Lecture 9: Outline

- Communication layers
  - Application
  - Network: presentation, session, transport
  - Protocol: link, stream, media access, physical

- Transaction-level modeling
  - Abstraction levels
  - Speed & accuracy tradeoffs

- Communication synthesis
  - Protocol stack generation & optimization
  - Backend HW/SW interface generation
Specification-Level Communication

• **Events, transitions**
  • Pure control flow, no data

• **Shared variables**
  • No control flow, no synchronization

• **Synchronous message passing**
  • No buffering, two-way control flow

• **Asynchronous message passing**
  • Only control flow from sender to receiver guaranteed
  • May or may not use buffers (implementation dependent)

• **Queues**
  • Fixed, defined queue length (buffering)

• **Complex channels**
  • Semaphores, mutexes

➢ **Reliable communication primitives (lossless, error-free)**

---

Taxonomy of “Busses”

• For each transaction between two communication partners
  • 1 sender, 1 receiver
  • 1 master (initiator), 1 slave (listener)

➢ **Any combination of master/slave, sender/receiver**
  ➢ Master/Slave bus
    ➢ Statically fixed master/slave assignments for each PE pair
    ➢ PEs can be masters, slaves or both (dual-port)
  ➢ Node-based bus (e.g. Ethernet, CAN):
    ➢ Sender is master, receiver is slave

➢ **Reliable (loss-less, error-free)??**
Communication Modeling

- **Pin-accurate model (PAM)**
  - Simulate every event (protocols)
- **Transaction-level model (TLM)**
  - Communications by transactions (abstract channels)
  - Granularity of transactions? Dynamic effects?

Source: OSCI TLM-2.0

Communication Layers

- **ISO/OSI 7-layer network model**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Semantics</th>
<th>Functionality</th>
<th>Implementation</th>
<th>OSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Channels, variables</td>
<td>Computation</td>
<td>Application</td>
<td>7</td>
</tr>
<tr>
<td>Presentation</td>
<td>End-to-end typed messages</td>
<td>Data formatting</td>
<td>Application</td>
<td>6</td>
</tr>
<tr>
<td>Session</td>
<td>End-to-end untyped messages</td>
<td>Synchronization, Multiplexing</td>
<td>OS kernel</td>
<td>5</td>
</tr>
<tr>
<td>Transport</td>
<td>End-to-end data streams</td>
<td>Packeting, Flow control, Error correction</td>
<td>OS kernel</td>
<td>4</td>
</tr>
<tr>
<td>Network</td>
<td>End-to-end packets</td>
<td>Routing</td>
<td>OS kernel</td>
<td>3</td>
</tr>
<tr>
<td>Link</td>
<td>Point-to-point logical links</td>
<td>Synchronization</td>
<td>Driver</td>
<td>2b</td>
</tr>
<tr>
<td>Stream</td>
<td>Point-to-point control/data streams</td>
<td>Multiplexing, Addressing</td>
<td>Driver</td>
<td>2b</td>
</tr>
<tr>
<td>Media Access</td>
<td>Shared medium byte streams</td>
<td>Data slicing, Arbitration</td>
<td>HAL</td>
<td>2a</td>
</tr>
<tr>
<td>Protocol</td>
<td>Media (word/frame) transactions</td>
<td>Protocol timing</td>
<td>Hardware</td>
<td>2a</td>
</tr>
<tr>
<td>Physical</td>
<td>Pins, wires</td>
<td>Driving, sampling</td>
<td>Interconnect</td>
<td>1</td>
</tr>
</tbody>
</table>

➢ A model, not an implementation!
From Layers to System Models…

- **Cycle Accurate Model**
- **Transaction Level Models**
- **Specification Model**

**MP Channel**

- **Abstract, high-level system functionality**
  - **Computation**
    - Processes
    - Variables
  - **Communication**
    - Sync./async. message-passing
    - Memory interfaces
    - Events
Application Layer (1)

- **Synchronization**
  - Synthesize control flow

  ![Diagram showing synchronization between processes]

  - Implement sequential transitions across parallel components

Application Layer (2)

- **Storage**
  - Shared variable mapping to memories

  ![Diagram showing storage and memory mapping]

  - Map global storage to local memories
Application Layer (3)

- **Channels**
  - Complex channel synthesis

- **Client-server implementation**
  - Server process
  - Remote procedure call (RPC) channels

Architecture Model

- **Application layer (virtual system architecture)**
  - Computation
    - PEs (functionality)
    - Memories (storage)
  - Abstract end-to-end communication
    - Queues, semaphores
    - Sync./async. message-passing
    - Shared variables/memories
    - Events, transitions

- **Reliable, loss-less application communication**
Presentation Layer

- **Data formatting**
  - Translate abstract data types into canonical network byte layout
    1. Global network data layout
    2. Shared, optimized layout for each pair of communicating PEs
  - Convert typed messages into untyped, ordered byte streams
  - Convert variables into memory byte layout

- **Bitwidth of machine character**
  - Smallest addressable unit
- **Size and Alignment (in characters)**
- **Endianness**

Session Layer

- **Channel merging**
  - Merge application channels into a set of untyped end-to-end message streams
    1. Unconditionally merge sequential channels
    2. Merge concurrent channels with additional session ID (message header)
  - Channel selection over end-to-end transports
Network Layer

- **Split network into subnets**
  - Routing of end-to-end paths over point-to-point links
  - Insert communication elements (CEs) to connect busses

- **Bridges**
  - Transparently connect slave & master sides at protocol level
  - Bridges maintain synchronicity, no buffering

- **Transducers**
  - Store-and-forwarding of data packets between incompatible busses
  - Intermediate buffering, results in asynchronous communication

Transport Layer

- **Packing and routing**
  - Packetization to reduce buffer sizes
    1. Fixed packet sizes (plus padding)
    2. Variable packet size (plus length header)
  - Protocol exchanges (ack) to restore synchronicity
    - Iff synchronous message passing and transducer in the path
  - Packet switching and identification (logical routing)
    1. Dedicated logical links (defer identification to lower layers)
    2. Network endpoint addressing (plus packet address headers)
  - Physical routing in case of multiple paths between PEs
    1. Static, predetermined routing (based on connectivity.headers)
    2. Dynamic (runtime) routing
Network Model

- **Topology of communication architecture.**
  - PEs + Memories + CEs
  - Upper protocol layers inserted into PEs/CEs
  - Communication via point-to-point links
    - Synchronous packet transfers (data transfers)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)

- **Network layers**
  - PEs + Memories + CEs
  - Transducers (store-and-forward)
  - Point-to-point link communication
    - Synchronous packet transfers (data link channels)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)
**Link Layer (1)**

- **Synchronization (1)**
  - Ensure slave is ready before master initiates transaction
    1. Always ready slaves (memories and memory-mapped I/O)
    2. Defer to fully synchronized bus protocol (e.g. RS232)
    3. Separate synchronization mechanism

  ![Diagram](image)

  - Events from slave to master for master/slave buses
  - Synchronization packets for node-based busses

**Link Layer (2)**

- **Synchronization (2)**
  - Dedicated interrupts
  - Shared interrupts

![Diagram](image)
**Link Layer (3)**

- **Synchronization (3)**
  - Slave polling
  - Flag in master

**Stream Layer**

- **Addressing**
  - Multiplexing of links over shared medium
  - Separation in space through addressing
  - Assign physical bus addresses to links
    1. Dedicated physical addresses per link
    2. Shared physical addresses plus packet ID/address in packet header
Media Access (MAC) Layer

- **Data slicing**
  - Split data packets into multiple bus word/frame transactions
  - Optimized data slicing utilizing supported bus modes (e.g. burst)

- **Arbitration**
  - Separate individual bus transactions in time
    1. Centralized using arbiters
    2. Distributed
  - Insert arbiter components

Protocol TLM

- **Abstract component & bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Communication layers down to protocol transactions
  - Communication via transaction-level channels
    - Bus protocol transactions (data transfers)
    - Synchronization events (interrupts)
Protocol TLM

- Link layers
  - PEs + Memories + CEs + Busses
  - Bus bridges
  - Communication via bus transactions (bus TLM)
    - Address, data, arbitration
    - Synchronization (interrupts, polling)

**System communication architecture**

Protocol, Physical Layers

- Bus interface
  - Generate state machines implementing bus protocols
  - Timing-accurate based on timing diagrams and timing constraints

**Bus protocol database**

- Port mapping and bus wiring
  - Connectivity of component ports to bus, interrupt wires/lines
  - Generate top-level system netlist
Pin-/Bus Cycle-Accurate Model (P/BCAM)

- Component & bus structure/architecture
  - PEs + Memories + CEs + Busses
  - Pin-accurate bus-functional components
  - Pin- and cycle-accurate communication
    - Bus and interrupt protocols
    - Pins and wires

Pin-Accurate Model (PAM)

- Bus-functional layers
  - PEs + Memories + CEs + Busses
    - Pin-, cycle- and bit-accurate bus-functional components
  - Communication via ports and wires
    - Address, data, control busses
    - Interrupts
Lecture 9: Outline

- Communication layers
  - Application
  - Network: presentation, session, transport
  - Protocol: link, stream, media access, physical

- Transaction-level modeling
  - Abstraction levels
  - Speed & accuracy tradeoffs

- Communication synthesis
  - Protocol stack generation & optimization
  - Backend HW/SW interface generation

TLM Abstraction Levels

- ISO/OSI reference layer-based architecture
  - Granularity of data and arbitration handling

- Layers:
  1) Media Access Control (MAC)
     - User Transaction
       » Contiguous block of bytes
       » Arbitrary length, base address
  2) Protocol
     - Bus Transaction
       » Bus primitives (e.g. store word)
       » Observes bus address restrictions
  3) Physical
     - Bus Cycle
       » Drive or sample bus wires on bus cycle

- Models are composed of layers
  - Using fewer layers yields a more abstract model

Bus Functional Model (BFM)

- May or may not be pin accurate
- Bus cycle accurate
  - Arbitration check on each cycle
- Includes additional active components
  - Multiplexers (tri-state-free bus)
  - Arbiter
  - Address decoder
  - Clock generator

Implemented Layers:
- MAC
- Protocol
- Physical

Granularity:
- User Transaction (1)
- Bus Transaction (2)
- Bus Cycle (3)

Transaction Level Model (TLM)

- Bus primitives
  - StoreWord, StoreBurst4
- Abstract model
  - Not pin accurate, not bus cycle accurate in all cases
- Priority arbitration per bus transaction
  - May lead to wrong arbitration decision, depending on execution order

Implemented Layers:
- MAC
- Protocol

Granularity:
- User Transaction (1)
- Bus Transaction (2)
- Bus Cycle (3)
MAC-TLM

Implemented Layers:

MAC

Granularity:

• User transaction (message)
  - Arbitrary length, contiguous block of bytes

• No arbitration: contention avoidance by semaphore
  - Resolution depends on simulator

➤ Expected to be the fastest model
  - Single memcpy, Single time wait

TLM Accuracy Limitations

• Example:
  - Low priority burst starting at \( t_0 \)
  - High priority preemption at \( t_1 \)

• BFM:
  - check every cycle
  - slow
  - accurate

• TLM:
  - coarse grain check
  - fast
  - inaccurate: low prio. burst ends at \( t_2 \) instead of \( t_4 \)
TLM Trade-Off

- **Bus models**
  - AMBA AHB, CAN, ColdFire
  - BFM, TLM, MAC-TLM

- **Performance analysis**
  - 2 masters, 2 slaves
  - Randomly distributed traffic
  - 100 byte transactions
  - 40% bus contention
  - Transfer duration

<table>
<thead>
<tr>
<th>Model</th>
<th>Speed</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-TLM</td>
<td>&lt;100 MByte/s</td>
<td>32% - 47%</td>
</tr>
<tr>
<td>TLM</td>
<td>~ 1 MByte/s</td>
<td>18% - 39%</td>
</tr>
<tr>
<td>BFM</td>
<td>&lt;0.2 MByte/s</td>
<td>0%</td>
</tr>
</tbody>
</table>

SystemC/TLM

- **Loosely-timed**
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: *begin* (call) and *end* (return)
  - Typically blocking transport calls

- **Approximately-timed**
  - Cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Each transaction has at least 4 timing points
  - Typically non-blocking transport calls

*Source: OSCI TLM-2.0*
Advanced TLM

- **Modeling tricks [SystemC/TLM]**
  - Early completion of non-blocking REQ/RESP sequence
  - Timing annotation on call/return (pass)
  - Temporal decoupling & quantum (lump, out-of-order)
  - Direct interface (bypass)

- **Advanced modeling approaches (see OS modeling)**
  - Conservative [ATGA]
  - Optimistic [ROM]

---

Result-Oriented Modeling (ROM)

- **Characteristics**
  - Observability at boundary of transaction
    - Internal state changes not visible, may not be modeled
  - **Optimistically predict** the *end result* at beginning
    - optimistic == earliest time to finish
  - Record *disturbing influence*
  - **Corrective measures** at the end

Result Oriented Modeling (ROM)

- Same example transfer:
  - Low priority burst at $t_0$
  - High priority preemption at $t_1$

- ROM:
  - $t_0$ low: predict $t_2$
  - $t_1$ high: preempt, predict $t_3$, record preemption for low
  - $t_2$ low: detect disturbance, prediction update $t_4$
  - $t_3$ high: no preemptions, finish
  - $t_4$ low: no preemptions, finish
  - Accurate

ROM Results

- Two concurrent masters
  - High priority: 33% bus load
  - Measure bandwidth of low priority

- 100% accuracy
  - ROM, BFM both on top
  - No error

- At coarse-grain speeds
  - ROM and TLM are in same magnitude!
  - CAN ROM can not outperform TLM
    - Data dependent length
    - Bit inspection
      - CRC, stuffing bits
Lecture 9: Outline

- Communication layers
  - Application
  - Network: presentation, session, transport
  - Protocol: link, stream, media access, physical

- Transaction-level modeling
  - Abstraction levels
  - Speed & accuracy tradeoffs

- Communication synthesis
  - Protocol stack generation & optimization
  - Backend HW/SW interface generation

Protocol Stack Optimizations (1)

- Automatically generated code during refinement
  - Layer-based code organization and separation

```plaintext
P1
- c1.send(m1);
- c2.send(m2);
- c3.send(m3);

Presentation:
- send(type msg) {
  char buf[M];
  1: msg->buf;
  2: net.send(buf);
}

Network:
- send(d) {
  1: for (p in d) {
    link.send(d[p]);
    2: link.recv(ack);
  }
}

Link:
- send(p) {
  1: wait(intr);
  2: mac.write(p);
}

MAC:
- write(b) {
  1: for(w in b) {
    protocol.write(w);
  }
}

Data slicing

Synchronization
```

(c) 2015 A. Gerstlauer
Protocol Stack Optimizations (2)

- Apply automatic code optimizations during refinement
- Code optimized for HW/SW synthesis
- Layer merging and cross-optimizations (inline/interleave)

```plaintext
PE

send(type1 msg1, type2 msg2) {
  word buf;
  1: wait(intr)
  2: for (w in msgs) {
      msg[w]->buf;
      protocol.write(buf);
  }
  3: wait(intr);
      protocol.read(ack);
}
```

Interface Synthesis (1)

- Bus Interface
- FSM / FU
- Protocol FSMD
- RTL Database / Generator / IP
- PE1 Computation FSM
- PE2 Computation FSM
- Interface of I-POX
- Protocol FSM
- Data
- Control
Interface Synthesis (2)

- PE1 Computation/protocol FSMs
- Bus Interface FSM/FU
- Interrupt generation
- RTL Database / Generator / IP

Transducer Synthesis

- T1
- RTL Database / Generator / IP
- Inline and compile/synthesize
- RTL Database / Generator / IP

(c) 2015 A. Gerstlauer
Cycle-Accurate Model (CAM)

- Component & bus implementation
  - PEs + Memories + CEs + Busses
  - Cycle-accurate components
    - Instruction-set simulators (ISS) running final target binaries
    - RTL hardware models
    - Bus protocol state machines

Lecture 9: Summary

- Communication modeling & refinement
  - Systematic, structured communication design flow
    - Layer-based modeling and refinement
    - Well-defined levels, models and design steps
    - Support for rich applications and wide variety of target architectures
    - Transaction-level modeling (TLM) of communication
      - Rapid, early feedback, validation and exploration
      - Various levels of abstraction, accuracy vs. speed tradeoffs

- Communication synthesis
  - Generation of communication layer implementations
    - Application and target-architecture specific, customized and optimized
  - Protocol stack optimizations
    - Merging and cross-optimizations of layers
  - Interface backend synthesis