Communication-Based Design
Motivation: System-on-a-Chip Design

- DSM technologies:
  - Migration of the system from board to chip
  - Complexity of chip design increases
  - “IP reuse”-based design methodology
Communication Design

- Determine a protocol so that no matter what the communication topology and the nature of the IP’s the functionality of the overall system is guaranteed (TCP/IP like)
- Given the IP set and the interconnections, automatically synthesize protocols and macro-shells
- Given the IP set and a set of time-varying interconnections, automatically synthesize adaptive protocol and macro-shells that optimize “performance” according to the current topology
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
Communication-based Design

Pearls (the IP Processes)
MicroShells (the IP Requirements)
MacroShells (the Protocol Interface)
Communication Channels
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.
The dream: formalize first!

- Formal specifications will be written, conformed to, and used.
- Standards will be precise and unambiguous
- Specifications will be used for
  - Simulation
  - Formal verification
  - Semiformal verification
  - Synthesis

Same specification will be used for all these purposes
Basic Goals

- Identify precisely and formally the concept of communication, its level of abstraction and of the corresponding models of computation.
- New theories to combine different 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.
Basic Goals

- Formal synthesis of protocols
- Bus architectures analysis and verification
  - formal specs
  - monitors
  - semi-formal analysis
- Abstract the notion of communication architectures and include estimation processes to quickly evaluate different architecture
- Protocol design for the Pico-radio and the Universal Spectrum Radio
- Formalize the specs of PCI and other busses and propose new ways of verifying them
Coping with Complexity

◆ Decomposition
  ◆ of the problem: separation of concerns
  ◆ of the object: exploit compositionality

◆ Formalization
  ◆ precise unambiguous semantics
  ◆ properties and formal techniques

◆ Abstraction
  ◆ eliminate unnecessary details

◆ Incremental Refinement
  ◆ include details while preserving properties
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

- Historically lots of work on Behavior
  - hierarchy well established
  - several descriptions available (with variable levels of precision)
  - synthesis

- Communication less well investigated
  - 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)
Phone call
Decomposition of Phone Call

Connect    Talk    Disconnect

ATM Cell

Bus Write
Refinement - Simple

substitute

repartition
Refinement - More Complex
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
Properties and Refinement

- Each model of computation guarantees some properties
- Protocols also imply properties
  - out of order, error correction, reliable packet delivery
- As you refine, must preserve properties
  - if order must be preserved
  - can refine to out-of-order communication method (like ethernet)
  - but have to reassemble in proper order
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 → Frame buffer → Video out

Host I/F → Video Decode

Onscreen Display
MPEG Architecture

Audio In → Audio Decode
Video In → Video Decode
Registers
Host I/F

shared bus

MMU/AGU

Memory buffers + frame buffer

Video out → Onscreen Display
Description Method

Algorithm

Map

Architecture
VCC Key Technology
Communication Refinement

Refinement from abstract tokens to articulated signals

Value

- Design and simulate at the level of abstraction at which designers think (e.g. ATM cell, GSM frame)
- hide implementation details of the communication until it is required (but simulate its overhead!)
- refine from abstract token level down to implementation of interface signals
- evaluate performance trade offs of communication effects
VCC Key Technology
Communication Interface Synthesis

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
Metropolis
**Metropolis Framework**

- **Design of Function Processes**
- **Design of Communication Media**
- **Design of Architecture Components**

**Metropolis Infrastructure**
- **Model of computation**
- Design methodology
  - Abstraction levels
  - Refinement
- **Base tools**
  - Design imports
  - User interface
  - Simulation

**Metropolis Point Tools:**
- **Synthesis/Refinement**
- **Analysis/Verification**
What is Communication?

\[ Os = Fs(is) \]

\[ Or = Fr(ir) \]
What is Communication?

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

Connection C enables the interaction between the behaviors S and R
What is Communication?

$$O_s = F_s(is)$$

$$O_r' = F_r'(i r')$$

Ideal Connection
What is Communication?

Os = Fs(is)

Or’ = Fr’(ir’)

S restricts the behavior of R to R’
Behavior Adaptation

\[ Os = Fs(is) \]

\[ Or = Fr(ir) \]

\[ \text{Ideal Connection} \]

\[ \text{R not defined for some output of S: behavior mismatch} \]
Behavior Adaptation

\[ Os = Fs(is) \]

\[ Or = Fr(ir) \]

Behavior Adapter \( Z \) maps outputs of \( S \) into the domain of \( R \)
Behavior Adaptation

\[ O_s' = F_s'(i_s') = F_s \circ Z'(i_s') \]

\[ O_r' = F_r'(i_r') = F_r \circ Z''(i_r') \]

- 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**
  - $S \rightarrow Z \rightarrow R$

- **Behavior Adaptation**
  - $S \rightarrow Z \rightarrow R$
  - $C \neq id$

- **Physical Channel selection**
  - $S \rightarrow Z' \rightarrow Z'' \rightarrow R$
  - $C \neq id$

- **Channel Adaptation**
  - $S \rightarrow Z' \rightarrow SC \rightarrow CR \rightarrow Z'' \rightarrow R$
  - $C \neq id$

- **Behavior Adaptation**
  - $S \rightarrow Z' \rightarrow SC \rightarrow Z''' \rightarrow Z'' \rightarrow CR \rightarrow Z'' \rightarrow R$
  - $C \neq id$
Protocol Design

(Ideal) Connection

Behavior Adaptation

Physical Channel selection

Channel Adaptation

Behavior Adaptation
**Metropolis: Model of Computation**

- **System function:** a network of processes
  - process: sequential function + ports

- **Do not commit to particular communication semantics**
  - ports: interconnected by **communication media**
  - communication media: define communication semantics
    - e.g. queues, shared memory, …, generic, …

- **Do not commit to particular firing rules of processes**
  - a special construct to define interaction between processes and media