Table of Contents

Introduction

TileLink is a protocol designed to be a substrate for cache coherence transactions in an on-chip memory hierarchy. Its purpose is to orthogonalize the design of the on-chip network and the implementation of the cache controllers from the design of the coherence protocol itself. Any cache coherence protocol that conforms to TileLink's transaction structure can be used interchangeably with the physical networks and cache controllers we provide.

TileLink is roughly analogous to the data link layer in the IP network protocol stack, but exposes some details of the physical link necessary for efficient controller implementation. It also codifies some transaction types that are common to all protocols, particularly the transactions servicing memory accesses made by agents that do not themselves have caches.

TileLink Architecture

Agents

TileLink assumes a Client/Manager architecture where agents participating in the coherence protocol are either:

- clients requesting access to cache blocks, or
- managers overseeing the propagation of cache block permissions and data

A client may be a cache, a DMA engine, or any other component that would like to participate in the coherent memory domain, regardless of whether or not it actually keeps a copy of the data locally. A manager may be an outer-level cache controller, a directory, or a broadcast medium such as a bus. In a multi-level memory hierarchy, a particular cache controller can function as both a client (wrt caches further out in the hierarchy) and a manager (wrt caches closer to the processors).

Channels

TileLink defines five independent transaction channels. These channels may be multiplexed over the same physical link, but to avoid deadlock TileLink specifies a priority amongst the channels that must be maintained. Channels may contain both metadata and data components. The channels are:

- **Acquire.** Initiates a transaction to acquire access to a cache block with proper permissions. Also used to write data without caching it.
- **Probe.** Queries an agent to determine whether it has a cache block or revoke its permissions on that cache block.
- **Release.** Acknowledgement of probe receipt, releasing permissions on the line along with any dirty data. Also used to voluntarily write back data.
- **Grant.** Provides data or permissions to the original requestor granting, access to the cache block. Also used to acknowledge voluntary Releases.
- **Finish.** Final acknowledgement of transaction completion from requestor, used for transaction ordering.

At present time, all channels are routed from clients to managers or from managers to clients. In the future, client-to-client Grants may be added.

The prioritization of channels is Finish >> Grant >> Release >> Probe >> Acquire. Preventing messages of a lower priority from blocking messages of a higher priority from being sent or received is necessary to avoid deadlock.

Transaction Flows

There are two types of transaction that can occur on a cache block managed by TileLink. The first supports clients acquiring a cache block:

- A client sends an Acquire to a manager
- The manager sends any necessary Probes to clients
- The manager waits to receive a Release for every probe that was sent
- The manager communicates with backing memory if required

- Having obtained the required data or permissions, the manager responds to the original requestor with a Grant
- Upon receiving a Grant, the original client responds to the manager with a Finish to complete the transaction

The second type of transaction is supports clients voluntarily releasing a cache block:

- A client sends a Release to a manager, specifying that it is voluntary
- The manager communicates with backing memory if required
- The manager acknowledges completion of the transaction using a Grant

Concurrency

TileLink does not make any assumptions about the ordering of messages sent point-to-point over particular channels. Therefore, concurrency must be managed by agents at several points in the system.

- A manager should not accept a request for a transaction on a block that is already in-flight for a different client (unless it knows how to merge the two transactions as discussed below). Specifically, the manager must wait until it has received a Finish from the original client in order to ensure proper ordering of any future Grants.
- If client has an outstanding voluntary writeback transaction, it cannot respond to an incoming Probe request on that block with Releases until it receives a Grant from the manager acknowledging completion of the writeback.

Transactions can be merged in certain situations. One specific situation that must be handled by all manager agents is receiving a voluntary Release for a block which another client is currently attempting to Acquire. The manager must accept the voluntary Release as well as any Releases resulting from Probe messages, and provide Grant messages to both clients before the transaction can be considered complete.

When running on networks that provide guaranteed ordering of messages between any client/manager pair, the Finish acknowledgment of a Grant (and the Grant acknowledgement of a voluntary Release) can be omitted.

Channel Signal Descriptions

This section details the specific signals contained in each channel of the TileLink protocol. Every channel is wrapped in the DecoupledIO interface, meaning that each contains ready and valid signals as well as the following. Channels with data may send the data over multiple

beats; the width of the underlying network is exposed to improve the efficiency of refilling data into caches whose data array rows are of a matching size.

Acquire

Initiates a transaction to acquire access to a cache block with proper permissions. Also used to write data without caching it (acquiring permissions for the write as it does so).

addr_block	UInt	Physical address of the cache block, with block offset removed
client_xact_id	UInt	Client's id for the transaction
data	UInt	Client-sent data, used for uncached writes
addr_beat	UInt	Offset of this beat's worth of data within the cache block
built_in_type	Bool	Whether the transaction is a built-in or custom type
a_type	UInt	Type of the transaction For built-in transactions, one of: [UncachedRead, UncachedWrite, UncachedAtomic, UncachedReadBlock, UncachedWriteBlock] Otherwise defined by the coherence protocol
subblock	Union	Used in uncached subblock transactions, possible subfields below:
allocate	Bool	R/W: Hints whether to allocate data in outer caches when servicing this request
operand_sz	UInt	A: Size of the request (Byte, Half, Word, Double)
subblock_addr	UInt	A: Address of the operand within the block
atomic_op	UInt	A: AMO ALU opcode
write_mask	UInt	W: Byte mask for write data

There are five built-in types of Acquire that are available to all clients that want to participate in the coherence protocol, even if they themselves will not keep cached copies of the data. Because these transactions do not create a new private copy of the targeted cache block, they are termed "uncached" transactions. The available uncached transactions are as follows:

- UncachedReadBlock: Fetches an entire cache block and serves it back to the requestor.
- UncachedRead: Fetched a single beat of data from a cache block and returns only that beat
- UncachedWriteBlock: Writes out an entire cache block to backing memory

- UncachedWrite: Writes up to a beat's worth of data to backing memory. Uses a write mask to determine which bytes contain valid data
- UncachedAtomic: Performs an atomic memory op in backing memory. The maximum available operand size is 64b (sizes and opcodes per RISC-V atomic insts).

Probe

Queries an agent to determine whether it has a cache block or revoke its permissions on that cache block.

addr_block	UInt	Physical address of the cache block, with block offset removed
p_type	UInt	Transaction type, defined by coherence protocol

Release

Acknowledgement of probe receipt, releasing permissions on the line along with any dirty data. Also used to voluntarily write back data or cede permissions on the block.

addr_block	UInt	Physical address of the cache block, with block offset removed
r_type	UInt	Transaction type, defined by coherence protocol
client_xact_id	UInt	Client's id for the transaction
data	UInt	Used to writeback dirty data
addr_beat	UInt	Offset of this beat's worth of data within the cache block
voluntary	Bool	Whether this release is voluntary or in response to a Probe

Grant

Provides data or permissions to the original requestor granting, access to the cache block. Also used to acknowledge voluntary Releases.

built_in_type	Bool	Whether transaction type is built-in or custom
g_type	UInt	Type of the transaction For built-in transactions, one of: [VoluntaryAck, UncachedRead, UncachedWrite, UncachedAtomic, UncachedReadBlock] Otherwise defined by the coherence protocol

client_xact_id	UInt	Client's id for the transaction
manager_xact_id	UInt	Manager's id for the transaction, passed to Finish
data	UInt	Used to supply data to original requestor
addr_beat	UInt	Offset of this beat's worth of data within the cache block

There are five built-in types of Grant that are available to all managers that want to participate in the coherence protocol. Because "uncached" transactions do not create a new private copy of the targeted cache block, we use these Grant types mostly as acknowledgements. The available types are as follows:

- UncachedReadBlock: Full cache block in response to Acquire.UncachedReadBlock
- UncachedRead: Single beat of data in response to Acquire.UncachedRead
- UncachedWrite: Acknowledgement of Acquire.{UncachedWrite,UncachedWriteBlock}
- UncachedAtomic: Single beat of data in response to Acquire.UncachedAtomic, response to atomic op is stored at the originally specified address within the beat
- VoluntaryAck: Acknowledgement of any voluntary Release

Finish

Final acknowledgement of transaction completion from requestor, used for transaction ordering.

manager_xact_id	UInt	Manager's id for the transaction

Parameters

This section defines the parameters that are exposed by the TileLink to the top-level design. All agents that implement TileLink should either work for all values of these parameters within the specified ranges, or should add Chisel.Constraints to the design to define functional limits on hem.

TLId	String	Which TileLink in a multi-level hierarchy
TLCoherence	CoherencePolicy	Coherency policy used on this TileLink
TLAddrBits	Int	Address size
TLManagerXactIdBits	Int	Size needed to track outstanding xacts

TLClientXactIdBits	Int	Size needed to track outstanding xacts
TLDataBits	Int	Amount of block data sent per beat
TLDataBeats	Int	Number of beats per cache block