Architecture and Communication-Based Design
The Design Methodology

- Functional IP
  - C/C++
  - SDL
  - SPW
  - Simulink

Platform Function

Platform Architecture

System Integration

Performance Analysis and Platform Configuration

Embedded System Requirements

Platform Configuration
  ... at the un-clocked, timing-aware system level

Architecture
  - IP
  - CPU/DSP
  - RTOS
  - Bus, Memory
  - HW
  - SW
VCC Front End
Define Architectural Options and Configuration

Embedded System Requirements

- Functional IP
- SDL
- C/C++
- SPW
- Simulink

Platform Function

Platform Architecture

Architecture IP
- CPU/DSP
- RTOS
- Bus, Memory
- HW
- SW

Platform Configuration
... at the un-clocked, timing-aware system level
VCC Front End
Define Function Architecture Mapping

Embedded System Requirements

Functional IP
C/C++ SDL SPW Simulation

Platform Function

Platform Architecture

System Integration

Architecture IP
CPU/DSP RTOS
Bus, Memory HW

Platform Configuration
... at the un-clocked, timing-aware system level

Design Exploration: DSP/AM/Demand Memory
Description: DSP/AM/Demand Memory
Mapping: VCC_GSM_Map_DSP/AM/Demand_Mapping
Analysis: VCC_GSM_Map_DSP/AM/Demand_Analysis

Design Exploration: Channel Decoder (Viterbi) in DSP
Description: DSP implementation of Viterbi Decoder
Mapping: VCC_GSM_Map_DSP_Exploration_map_mapping
Analysis: VCC_GSM_Map_DSP_Exploration_map_analysis

EE249Fall03
Design Space Exploration
From RTL through Gate Level options

Technology provider
– characterizes silicon technology for gates and interconnects

Synthesis Tools
– map constructs from RTL into registers and logic in between registers, does logic optimization
– explore the design space (“performance” – “area”) using gradient methods in a optimization process
Design Space Exploration
... through Function and Architecture
Design Space Exploration
... through Function and Architecture

**SOC Silicon provider**
- characterizes IP portfolio (typically in Integration Platforms) for intrinsic IP Block Performance and Inter IP Block Interconnect Performance

**System Integrator and SOC Provider**
- map function to architecture setting up design experiments
- determine using performance simulation feedback suitability of function-architecture combination
- explore design space through “function” and ”architecture”
Platform-based Design

Platform Mapping
Platform Design-Space Export

Application Space
Application Instance
System (Software + Hardware) Platform
Platform Instance
Architectural Space

ASV Triangles
A Hardware Platform is a family of architectures that satisfy a set of architectural constraints imposed to allow the re-use of hardware and software components.
And what is the next step?

**Modeling of Performance for IP Blocks**

- ... by attaching performance data to timing free functional models

- **Transistor Model**
  - Capacity Load

- **SDF**
  - Gate Level Model
  - Capacity Load

- **RTL**
  - Clusters

**IP Block Performance**

- **1970’s**
- **1980’s**
- **1990’s**
- **Year 2000 +**
And what is the next step?

**Modeling of Performance for Communication between IP Blocks**

- **Transistor Model Capacity Load**
- **SDF Gate Level Model Capacity Load**
- **RTL Clusters**
- **Inter IP Communication Performance**

- **1970’s**
- **1980’s**
- **1990’s**
- **Year 2000 +**
And what is the next step?

Apply this to Hardware and Software

Discontinuity: Embedded Software

IP Block Performance
Inter IP Communication Performance

1970’s
1980’s
1990’s
Year 2000 +
In general, a platform is an abstraction layer that covers a number of possible refinements into a lower level. The platform representation is a library of components including interconnects from which the lower level refinement can choose.
Principles of Platform methodology: Meet-in-the-Middle

• Top-Down:
  – Define a set of abstraction layers
  – From specifications at a given level, select a solution (controls, components) in terms of *components* (*Platforms*) of the following layer and propagate constraints

• Bottom-Up:
  – Platform components (e.g., micro-controller, RTOS, communication primitives) at a given level are abstracted to a higher level by their functionality and a set of parameters that help guiding the solution selection process. The selection process is equivalent to a covering problem if a common semantic domain is used.
Architecture Abstraction as a Service

• Architectures are used to implement functions
• Functions see architectures as services that offer functionality with parameters expressing non-functional properties (e.g., power, size, performance)
• A micro-processor is a programmable resource that supports any computable function but with limited performance
• A bus is a communication service with limited capacity
• The service is the element that defines the functionality of architecture

• A service performs a specific role to model architecture, for example:
  – bus arbitration
  – memory access
  – interrupt propagation
  – etc.
Example of Services

Behavior \rightarrow Post \rightarrow Pattern Sender \rightarrow ASIC \rightarrow BusMaster \rightarrow Bus \rightarrow BusArbiter

ASIC \rightarrow BusSlave \rightarrow Memory
Example of Services

• Behavior calls Post, i.e., send a communication
• Pattern hears Post and directs ASIC block’s BusMaster to send a communication
• BusMaster asks the Bus Block’s BusArbiter for use of the bus
• BusArbiter grants the bus, so communication can go to Memory Block
• Memory Block’s BusSlave receives communication and forwards to memory
• Memory stores communication.
Categories of Services

• Pattern Service
  – services that coordinate the communication of architecture services

• Architecture Service
  – services that define the functionality of architecture
Pattern Service

- A pattern coordinates architectural services that collectively model a communication path from sender to receiver

- Patterns are composed of a sender service and a receiver service
  - Sender service defines Post
  - Receiver service defines Enabled/Value

- Both the sender and receiver service direct the actions of architecture services to send/receive communication
Basic Example

• Let’s assume two behaviors.
• b1 and b2 talk to each other:
  – b1 says Post; b2 says Value
  – and vice versa
Basic Example (cont)

• What does it mean for b1 to talk to b2?
• What does it mean for b1 to say Post?
• What does it mean for b2 to say Value?
• We should consider an architecture to give meaning to b1 and b2.
• We should consider how the behavior blocks map to the architecture.
Basic Example (cont)

- Let’s assume the following architecture:

![Diagram with nodes labeled Simple Memory, ASIC, Co, ARM/TDMI, and connections labeled TDMI_DataBus and TDMI_InterruptBus.]

Architecture Parameters
DataBusClock: 33000000.0
Basic Example (cont)

• Here we map the behavior to the architecture:
• What do we see in the mapping diagram?
  – b1 is mapped to software.
  – b2 is mapped to hardware.
  – b1 to b2 communication is set to Shared Memory.
  – b2 to b1 communication is set to Interrupt Register Mapped.
• For simplicity’s sake, we’re focusing on b1-to-b2 communication.
  – b2 to b1 will be ignored for now.
• If b1 talks to b2, how does that look when mapped to an architecture?
  – What happens when b1 says Post?
  – What happens when b2 says Value?
  – Note b1 to b2 is shared memory communication.
Using Shared Memory, we have the following sequence of communication:

1. b1 writes to memory:
   b1 $\rightarrow$ RTOS $\rightarrow$ CPU $\rightarrow$ Bus $\rightarrow$ Mem
2. b2 reads from memory:
   b2 $\rightarrow$ ASIC $\rightarrow$ Bus $\rightarrow$ Mem
Basic Example (cont)

• So b1 talks to b2 through the various architecture components:
  – b1 says Post and that becomes a write to memory.
  – b2 says Value and that becomes a read from memory.
• What is the underlying mechanism that propagates Post/Value through the architecture?
  – It’s something called the “service”.
Architectural Services

• More basic than the architecture block is the service.
• The service is the atomic (unsplittable) piece that composes the architecture block.
• The next diagrams overlay the services on top of architectural blocks.
Service Example

Architecture

SimpleMemory
- Memory
- FCFSSlaveAdapter

TDMI_DataBus
- BusArbiter

ASIC
- FCFSSBusAdapter

eCos
- StandardCLibrary

SimpleCPU
- CPUMemoryAccess
- FCFSSBusAdapter
Service Example (cont)

Architecture

Pattern

b1

Sender

Receiver

b2
Communication-Based Design
Picoradio Sensor Networks (BWRC)

- Key challenges
  - Satisfy tight performance and cost constraints (especially power consumption)
  - Identify Layers of Abstraction (Protocol Stack)
  - Develop distributed algorithms (e.g. locationing, routing) for ubiquitous computing applications
  - Design Embedded System Platform to implement Protocol Stack efficiently

- Control Environmental parameters (temperature, humidity...)
- Minimize Power consumption
- Cheap (<0.5$) and small (< 1 cm³)
- Large numbers of nodes — between 0.05 and 1 nodes/m²
- Limited operation range of network — maximum 50-100 m
- Low data rates per node — 1-10 bits/sec average
- Low mobility (at least 90% of the nodes stationary)
Network Design Using Platforms

Application components
Requirements

Components Adaptation

Communication Refinement
(Protocol Stack + Gateways)

Embedded System Platform

On-Chip Networks
Network Platforms

- Network Platform (NP): Library of communication resources (protocols, channels…)
- Network Platform Instance (NPI): selection of NP resources
  - Structure: set of resources and topology
  - Behavior: set of event traces (events model send or receive actions)
Communication Refinement

• Replace components in abstract NPI with set of resources or NPIs
• Select resources from NP library and compose them to create NP with new Communication Services
• NP resources
  – Channels (C): transfer messages
  – Protocols (P): adapt Communication Services
  – Gateways (G): interface NPs
• Check refined NPI satisfies constraints
Sensor Network Applications

- **Application** - collection of sensors, controllers and actuators cooperating to achieve a common goal (environment control or monitoring)
- **Sensor** - measures the state of the environment
  - Parameters: phy. quantity, range, accuracy, ID, location...
- **Actuator** - sets the state of the environment
- **Controller** - gets the state of sensors and decides whether and how to set the state of actuators
Network Platforms

NP components:
- node
- link
- port
- NPI I/O port

Communication Services:
- CS1: Lossy Broadcast
  Error rate: 33%
  Max Delay: 30 ms
- CS2:
  …

Platform Specification

Platform Design-Space Parameters

System Platform

Application Space

Application Instance

Architectural Space

Platform Instance

Network Platform Instance

Network Platform API

Performance Estimates

Constraints Budgeting

Platform Specification

Platform Design-Space Parameters

System Platform

Application Space

Application Instance

Architectural Space

Platform Instance

Network Platform Instance

Network Platform API

Performance Estimates

Constraints Budgeting
Sensor Network Platform

- **Node**
  - computation and communication platform (memory, processors…)
  - sensor/actuator devices

- **Parameters:**
  - memory available,
  - energy level,
  - computation/communication cost…

- **Sensors, Controllers and Actuators mapped onto nodes**
• **Attribute-based Naming**
  – address groups of nodes using common attributes (“temperature & kitchen”), rather than individual nodes using their IDs
  – names defined by attribute (“temperature”) - selector (“< 30º”) pairs and logic predicates (“t < 30º OR p > 16”)

• **Query** - a controller gets the state of a group of components (sensors)
  – Parameters: QueryID, classes, frequency of responses, reliability..

• **Command** - a controller sets the state of a group of components (actuators)
Services

- **Concept Repository (CSR)**
  - maintains a repository with the definitions of the concepts uniquely defined within the network
  - defines scope of network
  - key for interoperation of networks
- **Time Synchronization (TSS)**
  - used by system components to share a common notion of time and agree on the ordering of the events
- **Location (LS)**
  - collects and provides information on the spatial position of the nodes in the network
Motivation

• System-level verification of large component-oriented designs will be very costly.

• We cannot afford to debug interface mismatches between internal components

• . . . especially considering that there will be many, many interfaces between so many components.

• Current situation is unacceptable
  – Interfaces are not specified precisely
  – No clear specification formalism exist.
  – Tools to create, debug, and make maximal use of the specifications don’t exist.
Basic Goals

- Identify precisely and formally the concept of communication, its level of abstraction and of the corresponding models of computation.
- Determine a set of properties that characterizes each level of abstraction.
- Provide methods and tools to extract formal properties and specifications from informal ones and existing, ambiguously specified standards.
Why separating computation and communication?

• Verification (debugging). If not:
  – Communication hard-wired with computation
  – Often hard to tell who is at fault
  – Bugs may be distributed, difficult to track down
  – Changes in the system may require rewriting of entire blocks, often leading to new bugs

• Reuse
  – Component may be plugged in different environments
  – Functions and interface behavior are difficult to separate

• Architecture exploration
  – Design components with abstract communication primitives
  – Explore different implementations without touching the component
Orthogonalizing Communication from Behavior

• Communication
  – hard to separate from behavior, usually intertwined
  – telecomm protocols are the best existing example

• Need to understand Formalism, Abstraction, and Refinement for communication
Formalism, Abstraction, and Refinement

• Formalism for Communication
  – Precise semantics for complex transactions
  – Multi-way communication, arbitration, addressing
  – Distributed control

• Abstraction and Refinement
  – Complex transaction mapped onto more primitive transactions
  – Elements of transaction refined onto concrete resources (pins, times)
Communication Refinement

- abstract from implementation concerns:
  - multi-target VC,
  - bus-independent VC

system transaction, «ANY» data structure (e.g. video line)

hardware or software

«ANY BUS» operation (data, address...)
VSI-Alliance OCB Group. Virtual Component Interface (VCI)

Physical Bus (e.g. PIBus) fixed bus-width, detailed protocol

Communication Interface (e.g. bounded FIFO)

Bus Wrapper
Abstraction

• Performance Model Abstraction Levels
  – Budget number, constant
  – Stochastic model with variation
  – Estimation based on approximate use of lower level abstraction
  – Actual simulation using lower level abstraction
Elements of Refinement

• Time vs Space
  – fewer resources means spreading out over time
  – extra handshaking means spreading out over space

• Arbitration
  – sharing resource between independent communication paths
  – data dependent arbitration

• Uneven source/sink speeds may require buffering

• Access to & storage for buffer memories may be shared
  – address computation
  – arbitration again
MPEG Algorithm

Audio In → Audio Decode

Video In → Video Decode

Video Decode → Frame buffer

Frame buffer → Video out

Video out → Onscreen Display

Host I/F → Audio Decode

Audio Decode → Host I/F
MPEG Architecture

- Audio In
- Video In
- Registers
- Host I/F
- Audio Decode
- Video Decode
- Video out
- Onscreen Display
- MMU/AGU
- Memory (buffers + frame buffer

Shared bus
Description Method

Algorithm

- Audio In
- Video In
- Host I/F
- Frame buffer
- Onscreen Display
- Audio Decode
- Video Decode
- Video out

Map

Architecture

- Audio In
- Video In
- Registers
- Host I/F
- Audio Decode
- Video Decode
- Video out
- Onscreen Display
- MMU/AGU
- Memory
Synthesize communication pattern through architecture

Value

- **Choose** from comprehensive set of *communication pattern*

- Pattern for **HW-SW**, **SW-HW**, **HW-HW** and **SW-SW** communication available

- *move function between HW and SW* boundaries and re-synthesize the *communication interface*

- customize platform communication environment through JAVA scripts
What is Communication?

Os = Fs(is)

Is → os

Or = Fr(ir)

ir → or
What is Communication?

- Connection C enables the interaction between the behaviors S and R

$O_s = F_s(i_s)$

$O_r = F_r(i_r)$
What is Communication?
A Formal View, basis for Metro MM

\[ Os = Fs(is) \]

\[ Or' = Fr'(ir') \]

Ideal Connection
What is Communication?

\[ Os = Fs(is) \]
\[ Or' = Fr'(ir') \]

- \( S \) restricts the behavior of \( R \) to \( R' \)
Behavior Adaptation

Os = Fs(is)   Or = Fr(ir)

S

Ideal Connection

R

- R not defined for some output of S: behavior mismatch
Behavior Adaptation

- Behavior Adapter $Z$ maps outputs of $S$ into the domain of $R$

$Os = Fs(is)$
$Or = Fr(ir)$

$S \xrightarrow{Fs} Z = Z' \circ Z'' \xrightarrow{Fz} R$

$S \xrightarrow{os} Z = Z' \circ Z'' \xrightarrow{ir} R$

$S \xrightarrow{Fs} Z = Z' \circ Z'' \xrightarrow{Fz} R$

$S \xrightarrow{Fs} Z = Z' \circ Z'' \xrightarrow{Fz} R$
Behavior Adaptation

\[ Os' = Fs'(is') = Fs \circ Z'(is') \]
\[ Or' = Fr'(ir') = Fr \circ Z''(ir') \]

- Behavior Adapter encapsulates S and R
- S’ and R’ communicate successfully over an ideal connection
Physical Channels

• Invalid Channels may introduce mismatch due to their physical properties (noise, interference…)

• Valid Channels satisfy QoS requirements
  – QoS-equivalent to the ideal connection \( (F_s' \circ F_c \circ F_r' \sim F_s' \circ F_r') \)
Choose SC and CR such that C' is valid
- \( Fs' \circ Fsc \circ Fc \circ Fcr \circ Fr' \sim Fs' \circ Fr' \)

Channel Adapter may introduce behavior mismatch
- need a Behavior Adapter
FIFOs as Behavior Adapters

- FIFOs adapt the rates of S and R
- Unbounded FIFOs
  - ideal adapter
- Bounded FIFOs
  - to prevent overflow, restrict S using blocking write (Req/Ack)
Protocol Design

(Ideal) Connection

Behavior Adaptation

Physical Channel selection

Channel Adaptation

Behavior Adaptation
Protocol Design

(Ideal) Connection

Behavior Adaptation

Physical Channel selection

Channel Adaptation

Behavior Adaptation