An Open and Coherent Memory Centric Architecture Enabled by RISC-V

Marjan Radi, Wesley Terpstra, Tu Dang, Paul Loewenstein, Dave Parry, Dejan Vučinić
An Open and Coherent Memory Centric Architecture Enabled by RISC-V

Marjan Radi, Wesley Terpstra, Tu Dang, Paul Loewenstein, Dave Parry, Dejan Vučinić
Forward-Looking Statements

Safe Harbor | Disclaimers

This presentation contains forward-looking statements that involve risks and uncertainties, including, but not limited to, statements regarding our contributions to and proposals for the RISC-V ecosystem, technology and product development, business strategies and growth opportunities, the capabilities and features of our RISC-V cores, expectations regarding data growth and its drivers, and industry trends. Forward-looking statements should not be read as a guarantee of future performance or results, and will not necessarily be accurate indications of the times at, or by, which such performance or results will be achieved, if at all. Forward-looking statements are subject to risks and uncertainties that could cause actual performance or results to differ materially from those expressed in or suggested by the forward-looking statements.

Key risks and uncertainties include volatility in global economic conditions; business conditions and growth in the storage ecosystem; unexpected advances in competing technologies; our development and introduction of products based on new technologies and expansion into new data storage markets; the impact of competitive products and pricing; actions by competitors; risks associated with acquisitions, mergers and joint ventures; difficulties or delays in manufacturing; and other risks and uncertainties listed in the company’s filings with the Securities and Exchange Commission (the “SEC”) and available on the SEC’s website at www.sec.gov, including our most recently filed periodic report, to which your attention is directed. We do not undertake any obligation to publicly update or revise any forward-looking statement, whether as a result of new information, future developments or otherwise, except as required by law.
Western Digital Proposes Open Standard Interface for Memory Fabric—OmniXtend™

Data is the center of the architecture
No established hierarchy—CPU doesn’t ‘own’ the GPU or the Memory
Preserved Cache Coherency over the Network
OmniXtend vs. other memory-centric concepts

- Memory fabric may mean different things to different people:
  - Page fault trap leading to RDMA request (incurs context switch and SW overhead)
  - Global address translation management in SW, leading to LD/ST across global memory fabric
  - Coherence protocol scaled out, global page management and no context switching

Context switch cost comparable to memory access latency

Require software/kernel support and/or rewriting of applications

This is OmniXtend
No rewriting of software, scalable like the algorithm
OmniXtend Architectures

Point to Point

Through an Ethernet Switch

Program switch to replace ethernet protocol with OmniXtend protocol to handle memory coherency traffic
First “real world” measurements

1. SiFive
   RISC-V SoC with OmniXtend running in FPGA

2. BAREFOOT NETWORKS
   Tofino Switch programmed with P4 code to support OmniXtend
First “real world” OmniXtend 0.1.1 measurements

Global memory, no local DRAM like in RDMA
SoC with OmniXtend support in 2020

• Western Digital and SiFive collaborating on OmniXtend
  – Interconnects Workgroup in CHIPS Alliance
  – Private meetings on SoC

• SoC architecture:
  – Quad core, Linux capable U74 RV64GC cores; 64KB private L1I$ and L1D$; 2MB shared L2$
  – PCIe Gen4x8 controller for 8X16Gb/s lanes
  – LPDDR4 controller

• Tapeout:
  – 3Q2020 (FPGA validation in progress)

• Reference boards for Houdini
  – Compute boards
  – Memory boards (with LF NAND or other technology)
Houdini: Memory Fabric Innovation Platform

Allegro files now in:
TileLink Cache Coherence Requests and States

Coherence protocols transmit permissions, not just data

• **Acquire**—sent by master to obtain access permissions from slave.

• **Probe**—sent by slave to master to obtain remove access permissions.

• **Release**—sent by master to slave to relinquish all access permissions.

• **Grant**—sent by slave to master to grant access permissions.

• TileLink Cache Coherence supports four different primary cache states:
  - M  Exclusive modified (read/write access with obligation to write data back upon eviction from cache).
  - E  Exclusive clean (read/write access, data can be discarded upon eviction).
  - S  Shared clean (read-only access, data can be discarded upon eviction).
  - I  No access.

• There is no shared dirty state in TileLink.
TileLink Network Channels

- Uses 5 channels:
  
  A  For requests by protocol “master”.
  B  For probe requests by protocol “slave” to protocol “masters”.
  C  For responses and release requests from “masters” to “slave”.
  D  For grants and release acks from “slave” to “master”.
  E  For grant ack from “master” to “slave”

  where later channels can back-pressure earlier channels.

- Using channel C for release allows allocating request to be sent before space is available for received data.
Recursive Acquire

Acquire

L1$  L1$  L1$  L1$  L1$  L1$  L1$  L1$

L2$  L2$  L2$  L2$  L2$  L2$  L2$  L2$

L3$  L3$  L3$  L3$
Recursive Acquire
Recursive Acquire
Recursive Acquire

L1$ → L2$ (Probe)

L1$ → L2$ (Probe)

L1$ → L2$ (Probe Ack)

L1$ → L2$ (Probe)

L1$ → L2$ (Probe)

L1$ → L2$ (Probe)

L3$ → L3$ (Probe)
Recursive Acquire
Recursive Acquire
Recursive Acquire

L1$ \rightarrow L2$ \rightarrow L3$

L1$ \rightarrow L2$ \rightarrow L3$

L1$ \rightarrow L2$ \rightarrow L3$

L1$ \rightarrow L2$ \rightarrow L3$

ProbeAck

ProbeAck

ProbeAck

ProbeAck
Recursive Acquire
Recursive Acquire

L1$  
L2$  
L3$  
L1$  
L2$  
L3$  
L1$  
L2$  
L3$  

ProbeAck  
ProbeAck  
ProbeAck
Recursive Acquire

[Diagram of recursive acquire process with nodes labeled L1$, L2$, and L3$ connected by arrows labeled ProbeAck and Grant]
Recursive Acquire

L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant
L1$ -> L2$ -> L3$ -> Grant

ProbeAck
GrantAck
Recursive Acquire
Recursive Acquire
An open standard Cache Coherent Fabric Interface repository

- 15 commits
- 1 branch
- 0 packages
- 0 releases
- 1 contributor
- Apache-2.0

Branch: master

- dejavuicinic freezing the WDC version of the repository, moved to CHIPS Alliance
- implementations: Updated data and control planes for 0.1.2 protocol
- specification: Added PDF version of spec to make github link work
- LICENSE.Aprache2: Added license file
- README.md: freezing the WDC version of the repository, moved to CHIPS Alliance

Development of OmniXtend has moved to CHIPS Alliance
OmniXtend version 1.0.3 released!

OmniXtend is a fully open networking protocol for exchanging coherence messages directly with processor caches, memory controllers and various accelerators.

OmniXtend is the most efficient way of attaching new accelerators, storage and memory devices to RISC-V SoCs.
Getting started

The pre-built binaries are available in this section.

Pre-requisites

```bash
# cat /proc/cpuinfo | grep hart | wc -l
```
dejan-vucinic added 1.03 draft spec

- OmniXtend-spec-1.0.3-draft.pdf
- StateTransitionTables-1.8.0.pdf
- TileLink-1.8.0.pdf

Latest commit 4eebc18 11 minutes ago

11 minutes ago

3 days ago

3 days ago
3.1 TLoE frame header

Figure 9 shows the encoding of the TLoE frame header.

<table>
<thead>
<tr>
<th>Byte 7</th>
<th>Byte 6</th>
<th>Byte 5</th>
<th>Byte 4</th>
<th>Byte 3</th>
<th>Byte 2</th>
<th>Byte 1</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC (8)</td>
<td>Res (7)</td>
<td>Sequence_number (22)</td>
<td>Sequence_number_ack (22)</td>
<td>R (1)</td>
<td>Chan (5)</td>
<td>Credit (5)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9 – TLoE frame header encoding

The Virtual Channel (VC) field allows the use of multiple instances of the TLoE protocol between two TLoE endpoints.

The Sequence_number/Sequence_number_ack/Ack fields are used for retransmission, and the Credit/Chan fields are used for flow control.

The Sequence_number field carries an incrementing sequence number and is used by the TLoE receiver to detect packets that are received out of order.

The Sequence_number_ack field is used by TLoE receiver to indicate (back to the TLoE transmitter) the sequence number of the last packet that was received in correct sequence order.

The Ack field is used by the TLoE receiver to indicate (back to the TLoE transmitter) a positive acknowledge (Ack=1) or a negative acknowledge (Ack=0):

- Ack=1 (ACK) indicates that the receiver has received packets in correct sequence order up to Sequence_number_ack.
- Ack=0 (NAK) indicates that the receiver has received packets with incorrect sequence order, and the transmitter should retransmit packets that are missing.
Figure 10—TileLink message formatting

Figure 11 through Figure 14 show the detailed encoding of the TileLink messages, and Table 1 shows the association among each TLoE field and the different TileLink channel signals.
OmniXtend Specification, version 1.0.3-draft.

Figure 11 — TileLink message encoding for channels A, B and C

Figure 12 — TileLink message encoding for channel D — AccessAck, ReleaseAck, HintAck and AccessAckData messages

Figure 13 — TileLink message encoding for channel D — Grant and GrantData messages
chipsalliance / omnixtend

Branch: master  omnixtend / OmniXtend-1.0.3 / spec /

dejan-vucinic added 1.0.3 draft spec

..

- OmniXtend-spec-1.0.3-draft.pdf  added 1.0.3 draft spec  11 minutes ago
- StateTransitionTables-1.8.0.pdf  StateTransitionTables-1.8.0.pdf  3 days ago
- TileLink-1.8.0.pdf  new directory structure  3 days ago
### Table 3: State transitions for node receiving AcquireBlockB

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>↓AcquireBlockB</td>
<td>Idle</td>
<td>aqb1</td>
<td>TT, TB</td>
<td></td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>aqb2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>aqb3</td>
<td>T</td>
<td>C,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>aqb1</td>
<td>TT, TB</td>
<td></td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>aqb4</td>
<td>T</td>
<td>T</td>
<td>C,D</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>aqb5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑GrantDataT</td>
<td>aqb1</td>
<td>aqb4</td>
<td>TT</td>
<td>T</td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aqb8</td>
<td>aqb5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑GrantDataB</td>
<td>aqb1</td>
<td>aqb4</td>
<td>TT, TB</td>
<td>TB</td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aqb8</td>
<td>aqb5</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑ProbeBlockB</td>
<td>aqb2</td>
<td>aqb6</td>
<td>T</td>
<td>C,D</td>
<td></td>
<td>7,18</td>
</tr>
<tr>
<td>↓ProbeAck</td>
<td>aqb6</td>
<td>aqb1</td>
<td>T</td>
<td>TB</td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TT</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>↓ProbeAckData</td>
<td>aqb6</td>
<td>aqb1</td>
<td>T</td>
<td>TB</td>
<td>C,D</td>
<td>D</td>
</tr>
<tr>
<td>↓AcquireBlockB</td>
<td>aqb3</td>
<td>aqb7</td>
<td>T</td>
<td>TB</td>
<td>C,D</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑GrantDataT</td>
<td>aqb7</td>
<td>aqb8</td>
<td>N</td>
<td>TT</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>↑GrantDataB</td>
<td>aqb7</td>
<td>aqb8</td>
<td>N</td>
<td>B</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>↓GrantAck</td>
<td>aqb4</td>
<td>Idle</td>
<td>TB, T</td>
<td></td>
<td>C,D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aqb5</td>
<td>aqb9</td>
<td>T</td>
<td>C,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↓GrantAck</td>
<td>aqb8</td>
<td>aqb1</td>
<td>T</td>
<td>C,D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
omnixtend / OmniXtend-1.0.3

- **dejan-vucinic** added README.md for future release
  Latest commit 8c3f32c 29 seconds ago

  - ..
  - **emu** added README.md for future release
  - **impl** new directory structure
    29 seconds ago
  - **sim** Update READMEs of OmniXtend simulation and emulation
    3 days ago
  - **spec** added 1.0.3 draft spec
    21 minutes ago
This emulator aims to provide a sample implementation in C of the TileLink over Ethernet (TLoE) memory target. The software memory target and the FPGA-based memory target can be used interchangeably. The emulator uses the Data Plane Development Kit (DPDK) which is a set of data plane libraries and network interface controller drivers for fast packet processing.

Prerequisites

To run the emulator, a network interface card which supports DPDK is required. The list of supported hardware can be found [here](https://dpdk.org). We used a breakout cable to connect a QSFP port of the VCU118 FPGA to a port of the NIC.

Instructions to compile and to run the emulator

Install DPDK Dependency

Users can follow instructions to build DPDK at: [http://doc.dpdk.org/guides/linux_gsg/index.html](http://doc.dpdk.org/guides/linux_gsg/index.html) or run the script

```
./install_dpdk.sh
```

Build TLoE memory emulator

Make sure RTE_SDK and RTE_TARGET environment variables have been set. Then run `make` to compile the code.
OmniXtend Architectures

Point to Point

Through an Ethernet Switch

Program switch to replace ethernet protocol with OmniXtend protocol to handle memory coherency traffic

ML Accelerator