Communication-Based Design

Why is System Design difficult?

- Semiconductor capability has outstripped design capability
- High-end consumer electronics (time-to-market) is driving the most complex chips now
- Consumer products combine multiple vastly disparate application areas (heterogeneity):
  - video
  - DSP
  - analog
  - protocols
- Nobody can possibly know all these areas in depth
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

Coping with Complexity

◆ Automatic Synthesis
◆ Formal Verification
◆ Simulation
◆ Design Reuse - IP
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**

[Diagram of a phone call]
**Decomposition of Phone Call**

**Refinement - Simple**
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

Host I/F → Frame buffer

Video out

Onscreen Display

MPEG Architecture

Audio In → Audio Decode

Video In → Video Decode

Registers

Host I/F

shared bus

MMU/AGU

Memory: buffers + frame buffer

Onscreen Display
**Description Method**

- **Algorithm**
  - Audio In → Audio Decode
  - Video In → Video Decode
  - Host I/F → Frame buffer → Onscreen Display

- **Architecture**
  - Audio In → Audio Decode
  - Video In → Video Decode
  - Host I/F → Frame buffer → Onscreen Display

---

**Simulation Speed - Cheetah**

- Take advantage of abstraction and decomposition
- Note that protocol has “anchor points”
  - timing is data or contention dependent
  - other actions shift with constant offset from anchor
- Should only need to simulate the anchor points
- Constant offset actions are “inevitable” and can be abstracted
- GUI can recreate inevitable behavior on demand
- Replacing non-branching state sequences with delay

Source: James Rowson
**ATM Example**

- Must simulate the bus, because contention is a major factor
- Can abstract away the PI bus detail because
  - anchor point is when arbitration grants master the bus
  - might be another anchor point when slave is ready (here constant time)
  - all other actions follow with constant offset from these
- Important to note that cell creator and transmitter code is unmodified in these examples

Source: James Rowson
**Cheetah Most Abstract**

Source: James Rowson

**Mapping ATM cell onto Bus**

Token

53 bytes

decompose

Bus

write 500 write 501 write 501 ... write 501

Source: James Rowson
Cheetah Abstract Bus - burst == 1

Source: James Rowson

Cheetah Abstract Bus - burst == 10

Source: James Rowson
Cheetah PI Bus - burst == 1

Cheetah PI Bus - burst == 10

Source: James Rowson
**Cheetah - Performance**

- **Most abstract** 38x faster than most detailed
  - measured by CPU time
- **Most detailed** (PI bus, burst==1) expands by 7x
  - each bus transaction expands into 7 “assignments” in history
- **Postulate performance comparison with HDL**
  - dominated by event traffic
  - event traffic proportional to size of simulation history
  - can compare relative performance through history expansion
- **Worst case** is 7x performance improvement over HDL
  - as bursts get longer, get proportionally better performance

Source: James Rowson
**Cheetah Performance Analysis**

Space == # wires

Time = # clocks

Speedup ~ space expansion \* time expansion

as burst size increases, so does speedup

Source: James Rowson

---

**Metropolis**

Metropolis Project Team
University of California Berkeley
Cadence Berkeley Laboratories
**Metropolis Framework**

![Diagram of Metropolis Framework]

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

- **Metropolis Point Tools**
  - Synthesis/Refinement
  - Analysis/Verification

---

**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

---

![Diagram of Metropolis Model of Computation]
Functional Decomposition

- **Functional Decomposition**
  - at the highest abstraction level, a system is a single process
  - it is refined into a set of concurrent processes

- **Process:**
  - relation between an input domain and an output co-domain
  - only behavior, no communication
  - denotational specification
**Functional Decomposition (ex.)**

MPEG Decoder

\[ \text{VLD} \rightarrow \text{IDCT} \rightarrow \text{MC} \rightarrow \text{DISPLAY} \]

**Behavior Adaptation**

- **Behavior adapters**
  - match different domains, so that processes can understand each other
  - relation between two domains
  - not part of original system specification: needed because of the particular decomposition
  - needed independently of how the communication is performed
**Behavior Adaptation (ex.)**

- VLD → vect
- Idct → blkvect
- MC → mot. vect
- Mblock → Mblock
- DISPLAY

**Communication and MoC**

- **Communication medium**
  - each link needs a communication medium
  - does not affect or change the relation inside processes

- **MoC wrapper**
  - used to establish a firing rule and a communication semantics for each process
  - only the Moc wrapper is modified if a medium is changed
Communication and Mod (ex.)

Refinement

- Refinement
  - any communication medium can be refined into an arbitrary netlist, as long as the interface is not changed

- Channel adapters
  - used to preserve properties of a given interface
  - example: lossless communication realized with a lossy medium (retransmission + acknowledge)
Refinement (ex.)

Mapping and Optimization

- **Optimization**
  - map each element (processes, adapters, media) onto architecture
  - merge processes, adapters and media into a single process, when applicable
  - provide an imperative description for each process
Mapping and Optimization (ex.)