What is TileLink?

A protocol for connecting masters (like cores) with slaves (like memory)

- A single message vocabulary with many different serializations
- Three conformance levels for reduced messages in simple devices
- Full protocol supports moving both data and ownership
## Data movement

TileLink *agents* (masters/slaves) move data in messages across a link

<table>
<thead>
<tr>
<th>Master Request</th>
<th>Slave Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put (write)</td>
<td>AccessAck (success/failure)</td>
</tr>
<tr>
<td>Get (read)</td>
<td>AccessAckData (current value)</td>
</tr>
</tbody>
</table>

- TL-UL (simplest conformance level) only supports Get/Put
- TL-UH adds atomic operations and hints
Example: Data movement

- Core 1
- Clock Crossing
- Router
- Memory 1
- Bus Width Adapter
- Memory 2
- Core 2
- Master
- TileLink
- Slave
Example: Data movement

Every TileLink link has a master end and a slave end.
The boxes are the *agents*, which may have multiple TileLink links.
TileLink does not define what happens inside an agent.
Example: Data movement

TileLink messages are only defined on a per-link basis
Example: Data movement

But TileLink messages may get forwarded over many links
Example: Data movement

AccessAckData

Core 1 → Clock Crossing → Router → Memory 1

AccessAckData

Core 2 → Router → Bus Width Adapter → Memory 2

Some messages flow the other direction (Slave => Master)
Why Ownership Movement?

• Gets and Puts leave ownership of the data at the Slave
• Masters always know where to send their reads/writes
• The Slave decides the order in which requests are serviced

=> This model works well for streaming reads/writes (DMA)

What if the data is far away AND you want to use it more than once?
• Repeated accesses waste traffic and power on messages
• Latency sensitive devices (like cores) suffer a performance hit
Degrees of ownership

• Who has the block?
  – Unique only one agent
  – Shared many agents

- Slave can service reads/writes
- Master sends requests to slave

- Master can service reads/writes
- Slave sends requests to master

- Slave+Master can service reads
- Ownership must change for writes
Ownership Exchange

- TileLink defines messages to increase/decrease ownership
- These operations can be initiated by either Slave or Master

<table>
<thead>
<tr>
<th>Master Initiated</th>
<th>Slave Increases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquire</td>
<td>Release</td>
</tr>
<tr>
<td>(Stash)</td>
<td>Probe</td>
</tr>
</tbody>
</table>

The addition of these message defines the third conformance level (TL-C)
Protocol Definition

To actually implement any bus protocol you need:
- Message definitions (we’ve covered this now)
- Field definitions (size, sender, address, ... -- see the spec)
- Ordering (message A may/must wait on message B)
- Serialization (how messages are encoded on the wire)

TileLink defines ordering using 5 priority levels
Every message has a specified priority
Conforming systems are proven not to deadlock

... but TileLink can be serialized many different ways
Parallel On-Chip Serialization (TL v1.8)

clock

a_ready

a_valid

Get

Get

a_opcode

4

4

5

5

da_ready

da_valid

AccessAckData

AccessAckData

d_opcode

1

1

5

5

d_size
Different Serializations for Different Needs

On-chip

- Parallel ready/valid bus (v1.8)
- Packet-based message interconnect (NoC)

Off-chip

- Packet-based parallel credit bus (ChipLink/proprietary)
- … over the network? (see next talk!)
**TileLink in a real SoC**

[Diagram showing the architecture of a real SoC with TileLink integration.]
TileLink in a real SoC
Summary

TileLink defines a singular message vocabulary
• Greatly enhances interoperability
• Composable; no costly protocol converters
• Pay only for what you need (TL-UL / TL-UH / TL-C)

That vocabulary can be serialized in many ways
• Parallel or packet
• Ready/valid or credits
• On-chip or off-chip
Any Questions?