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. Annotation of architectural timing and energy onto behavior
2. Performance Simulation behavior annotated with architectural effects
3. Analyze / Visualize Results

Functional Modeling

MPEG Decoder

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;
    __DelayCycles(2);
    y.write(r);
}

Separate Delay Model

Inline Delay Model

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

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

Annotated

Software Performance Estimation

Compile generated C and run natively

Specify behavior and I/O

Generate new C with delay annotations

Virtual Machine Instructions

Analyse basic blocks, compute delays

Performance Estimation

Architecture Characterization

ANSI C Input

Analyse basic blocks, compute delays

Generate new C with delay annotations

Specify behavior and I/O

Compile generated C and run natively

Performance Estimation
Communication refinement

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

Process

HW/SW Independent System
Communications e.g. Bounded FIFO

Module

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

Module Interface

Physical Bus Transfers
e.g. Arbitrated PBus protocol

VCI to Physical-Bus Wrapper
Communication refinement

Mapping communication links to a pattern
Mapping communication links to a pattern

Communication Refinement

A

Semaphore Protected

SemProt_Send

mutex_lock

setEnabled

signal

SwMutexes

MemoryAccess

BusMaster

CPU

SlaveAdapter

Mem

BusArbiter

busRequest

busResponse

SemProt_Recv

wait

memcpy

signal

SemProt_Send

SemProt_Recv

B

Value()

Post(5)

ETOS

Computation

Comm. Services

Arch. Services
Performance simulation by mapping

Performance simulation model

Software Gantt Charts

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

Flow to Implementation

- Export refined design to co-verification and implementation tools
**Flow to Implementation**

- **Behavior**
  - A
  - B
  - C
  - D
  - E
  - F
  - G
  - H
  - I
  - J

- **Architecture**
  - Task X
  - Task Y
  - RTOS
  - CPU
  - RAM
  - ROM
  - bus

---

**Customizing RTOS**

- `$<StandardHeader,RTOS root>`
- `$<RtosAndCpuIncludes>`
- `$<BoardSupportPackageIncludes>`
- `$<LynxSwIncludes>`
- `/* Device Driver includes/device handle decls */`
- `/* Mutex semaphore per protected data-buffer */`
- `/* Define an identifier for each task */`
- `void root(void) { /* Mutex semaphore per protected data-buffer */`
- `/* Create each software task */`
- `/* Register interrupt service */`
- `/* Schedule each software task */`
- `/* Delete or suspend the root */`

```c
#include <psos.h>
#include "init.h"
#include "tasks.h"
/* Device Driver includes/device handle decls */
/* Mutex semaphore per protected data-buffer */
unsigned long I_24_I_50_MainDisp_mutex;
unsigned long I_24_I_50_SubDisp_mutex;
/* Define an identifier for each task */
unsigned long task_I_13_I_6__ready;
unsigned long task_I_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_I_13_I_6__ready)) k_fatal(0x20000001,
  if (t_create("T1", 11, 1024, 1024, T_LOCAL|&task_I_26__ready)) k_fatal(0x20000001,
  ...
```
Creating SW Communication Code

```c
#include <psos.h>
#define LYNX_BEGIN_ATOMIC() OSDisableInt()
#define LYNX_END_ATOMIC() OSEnableInt()
#define LYNX_SET_PENDING(taskEventName) ev_receive(allevents, \n    (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() OSEnterISR()
#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 and functionality ...
        lynx_Post(inst_id, out, &buffinput);
    }
    #define I_31_I_64_Value_MainDisp(inst_id, buff_p)
    (LYNX_MUTEX_REQUEST(I_31_DM_1_X_mutex)),
    (LYNX_MEMCPY(buff_p, &I_31_DM_1_X, sizeof(I_31_DM_1_X)),
    (Probe_I_31_I_64_Value_MainDisp),
    (LYNX_MUTEX_RELEASE(I_31_DM_1_X_mutex))
    void lynx_Run(lynx_inst_ident_t inst_id)
    {
        char buffinput[10] = "";
        if (lynx_Enabled(inst_id, in))
        {
            lynx_Value(inst_id, in, &buffinput);
        }
        #define I_31_I_64_Value_MainDisp(inst_id, buff_p)
        (LYNX_MUTEX_REQUEST(I_31_DM_1_X_mutex)),
        (LYNX_MEMCPY(buff_p, &I_31_DM_1_X, sizeof(I_31_DM_1_X)),
        (Probe_I_31_I_64_Value_MainDisp),
        (LYNX_MUTEX_RELEASE(I_31_DM_1_X_mutex))
    }
}
```

Creating HW Communication Code

```c
#define LYNX_MUTEX_REQUEST(mutex) sm_p(mutex, SM_WAIT, 0)
#define LYNX_MUTEX_RELEASE(mutex) sm_v(mutex)
#define LYNX_ISR_ENTER() OSEnterISR()
#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 and functionality ...
        lynx_Post(inst_id, out, &buffinput);
    }
    #define I_31_I_64_Value_MainDisp(inst_id, buff_p)
    (LYNX_MUTEX_REQUEST(I_31_DM_1_X_mutex)),
    (LYNX_MEMCPY(buff_p, &I_31_DM_1_X, sizeof(I_31_DM_1_X)),
    (Probe_I_31_I_64_Value_MainDisp),
    (LYNX_MUTEX_RELEASE(I_31_DM_1_X_mutex))
    void lynx_Run(lynx_inst_ident_t inst_id)
    {
        char buffinput[10] = "";
        if (lynx_Enabled(inst_id, in))
        {
            lynx_Value(inst_id, in, &buffinput);
        }
        #define I_31_I_64_Value_MainDisp(inst_id, buff_p)
        (LYNX_MUTEX_REQUEST(I_31_DM_1_X_mutex)),
        (LYNX_MEMCPY(buff_p, &I_31_DM_1_X, sizeof(I_31_DM_1_X)),
        (Probe_I_31_I_64_Value_MainDisp),
        (LYNX_MUTEX_RELEASE(I_31_DM_1_X_mutex))
    }
```

Creating Testbench

- Test1
- Test2

System-level Simulation

Results DB

Comparing Results

Cycle / TCC
Cycle Cycles Switch
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

- Protocol Stack
- HiperLan/2
- MultiMedia Wireless Networks;
  - High Rate: 10 Mb/sec
  - Low Power: 10-100 mW

Ad Hoc Networks:
- Low Rate: b/sec - kb/sec
- Low Power: 100μW

PicoRadio

OFDM TX

OFDM RX

Dynamic Reconfiguration

OFDM Physical Layer/Digital BB
From board to SOC

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

Application Specification
English (UML, …)

Algorithm Exploration
COSSAP/C (Matlab/Simulink, …)

Functional Simulation and Refinement
VCC (SystemStudio, …)

Architecture Exploration:
Performance Simulation
VCC (SystemStudio, …)

Architecture Refinement

Functional IP Reuse

Mapping
Top-level Hiperlan/2 Functional Model

Hiperlan/2 OFDM Transmitter
Hiperlan/2 OFDM Receiver

Heterogeneous Behavior

RX – Dynamic Dataflow

TX – Static Dataflow

MAC

Idle State

GoT/GoR

Sync

Sym

DataPath

TX

RX

Control-FSM

GoT

GoT

GoR

GoR

Idle

State

Preamble

N

N
Example of functional block

```
void CPP_MODEL_IMPLEMENTATION::Init()
{
    Length = LengthPar.Value(); // read parameter
    Real.SetDataRate(Length);
    Imag.SetDataRate(Length);
}

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<Length; i++) {
        OutReal.Post(data[i][0]);
        OutImag.Post(data[i][1]);
    }
}
```

Imported from Cossap environment

```
Example of functional block

void CPP_MODEL_IMPLEMENTATION::Init()
{
    Length = LengthPar.Value(); // read parameter
    Real.SetDataRate(Length);
    Imag.SetDataRate(Length);
}

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<Length; i++) {
        OutReal.Post(data[i][0]);
        OutImag.Post(data[i][1]);
    }
}
```

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

- **Behavior Network**
  - Beh1
  - Beh2

- **Architecture Network**
  - FPGA
  - MEM
  - FFT

- **Communication Pattern**
  - DFSHAREDMEMORY (HW->HW)
  - XBarArbiter
  - Sender
  - Receiver

- **Service Stack**
  - Mem0
  - Mem1
  - FPGA PORT
  - MEM PORT
  - FFT PORT
  - XBarMaster
  - XBarSlave
  - BUS

- **Network**
  - XBAR

Sender Service Communication Refinement

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_);...}

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 (1/4 clock cycle vs 1 clock cycle)
  - Number of buffer ports FFT $\rightarrow$ FIR on crossbar
  - FPGA $\rightarrow$ FFT communication pattern
    - Shared Memory
    - Register Direct

Exploration Results

<table>
<thead>
<tr>
<th>BitRate (Mb/s)</th>
<th>5.8</th>
<th>7.2</th>
<th>8.2</th>
<th>8.2</th>
<th>9.6</th>
<th>13.7</th>
<th>12.5</th>
<th>15.6</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiperlan/2 spec.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

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