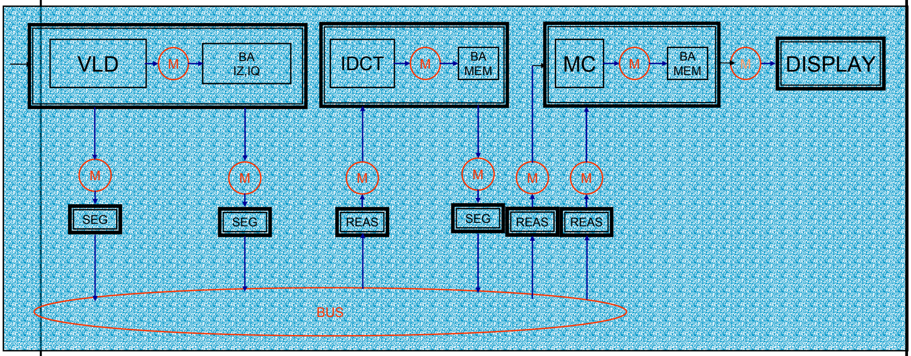
MPEG Decoder

VLD → M → BA IZ,IQ → M → IDCT → M → BA MEM → M → MC → M → BA MEM → M → DISPLAY

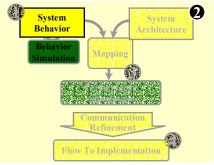
Communication Refinement

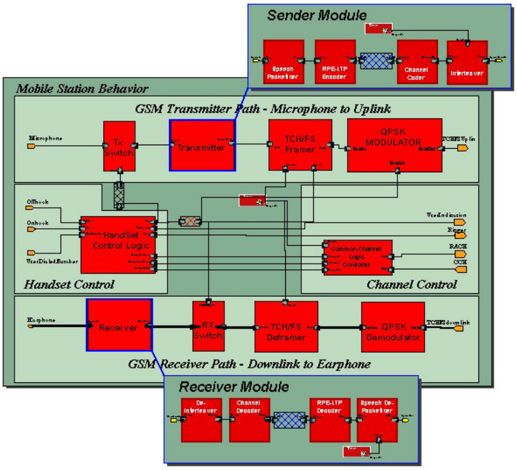


Optimization

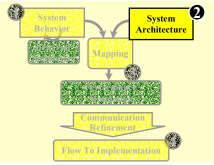


Functional modeling

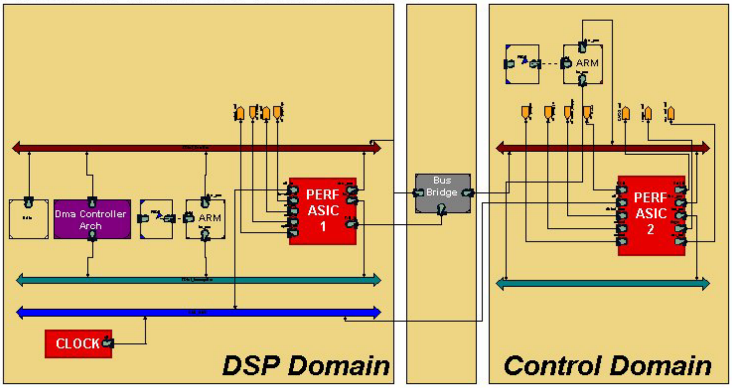




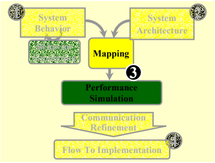
Architectural Modeling

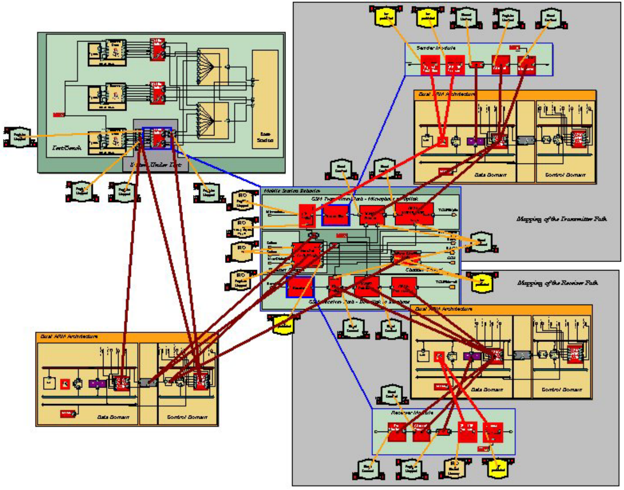


Double Processor Architecture



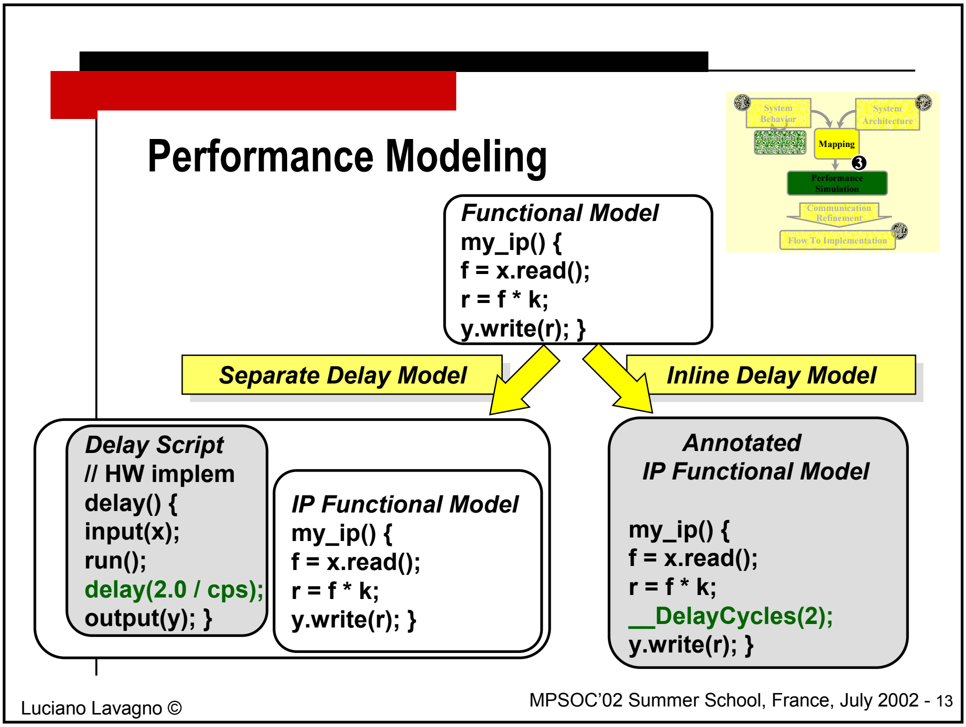
Mapping





Outline

- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration



Performance Modeling

Functional Model

```
my_ip() {
f = x.read();
r = f * k;
y.write(r); }
```

Separate Delay Model

Delay Script

```
// HW implem
delay() {
input(x);
run();
delay(2.0 / cps);
output(y); }
```

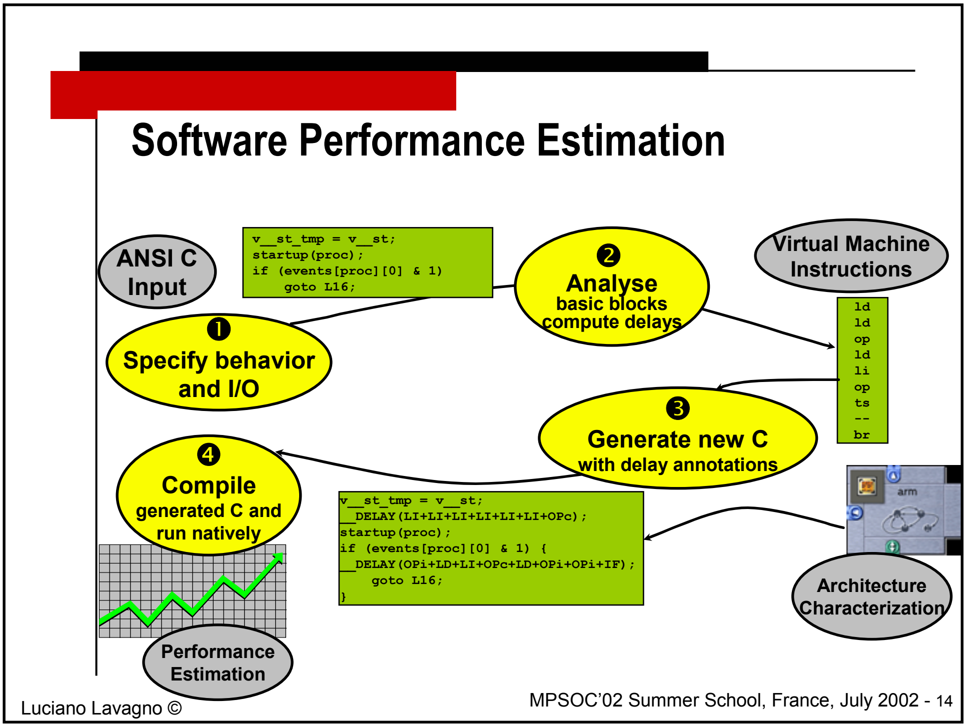
IP Functional Model

```
my_ip() {
f = x.read();
r = f * k;
y.write(r); }
```

Inline Delay Model

Annotated IP Functional Model

```
my_ip() {
f = x.read();
r = f * k;
__DelayCycles(2);
y.write(r); }
```



Software Performance Estimation

ANSI C Input

❶ Specify behavior and I/O

```
v__st_tmp = v__st;
startup(proc);
if (events[proc][0] & 1)
    goto L16;
```

❷ Analyse basic blocks compute delays

Virtual Machine Instructions

```
ld
ld
op
ld
li
op
ts
--
br
```

❸ Generate new C with delay annotations

```
v__st_tmp = v__st;
__DELAY(LI+LI+LI+LI+LI+LI+OPc);
startup(proc);
if (events[proc][0] & 1) {
__DELAY(OPi+LD+LI+OPc+LD+OPi+OPi+IF);
    goto L16;
}
```

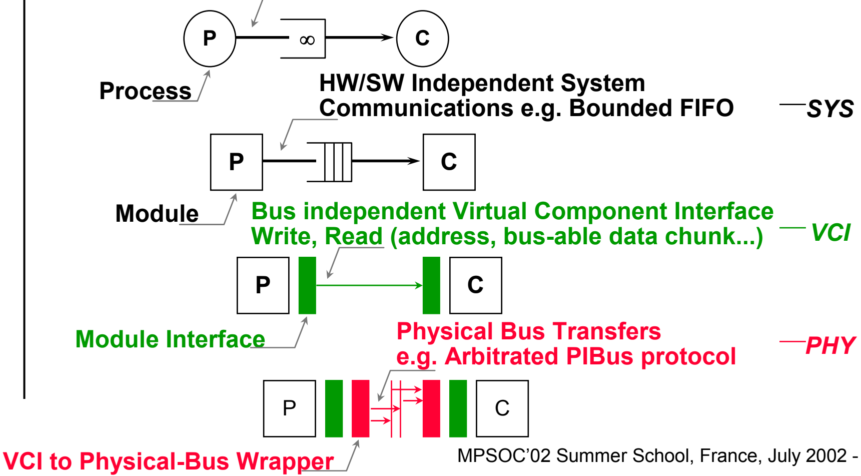
Architecture Characterization

❹ Compile generated C and run natively

Performance Estimation

Communication refinement

Delay Independent API
e.g. unbounded FIFO Write, Read (vector of «any» type) — *APP*



HW/SW Independent System
Communications e.g. Bounded FIFO — *SYS*

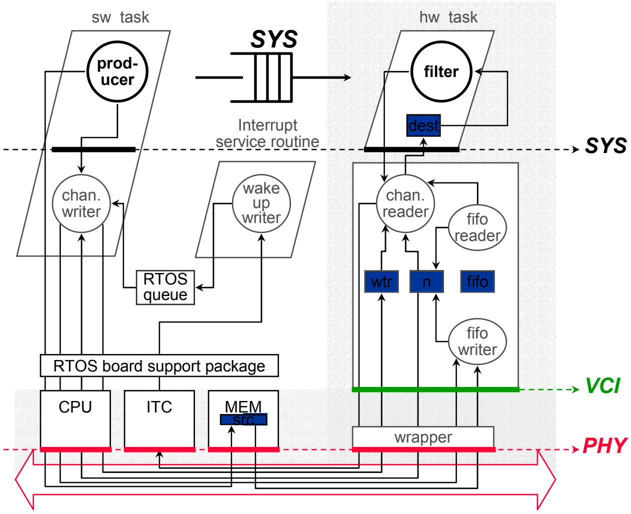
Bus independent Virtual Component Interface
Write, Read (address, bus-able data chunk...) — *VCI*

Module Interface

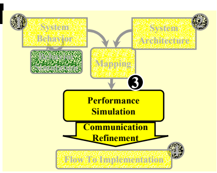
Physical Bus Transfers
e.g. Arbitrated PIBus protocol — *PHY*

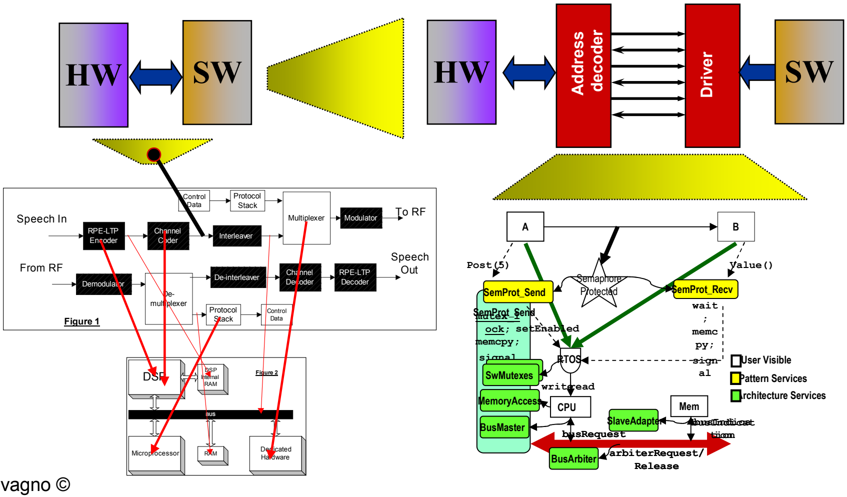
VCI to Physical-Bus Wrapper

Communication refinement

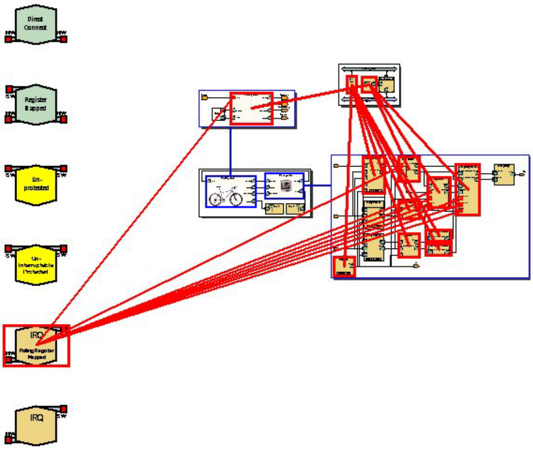


Communication refinement

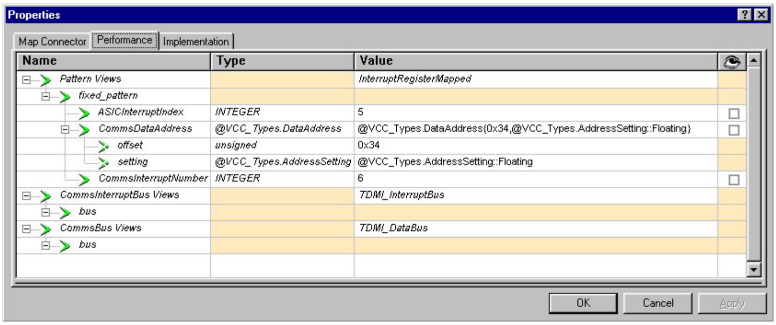




Mapping communication links to a pattern



Mapping communication links to a pattern



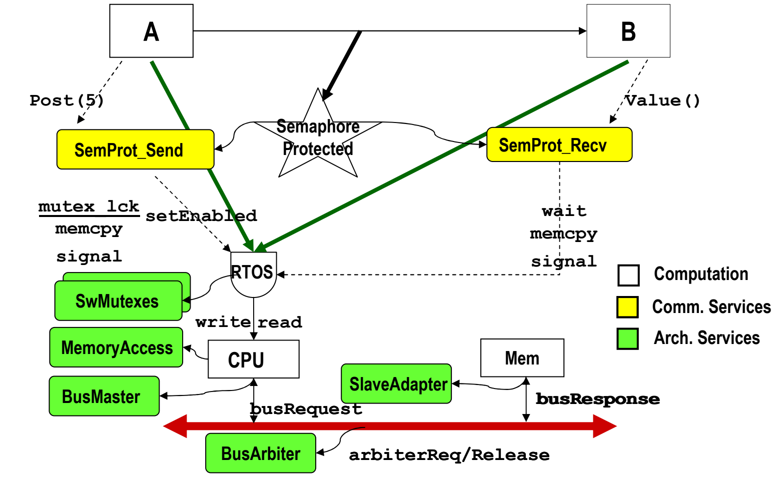
Properties

Map Connector | Performance | Implementation

Name	Type	Value	
Pattern Views		*InterruptRegisterMapped*	
fixed_pattern			
ASICInterruptIndex	*INTEGER*	5	☐
CommsDataAddress	*@VCC_Types.DataAddress*	@VCC_Types.DataAddress{0x34,@VCC_Types.AddressSetting::Floating}	☐
offset	*unsigned*	0x34	
setting	*@VCC_Types.AddressSetting*	@VCC_Types.AddressSetting::Floating	
CommsInterruptNumber	*INTEGER*	6	☐
CommsInterruptBus Views		*TDMI_InterruptBus*	
bus			
CommsBus Views		*TDMI_DataBus*	
bus			

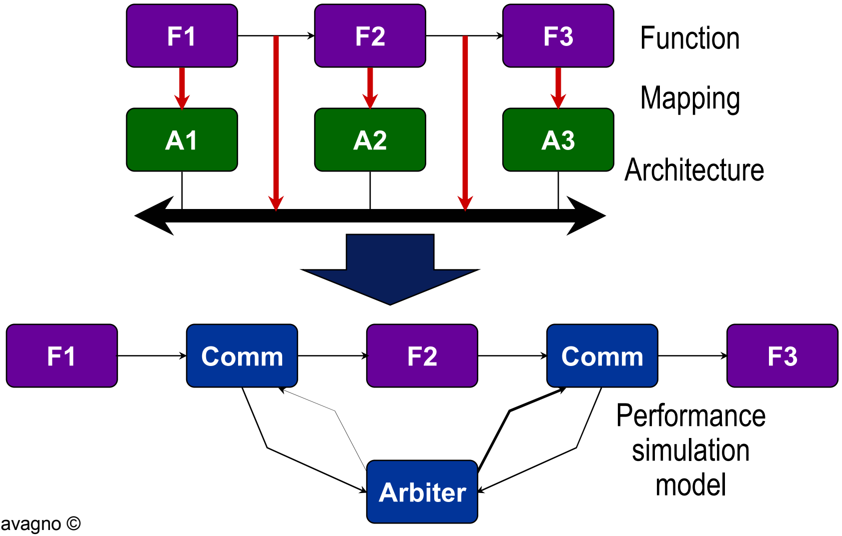
OK | Cancel | Apply

Communication Refinement



☐ Computation
🟨 Comm. Services
🟩 Arch. Services

Performance simulation by mapping



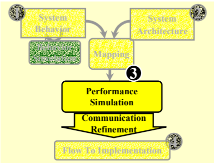
Function

Mapping

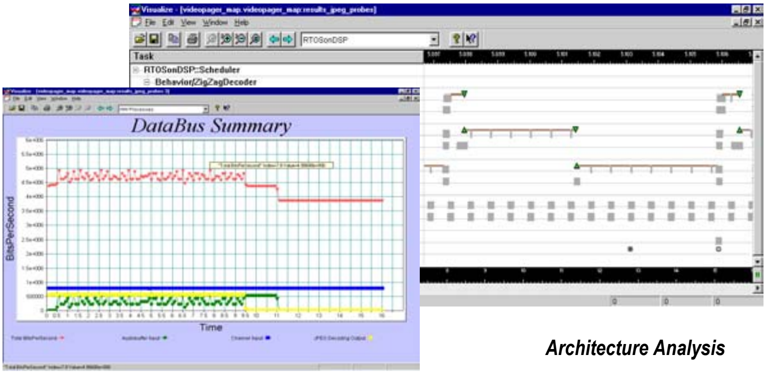
Architecture

Performance
simulation
model

Performance simulation



Software Gantt Charts


Visualize - [videopager_map.videopager_map:results_jpeg_probes]
File Edit View Window Help
RTOSonDSP
Task
RTOSonDSP::Scheduler
Behavior/ZigZagDecoder
DataBus Summary
BitsPerSecond
Time

Architecture Analysis

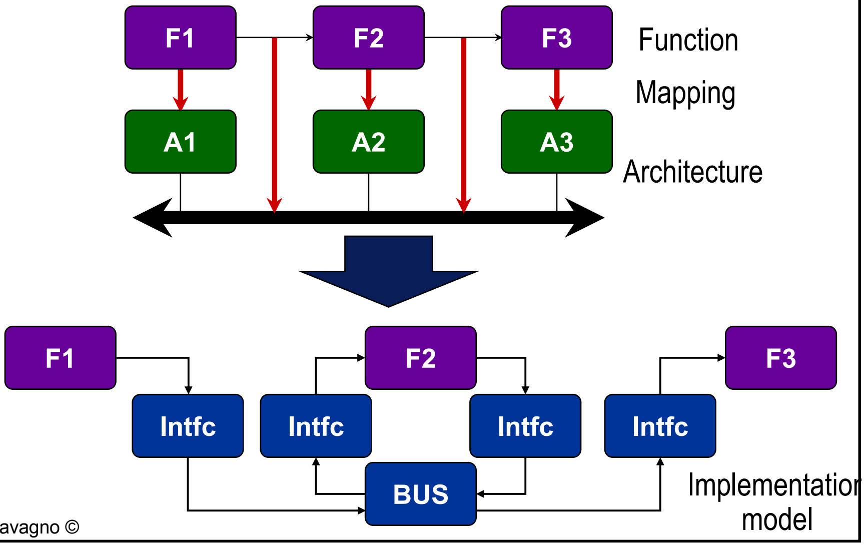
Exploring Design Trade Offs

- Iteration through different mapping experiments
- Gradual refinement of the design
- Evaluation
  - of the "refined" design
  - of system performance after implementation
- Export implementation to
  - Testbench and top-level netlist
  - Hardware netlist
  - Software RTOS customization

Outline

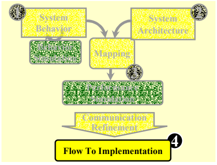
- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration

Implementation by mapping

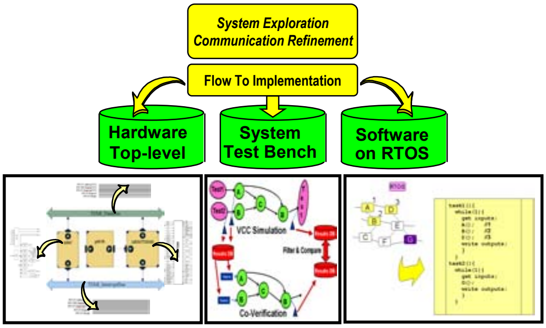


Luciano Lavagno ©

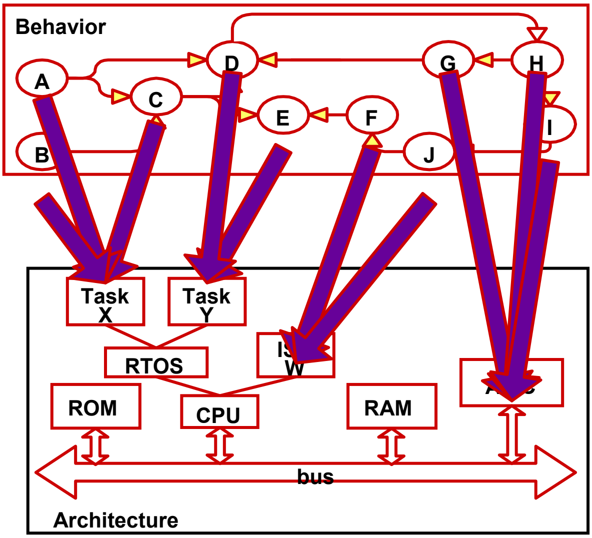
Flow to Implementation



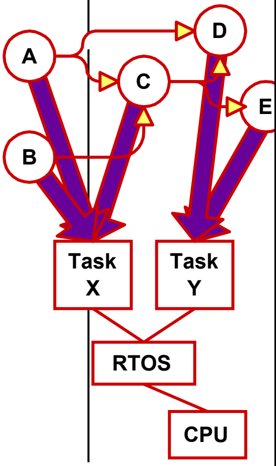
- Export refined design to co-verification and implementation tools



Flow to Implementation



Customizing RTOS

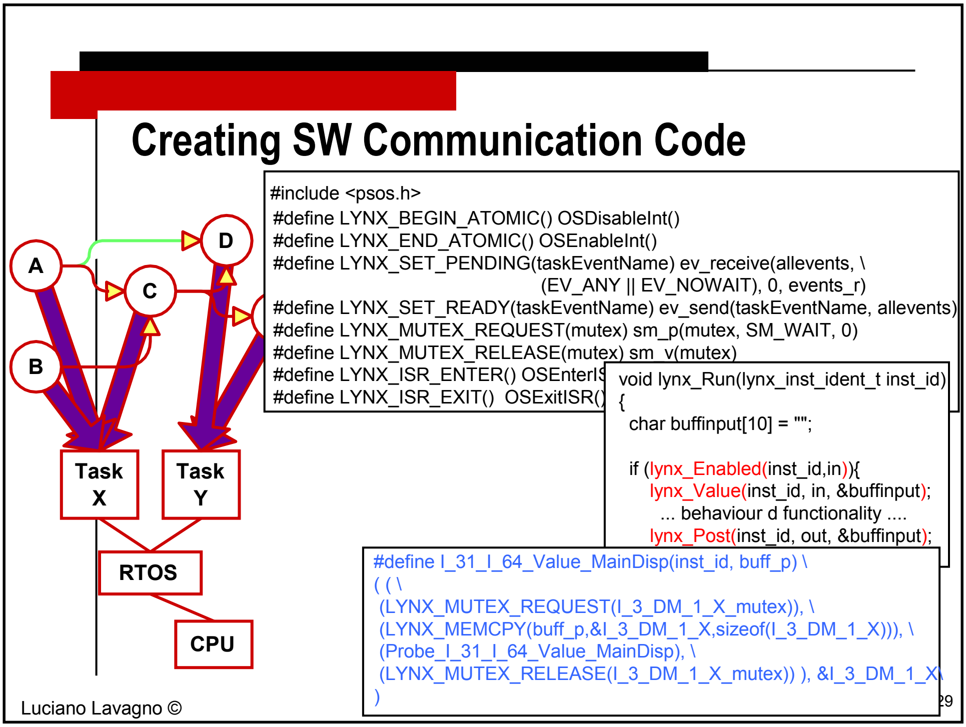


```
$<StandardHeader,'RTOS r
$<RtosAndCpuIncludes>
$<BoardSupportPackageInc
$<LynxSwIncludes>
/* Device Driver includes/de
$<LynxDriverIncludes>
$<LynxDeviceHandleDecs>
/* Mutex semaphore per pro
$<MutexVariableDefinitions
/* Define an identifier for eac
$<TaskIdDefinitions>

void root(void ) {
/* Mutex semaphore per pro
    $<CreateMutexes>
/* Create each software task
    $<CreateTasks>
/* Register interrupt service
    $<RegisterInterrupts>
/* Schedule each software ta
    $<StartTasks>
/* Delete or suspend the roo
    $<DeleteSelf>
}
```

```
#include <psos.h>
#include "init.h"
#include "tasks.h"
/* Device Driver includes/device handle decls  */
#include "drivers.h"
/* Mutex semaphore per protected data-buffer */
unsigned long l_24_l_50_MainDisp_mutex;
unsigned long l_24_l_50_SubDisp_mutex;
/* Define an identifier for each task              */
unsigned long task_l_13_l_6__ready;
unsigned long task_l_26__ready;

void root(void ) {
/* Mutex semaphore per protected data-buffer */
k_fatal(0x20000004, K_LOCAL);
k_fatal(0x20000004, K_LOCAL);
/* Create each software task                              */
    if (t_create("T0", 10, 1024, 1024, T_LOCAL|,
&task_l_13_l_6__ready)) k_fatal(0x20000001,
    if (t_create("T1", 11, 1024, 1024, T_LOCAL|,
&task_l_26__ready)) k_fatal(0x20000001,
…
```



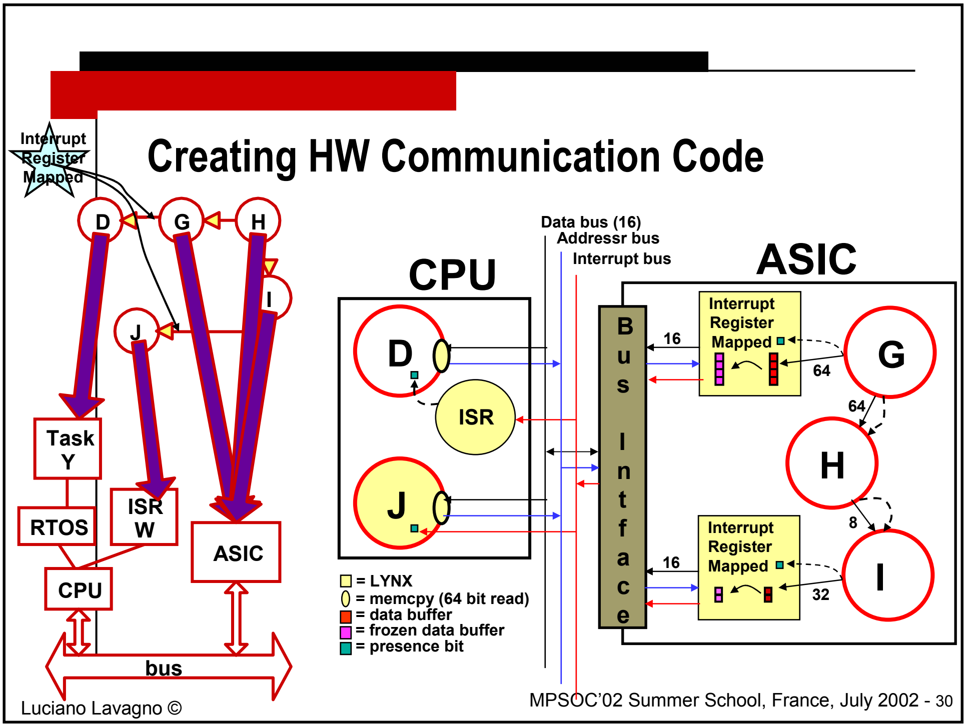
Creating SW Communication Code

```
#include <psos.h>
 #define LYNX_BEGIN_ATOMIC() OSDisableInt()
 #define LYNX_END_ATOMIC() OSEnableInt()
 #define LYNX_SET_PENDING(taskEventName) ev_receive(allevents, \
                                    (EV_ANY || EV_NOWAIT), 0, events_r)
 #define LYNX_SET_READY(taskEventName) ev_send(taskEventName, allevents)
 #define LYNX_MUTEX_REQUEST(mutex) sm_p(mutex, SM_WAIT, 0)
 #define LYNX_MUTEX_RELEASE(mutex) sm_v(mutex)
 #define LYNX_ISR_ENTER() OSEnterIS
 #define LYNX_ISR_EXIT()  OSExitISR()
```

```
void lynx_Run(lynx_inst_ident_t inst_id)
{
  char buffinput[10] = "";

  if (lynx_Enabled(inst_id,in)){
     lynx_Value(inst_id, in, &buffinput);
       ... behaviour d functionality ....
     lynx_Post(inst_id, out, &buffinput);
```

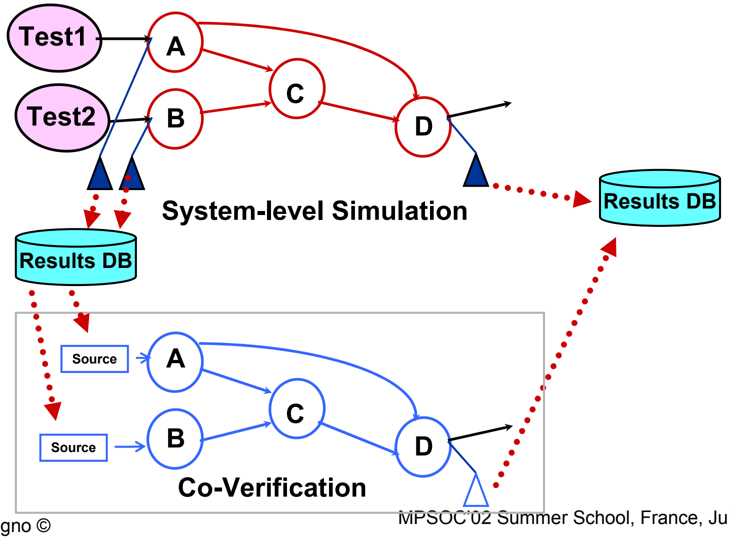
```
#define I_31_I_64_Value_MainDisp(inst_id, buff_p) \
( ( \
 (LYNX_MUTEX_REQUEST(I_3_DM_1_X_mutex)), \
 (LYNX_MEMCPY(buff_p,&I_3_DM_1_X,sizeof(I_3_DM_1_X))), \
 (Probe_I_31_I_64_Value_MainDisp), \
 (LYNX_MUTEX_RELEASE(I_3_DM_1_X_mutex)) ), &I_3_DM_1_X\
)
```



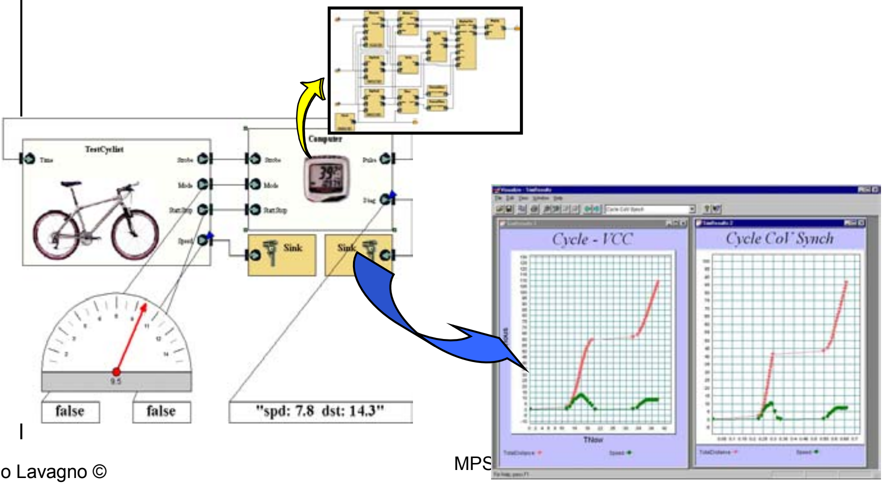
Creating HW Communication Code

□ = LYNX
○ = memcpy (64 bit read)
🟥 = data buffer
🟪 = frozen data buffer
🟩 = presence bit

Creating Testbench



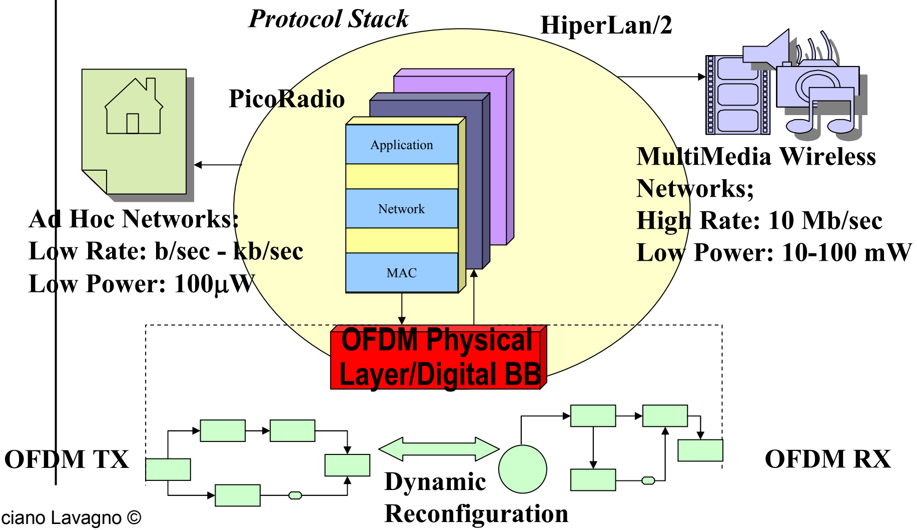
Comparing Results



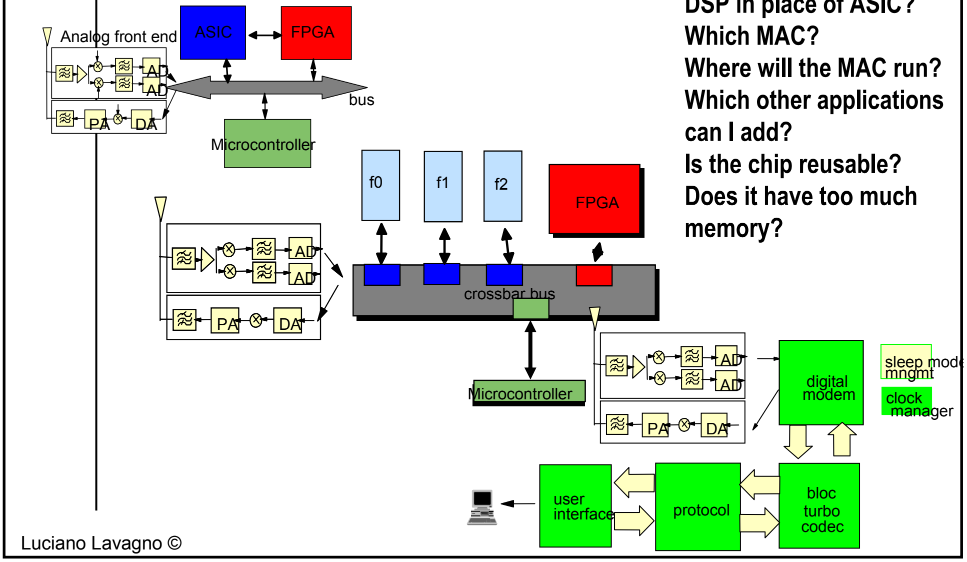
Outline

- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration

Case study: wireless LAN physical layer



From board to SOC



Analog front end

ASIC

FPGA

bus

Microcontroller

AD

AD

PA

DA

f0

f1

f2

FPGA

crossbar bus

Microcontroller

digital modem

sleep mode mngmt

clock manager

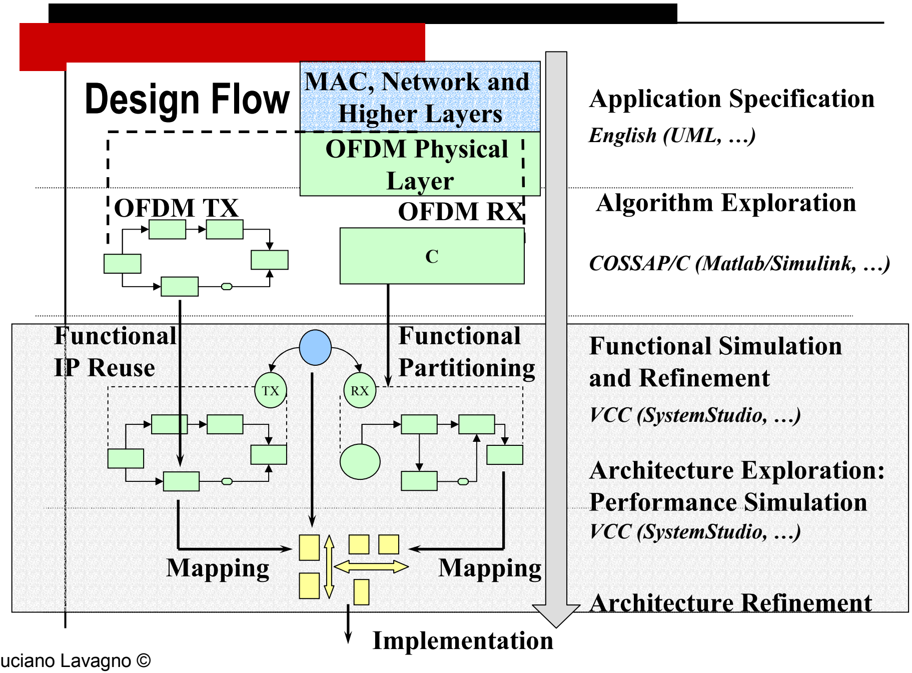
bloc turbo codec

protocol

user interface

Which microcontroller?
Do I need more FPGA?
DSP in place of ASIC?
Which MAC?
Where will the MAC run?
Which other applications can I add?
Is the chip reusable?
Does it have too much memory?

Design Flow



MAC, Network and Higher Layers

OFDM Physical Layer

OFDM TX

OFDM RX

C

Functional IP Reuse

Functional Partitioning

TX

RX

Mapping

Mapping

Implementation

Application Specification

English (UML, …)

Algorithm Exploration

COSSAP/C (Matlab/Simulink, …)

Functional Simulation and Refinement

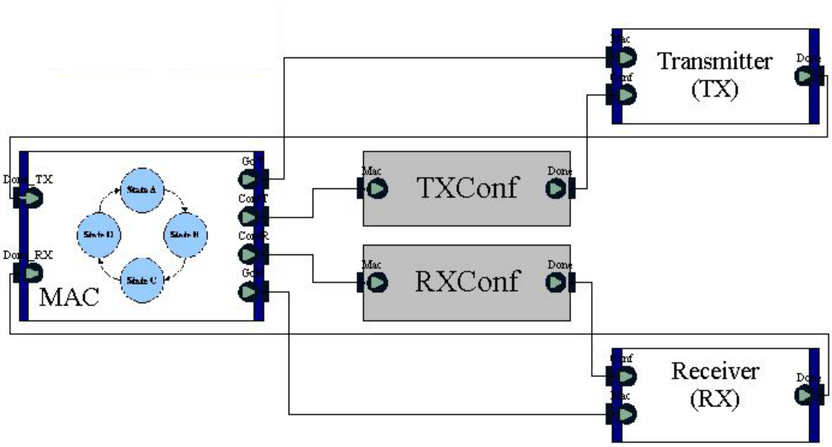
VCC (SystemStudio, …)

Architecture Exploration: Performance Simulation

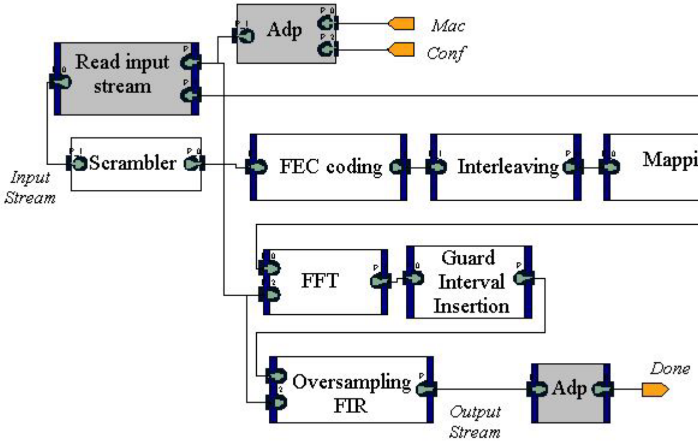
VCC (SystemStudio, …)

Architecture Refinement

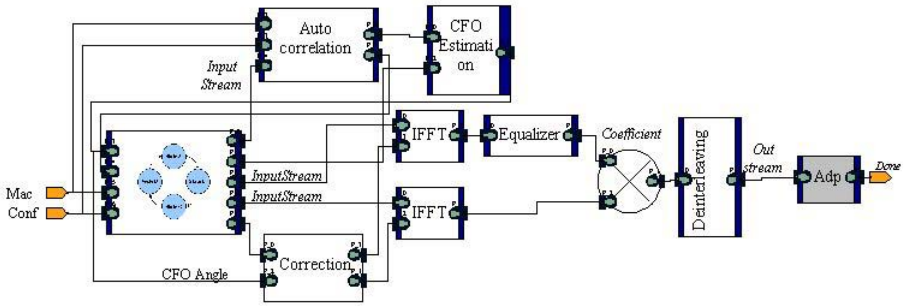
Top-level Hiperlan/2 Functional Model



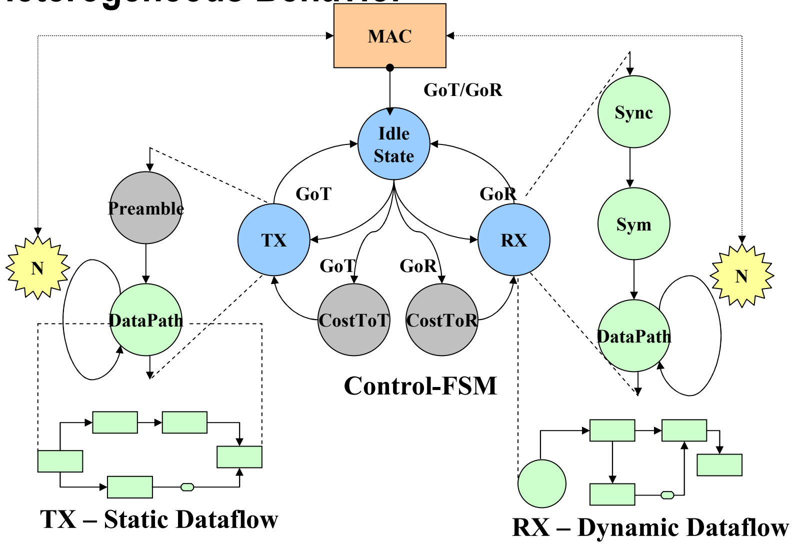
Hiperlan/2 OFDM Transmitter



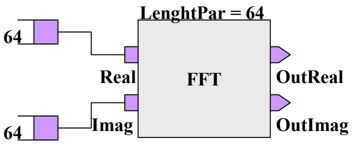
Hiperlan/2 OFDM Receiver



Heterogeneous Behavior



Example of functional block



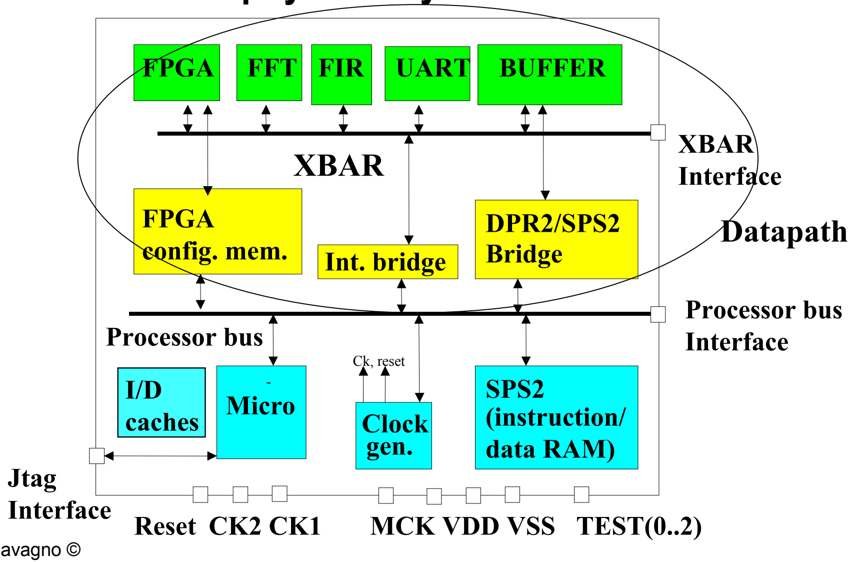
```
void CPP_MODEL_IMPLEMENTATION::Init()
{
  ….; Length = LenghtPar.Value(); // read parameter
 // Set data rate on 2 input ports: Real and Imag
 Real.SetDataRate(Length);
 Imag.SetDataRate(Length);
}
// Run() is executed every time the firing rule is satisfied
void CPP_MODEL_IMPLEMENTATION::Run()
{
   for (i=0; i<Real.GetDataRate(); i++) {
              // Read data from the input ports
              data[i] [0] = Real.Value();
              data[i] [1] = Imag.Value();
  }
  // Call the FFT procedure (C functional model)
  fft_cns_rot_bfp(data,….);
  // Write data to two output ports (OutReal, OutImag)
  for( i=0; i< Lenght; i++) {
              OutReal.Post(data[i][0]);
              OutImag.Post(data[i][1]);}}
 }
}
```

Imported from Cossap environment →

Outline

- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration

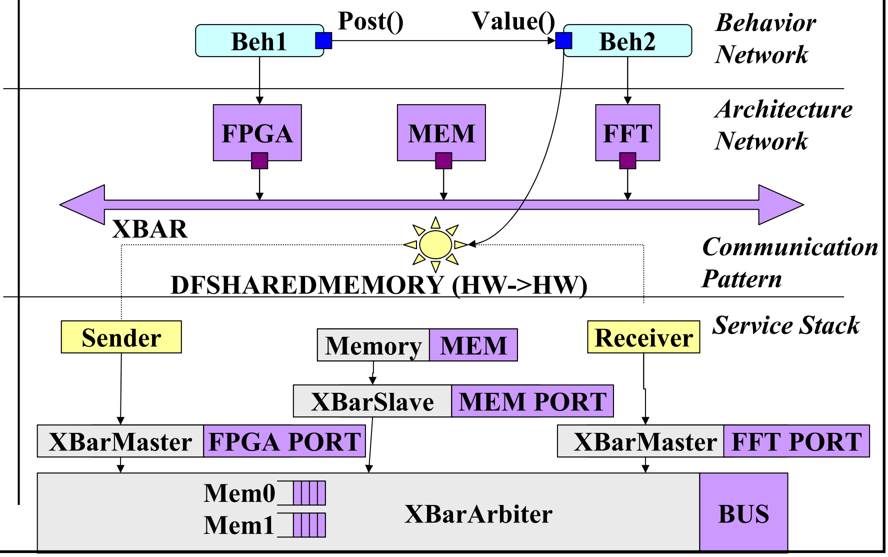
Wireless LAN physical layer SOC architecture



Crossbar features

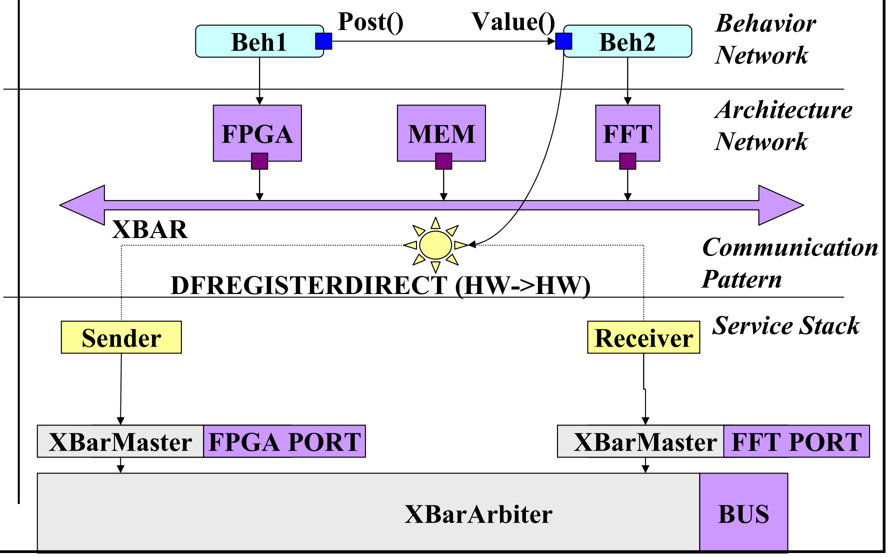
- The crossbar model is flexible in the number of masters and slaves supported (evaluated at simulation initialization time)
- A prioritized FIFO is used to arbitrate multiple master requests for each slave
- Number of parallel slave accesses defined through a parameter
- A transmission can be suspended by higher priority requests (preemptive)
- Arbitration overhead and slave access delays are parameterized

Crossbar Architecture Service Structure



Post()
Value()
Beh1
Beh2
Behavior Network
FPGA
MEM
FFT
Architecture Network
XBAR
DFSHAREDMEMORY (HW->HW)
Communication Pattern
Sender
Memory | MEM
Receiver
Service Stack
XBarSlave | MEM PORT
XBarMaster | FPGA PORT
XBarMaster | FFT PORT
Mem0
Mem1
XBarArbiter
BUS

Crossbar Architecture Service Structure

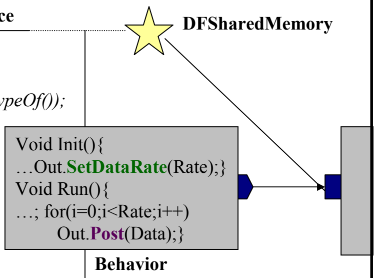


Post()
Value()
Beh1
Beh2
Behavior Network
FPGA
MEM
FFT
Architecture Network
XBAR
DFREGISTERDIRECT (HW->HW)
Communication Pattern
Sender
Receiver
Service Stack
XBarMaster | FPGA PORT
XBarMaster | FFT PORT
XBarArbiter
BUS

Sender Service Communication Refinement

Pattern Sender Service

```
void Init() {....
// Evaluate the buffer size.
   bitSize_ = typeInfo.bitsPerByte() * typeInfo.getTypeSize(&TypeOf());
// Read parameter Rate
void SetDataRate(int rate){dataRate_ = rate; }
// Redefine Post
void Post_(const typeObject& a, double delay) {....
  numPosted_++;
  if (numPosted_== dataRate_) {
    numPosted_=0;
// Generate a transaction record
   trans_.setTransactionType(vccTransWriteNoStore);
   trans_.setTransferSize(bitSize_);
   trans_.setTargetAddress(dataAddr_);
// Submit the buffer transaction to the bus
   busAdapter.busRequest(&trans_); ...}
```



```
Void Init(){
...Out.SetDataRate(Rate);}
Void Run(){
...; for(i=0;i<Rate;i++)
      Out.Post(Data);}
```

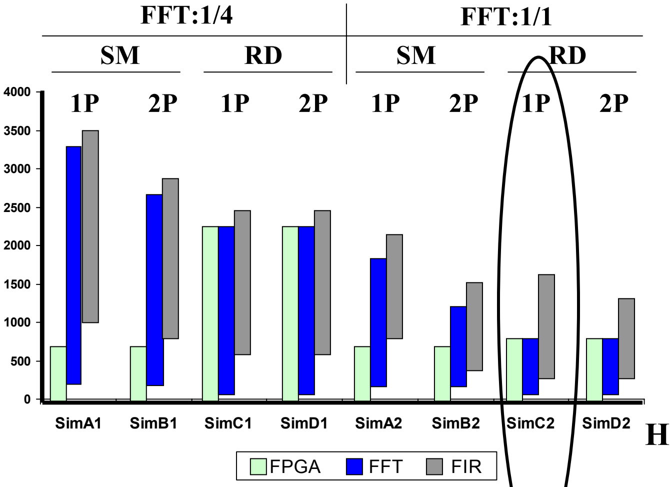
Outline

- System-on-chip design flow
  - Functional and architectural modeling
  - Mapping
  - Performance simulation
  - Communication refinement
  - Implementation
- Case study: wireless LAN architectural exploration
  - functional model
  - on-chip communication architectural model
  - design space exploration

Design Space Exploration

- Explored several computation/communication architecture configurations
  - FFT throughput (¼ clock cycle vs 1 clock cycle)
  - Number of buffer ports FFT → FIR on crossbar
  - FPGA → FFT communication pattern
    - Shared Memory
    - Register Direct

Exploration Results



									Hiperlan/2 spec.
BitRate (Mb/s)	5.8	7.2	8.2	8.2	9.6	13.7	12.5	15.6	12

Conclusion

- System-On-Chip Design requires methodology, tools and libraries
- Separate computation, communication and architecture
  - computation: compiled and scheduled
  - communication: refined via patters
- Map computation and communication onto platform
  - simulate performance
  - generate implementation model for HW, SW and communication