OmniXtend Remote Agent – OmniXtend Based Remote Code Execution over Ethernet

December 06, 2022
Revision 1.0

PO Box 9276
San Jose, CA 95157
www.lewiz.com
Email: support@lewiz.com

Change Log

Version	Date	Description
v.1.0	2022-12-06	Document created

Table of Contents

1. Introduction

This OmniXtend RemoteAgent (OXRemoteAgent) project is developed to use the TileLink over Ethernet protocol (TLoE, a.k.a OmniXtend) as a communication bridge between the processor(s) (e.g., RISC-V CPU) and a network storage endpoint containing program instructions. (See [OXTEND] and [TLINK] references). It allows the processor(s) to fetch program instructions and data from the endpoint as if the information is contained in its local memory. The protocol enables remote execution with low latency but still maintains data coherency across different levels of CPU memory hierarchy (i.e., L1/L1.5/L2 caches, external memory and so on). This protocol is useful in large scale clustering of processors (i.e. servers) and remote storage or memory systems.

In this project, LeWiz developed an IP core (OX_CORE) with complete test bench, documentation, etc. and released them in open source under Apache 2.0 license. An implementation specific with RISC-V processor and OX_CORE has also been developed and targeted for Xilinx VCU118 UltraScale+ FPGA board. This implementation uses a combination of the open source OpenPiton Framework and Ariane 64-bit RISC-V CPU core. An OmniXtend endpoint (see Figure 1) has also been implemented and targeted for the Alveo U50 board from Xilinx. The endpoint, in this case, acts as remote memory containing the RISC-V CPU program(s). The RISC-V CPU fetches instructions from the endpoint and executes the instructions as if the code were contained in its local memory.

The overall design is shown in Figure 1. It contains the OX_CORE, LeWiz's 100/50/40/25/10Gbps Ethernet [LMAC3], and Xilinx 10Gbps PHY. All modules to the left of the network in this diagram are implemented on Xilinx's VCU118 Virtex UltraScale+ FPGA board. The endpoint where the remote program is stored is implemented on Xilinx's Alveo U50 accelerator board. The U50 board has large High Bandwidth Memory (HBM) which are used as remote storage. In this design example, the CPU on the VCU118 board fetches a "Hello World" program from the U50 board over the network, executes it, and displays the output on a UART serial terminal. An overview of the hardware topology is shown in Figure 2.

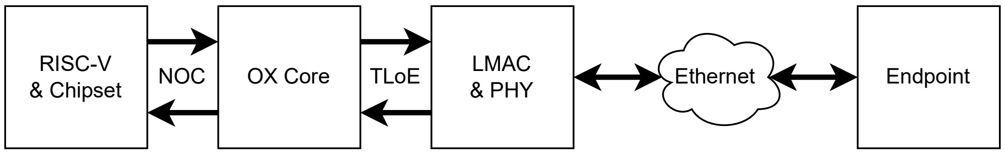

Figure 1: Design Overview

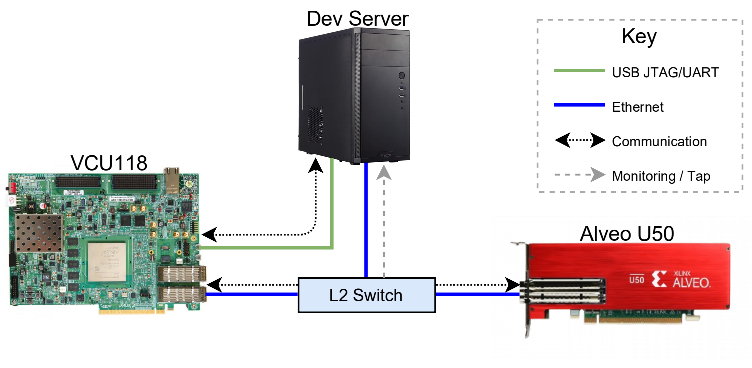

Figure 2: Hardware Configuration Overview

2. Overview

A detailed overview of this design is shown in Figure 3. Here, the CPU acts as an initiator, fetching a “hello world” program from the endpoint through the OX_Bridge. The OX_Bridge is connected to the CPU’s Chipset via a network on chip interface (NoC Interface) and the endpoint via Ethernet, respectively. The OX_Bridge contains an OX_Core, LeWiz’s 100Gbps Ethernet LMAC Core 3 (refer to LeWiz’s LMAC3 for detail), and a PHY. Inside the OX_Core, it has a TX path and an RX path. Each path is responsible for transmit and receive, respectively. In the transmit (TX) path, the OX Core converts the NoC commands from the CPU into a TileLink message, and then into TileLink over Ethernet (TloE or OX) message before delivering to the LMAC. The receive (RX) path performs the reversed order of these operation. In Figure 3, the area where an ILA is pointed at are the locations where signals are probed and used for debugging purposes.

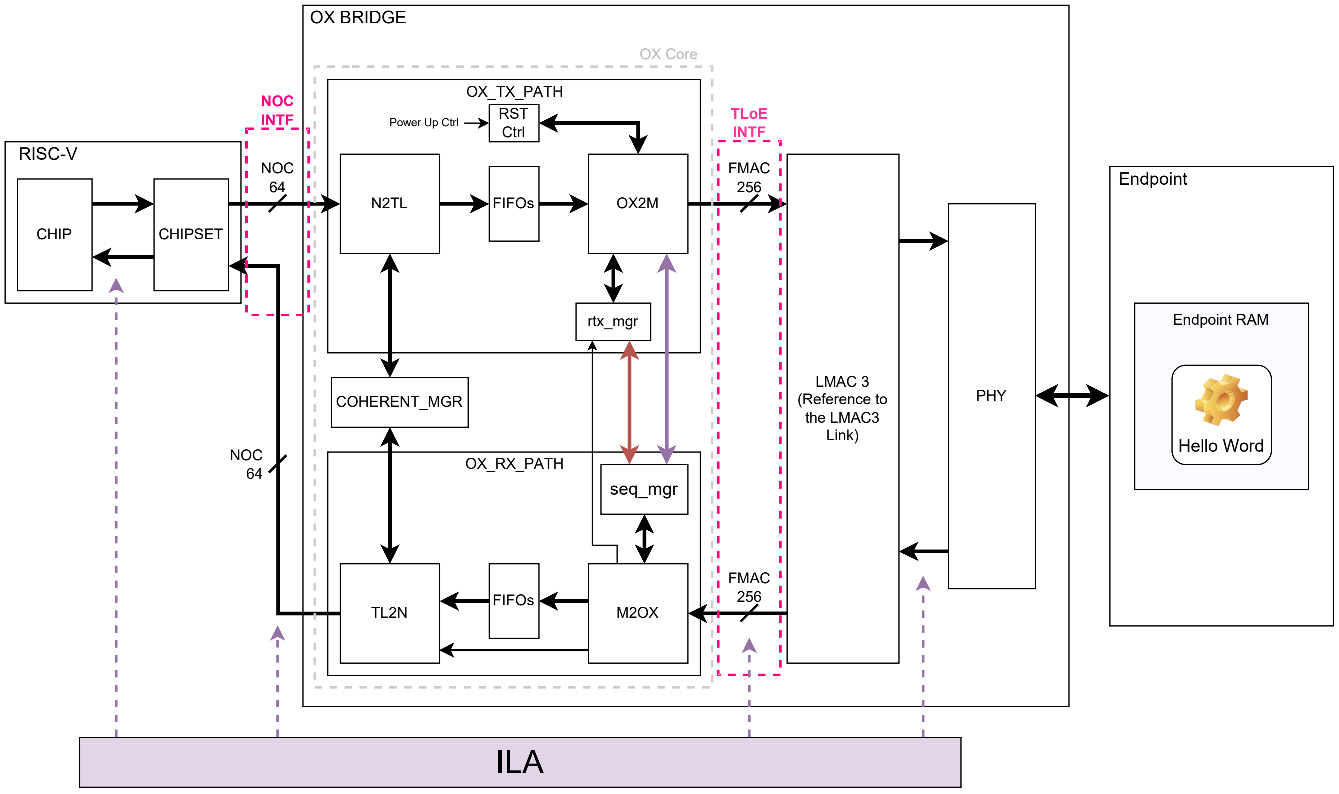

Figure 3: Internal Design of OX Bridge

3. Directory Map

3.1 `OX_CORE_INFO`

This directory contains the OmniXtend Core and testbench along with supporting modules and scripts. Notable contents include the OX Core and LMAC source code, OX Core testbench and OX Core hardware validation module.

`code`: All source and simulation code and scripts are located in this directory.

`code/src`: Contains synthesizable design source code for all modules.

> **`src/ox_core`:** Contains all OX Core source code.
>
> **`src/lmac`:** Contains all LMAC Core 3 source code.
>
> **`src/endpoint`:** OmniXtend Endpoint source code. Contents pending.
>
> **`src/endpoint/memoryconfig.hex`:** Hexadecimal format memory file containing data to be loaded onto the endpoint.
>
> **`src/fifo`:** Contains variable depth FIFO wrappers and an RTL asynchronous FIFO.
>
> **`src/utility`:** Contains source code for assorted utility modules.
>
> **`src/fpga`:** Contains FPGA level modules and supporting components.
>
> **`src/global.vh`:** Global compiler directive listing file.

`code/sim`: Contains source code for testbenches and simulation-only modules along with scripts and supporting data files.

> **`sim/core_tb`:** Contains OX Core testbench, simulation scripts and waveform configurations.
>
> **`sim/core_fpga`:** Contains simulation script and waveform configurations for **`core_fpga.`**
>
> **`sim/utility`:** Contains source code for assorted simulation-only utility modules.

`code/ip`: Contains IP descriptor files (`.xci`) and Block ROM IP initialization files.

`code/cons`: Contains project constraint files for synthesis and implementation.

`docs`: Contains supplemental documentation pertaining to the lower level details or testing of the OmniXtend Core.

`proj`: Contains Vivado generated project files. Initially empty.

`setup.tcl`: Vivado project setup script. Source script using Vivado to generate project

`update_ip.tcl`: IP configuration update script. To be run after any IPs are added to the project.

3.2 **FPGA/VCU118_FPGA_INFO**

This directory contains all files related to the OmniXtend RemoteAgent CPU, for implementation on a VCU118 FPGA board. Notable contents include the OX Core and LMAC source code, OpenPiton Framework with Ariane CPU core and build/synthesis utilities.

`ox_agent/build`: Build product output folder, initially empty.

`ox_agent/piton/design`: Design source code. Includes OX Core, LMAC, Endpoint (for simulations), OpenPiton chipset and Ariane CPU core. This directory and its sub-directories generally follow the layout presented below.

> **`rtl`:** RTL code pertaining to this level of the design. Contains an 'Flist' file listing each source file in the directory. Source files ending in **`.pyv`** are pre-processed to generate **`.tmp.v`** files.
>
> **`xilinx`:** Implementation files specific to Xilinx boards and FPGAs. May include build scripts, board configurations, constraints or IPs. Board specific files are contained in a sub-directory with the board's name.
>
> **<Other Sub-directories>:** Lower level components of the design

Notable `ox_agent/piton/design` sub-directories:

> **`rtl`:** Contains top-level system module that instantiates CPU core, chipset and OmniXtend Bridge.
>
> **`xilinx`:** Contains constraint files for all supported boards.
>
> **`oxbridge`:** Contains all LeWiz OmniXtend related source files and supporting IPs. Includes OX Bridge, OX Core, LMAC and PHY.
>
> **`oxbridge/ox_core`:** Contains all OX Core source code.
>
> **`oxbridge/lmac`:** Contains all LMAC Core 3 source code.
>
> **`oxbridge/endpoint`:** Contains endpoint source code for use in simulation.
>
> **`oxbridge/utility`:** Contains utility and support modules for the OX Bridge.
>
> **`chip`:** Contains CPU core, L1/L1.5 caches and supporting modules
>
> **`chipset`:** Contains OpenPiton chipset and related interface modules including memory controllers, SD card bridge and UART controller.

`ox_agent/piton/tools`: OpenPiton build, simulation and synthesis tools.

`ox_agent/piton/verif`: OpenPiton verification files.

`ox_agent/piton/verif/diag/c/riscv/ariane/hello_world_uart.c`: Source code for 'Hello World' program used to be stored on the Endpoint for execution.

`example/ox_agent.bit`: Pre-made bitstream of OpenPiton-based OX Agent.

`example/ox_agent.ltx`: Debug probe data for above bitstream.

`example/hello_uart.ila`: Example ILA capture of program execution showing primarily NoC requests and responses.

`example/hello_uart_zoom.ila`: Example ILA capture of program execution showing NoC, TLoE and CGMII stages in both directions for two NoC requests/responses.

`example/hello_uart.pcapng`: Example network capture showing traffic between OX Agent and Endpoint.

3.2 `FPGA/U50_FPGA_INFO`

This directory contains all files related to the OmniXtend Endpoint, for implementation on an Alveo U50 accelerator card. Currently empty. Pending release of OmniXtend Endpoint.

4. Design Hierarchy

4.1 OmniXtend Core

OX_CORE: A wrapper that contains separate RX and TX paths, and cache coherency manager.

- **OX_TX_PATH:** A wrapper to encapsulate all transmit direction modules.
  - **N2TL:** Receives NoC commands from the processor and translates them into TileLink (TL) messages. Each message is divided into header, address, mask, data, and byte count signals.
  - **FIFOs:** A set of FIFOs that store header, address, mask, data, and byte count for OX2M to read when ready. Header, address, and mask are all 64-bit wide. Data FIFO is 256-bit wide and byte count FIFO is 16-bit wide. While every message has header and address, not all messages have mask, data and byte count. So it is recommended to use header's full signal to determine overall FIFO status.
  - **OX2M:** This module reads data from the above FIFOs and wraps them into TLoE frames with the proper endianness for the endpoint to receive. This module also determines the correct channel to send on based on the command. This outputs a TLoE message into a FIFO and byte count into another FIFO.
  - **RTX_MGR:** Manages retransmission of stored outing frames. Triggered by reception of NAK by RX path.
  - **rtx_buf:** Stores outgoing frames for retransmission under control of RTX_MGR.
  - **RST_CTRL:** Controls the reset timing.
  - **PWRUP_CTRL:** Initiates the reset sequence and TLoE credit advertisement upon powering up.
- **OX_RX_PATH:** A wrapper to encapsulate all the receiving direction modules.
  - **M2OX:** The counterpart to the OX2M module, it reverses the process performed in the TX path. The M2OX translates the received TLoE frames into TL messages with the proper endianness. It also determines the proper channel and returning address, directing the received packet to the correct CPU tile.
  - **FIFOs:** Equivalent to the FIFOs in the transmit path. Stores TileLink messages for reading by TL2N. Each message is divided into header, address, mask, data, and byte count.
  - **TL2N:** The counterpart of N2TL in the transmit path. Translates TileLink messages from FIFOs to NoC commands for the processor.
  - **SEQ_MGR:** Tracks local and remote sequence numbers and verified order of incoming frames. Also provides grant signals to other modules for acknowledgment, retransmission and normal transmission packets.

- **COHERENT_MGR:** Handles cache coherency for requests such as acquire, probe, and release. (NOT used in this version)
    - **N2TL_AQSM:** A state machine driven control signal module that receives acquire signals and provides grant signals alongside with coherent manager. This handles acquire requests. (NOT used in this version)
    - **N2TL_PRBSM:** A state machine driven control signal module that receives probe signals and provides grant signals alongside with coherency manager. This handles probe requests. (NOT used in this version)
    - **N2TL_RLSSM:** A state machine driven control signal module that receives release signals and provides grant signals working alongside with coherency manager. This handles release requests. (NOT used in this version)

4.2 OX Core Testbench

core_tb: OX Core testbench

- **NOC_MASTER:** File-based NoC Master Emulator. Generates NoC commands from file.
- **OX_CORE:** See Section 4.1
- **NETE_MASTER:** Emulates LMAC and all network components up to Endpoint.
- **OmnixtendEndpoint:** OX Endpoint with AXIS interface.
- Testbench logic allows automatic control of the NoC Master and printing of NoC commands and TileLink messages.

4.4 OX Core FPGA Implementation

core_tb: OX Core FPGA level module

- **brom_noc:** A Block ROM containing NoC commands to be issued to the OX Core.
- **axis_mastersim_mem:** Repurposed as a NoC Master. Sends commands from above ROM.
- **axis_delay:** Causes a configurable delay between NoC commands by gating NoC bus `valid` and `ready` signals.
- **OX_CORE:** See Section 4.1
- **LMAC:** Refer to LeWiz's LMAC Core 3 documentation for more details.
- **PHY:** Physical layer connection between LMAC and network switch with the endpoint.
- An Integrated Logic Analyzer (ILA) allows monitoring of NoC, MAC and CGMII signals. Virtual Input Output modules (VIOs) allow remote resetting and NoC command delay configuration.

Note: A set of RTL FIFOs are used for simulation and a different set of IP FIFOs are used for implementation. Appropriate FIFOs are selected automatically

4.4 OX Agent System

system: The top module for this project where `chip`, `chipset`, and `oxbridge` are instantiated.

- **chip:** Contains the CPU core(s) and L1.5/L2 caches.
- **chipset:** The bridge module connecting the CPU to the ox bridge and other peripherals.
- **oxbridge**: Contains the OX Core, LMAC, and PHY. It connects to to the CPU via NoC interface and endpoint by Ethernet.
  - **OX_CORE:** See Section 4.1
  - **LMAC:** Refer to LeWiz’s LMAC Core 3 documentation for more details.
  - **PHY:** Physical layer connection between LMAC and network switch with the endpoint.

Note: A set of RTL FIFOs are used for simulation and a different set of IP FIFOs are used for implementation. Appropriate FIFOs are selected automatically

5. OmniXtend Core Simulation & Validation

The OmniXtend Core simulation and validation project may be opened in Vivado by sourcing the TCL script `setup.tcl` within `OX_CORE_INFO`. A Vivado project will be generated and opened automatically. Currently, only Vivado version 2021.2.1 is supported. Other versions may work, but success is not guaranteed.

5.1 Simulations

Primary Testbench '`core_tb`'

> This testbench issues a select-able, preset sequence of NoC commands to the OX Core. NoC responses from the OX Core are verified manually. By default, NoC commands are issued automatically. This may be disabled to allow finer control of commands via TCL script. Please note, a \`run all\` TCL command may need to be issued to ensure the bench runs to completion.

FGPA Level Module '`core_fpga`'

Module `core_fpga` may be set as simulation top, and script `code/sim/core_fpga/core_fpga.tcl` used to test the basic functionality of the module prior to synthesis. The system is reset in the appropriate sequence and NoC commands are automatically issued to the OX Core. Resuting CGMII TX data is written to `code/src/fpga/core_fpga.TX.mem`. The RX path is fed with CGMII RX data read from `code/src/fpga/core_fpga.RX.mem`.

5.2 Hardware Validation Module

FGPA Level Module '`core_fpga`' is the intended top level module for synthesis. Once programmed onto a VCU118 FPGA board, Vivado's hardware manager may be used to view the ILA connected to NoC, MAC and CGMII signals. The VIOs may be used to control remote reset and enable, and to configure the NoC command delay. The onboard GPIO buttons may also be used for reset and enable.

The FPGA may be connected to either a real OmniXtend Endpoint or a stand-in. Suggested hardware configuration matches that shown in Figure 2. If the OX Core does not receive any TileLink responses, only eight NoC commands will be translated and transmitted. TileLink responses containing data will be translated to NoC commands, visible using the ILA.

6. Using the Complete OX Agent System

The following procedure has been tested on Ubuntu Server 18.4. Two separate machines were used for the VCU118 and Alveo U50. It may be possible to use a single machine for both, but this is untested. An implementation using only a single VCU 118 is being considered. Vivado is assumed to be preinstalled on both machines.

6.1 Hardware Configuration / Environment

Hardware configuration is as shown in the following figure. The VCU118 and Alveo U50 are connected to a network switch via their QSFP ports. The development server containing the U50 is connected to the switch via Ethernet to allow loading data onto the Endpoint. A mirroring port on the switch is used to allow the other development server to monitor network traffic. This machine is also connected to the VCU118 via USB for programming, debugging and UART serial terminal access.

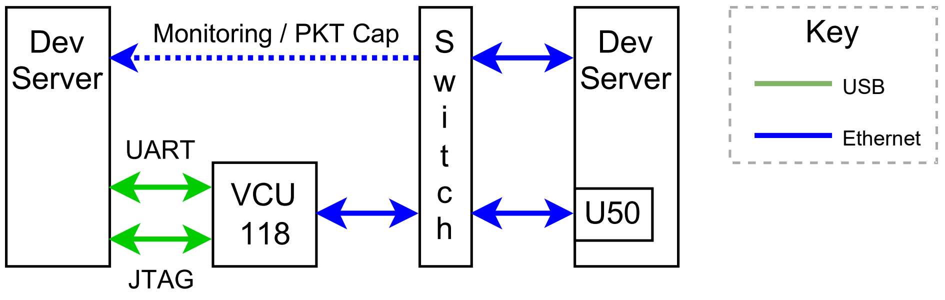

Figure 4: Hardware Configuration

6.2 OmniXtend RemoteAgent (VCU118)

Perform these steps on the machine used for programming the VCU118.

6.2.1 Initial Setup

1. Clone the OmniXtend RemoteAgent repo
```
$ git clone https://github.com/lewiz-support/OmniXtend_RemoteAgent_RISC-V.git
$ cd OmniXtend_RemoteAgent_RISC-V
$ git submodule update --init
```

2. Ensure dependencies are installed
```
$ sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool
patchutils bc zlib1g-dev libexpat-dev device-tree-compiler tcsh
```

3. Ensure directory "`/scratch`" exists and is accessible to all
```
$ sudo mkdir -m777 /scratch/
```

4. Set Temporary OpenPiton Environment Vars
```
$ cd VCU118_FPGA_INFO/ox_agent/
$ source piton/ariane_setup.sh
$ source piton/piton_settings.bash
```

5. Build CPU & related utilities (pulls CPU core and build utils from respective repos)
```
$ piton/ariane_build_tools.sh
```

```
#WARNING: Building is somewhat failure prone. For easier error diagnosis, run some
of the steps manually:
  $ git submodule update --init --recursive piton/design/chip/tile/ariane
  $ export NUM_JOBS=4
  $ cd piton/design/chip/tile/ariane/
  $ ci/make-tmp.sh
  $ ci/build-riscv-gcc.sh
  $ ci/install-fesvr.sh
  $ ci/install-spike.sh
  $ ci/install-verilator.sh
  $ cd $PITON_ROOT
  $ piton/ariane_build_tools.sh
```

6.2.2 Simulating Design

~Pending release of OmniXtend Endpoint~

6.2.3 Synthesizing Design

1. Set Temporary OpenPiton Environment Vars
```
$  cd VCU118_FPGA_INFO/ox_agent/
$  source piton/ariane_setup.sh
$  source piton/piton_settings.bash
```
2. Set Temporary Vivado Environment Vars
```
$ source <Vivado Install Path>/settings64.sh
```
3. Run Synthesis, Implementation & Bitstream Generation. Generates a Vivado project
```
$ cd $PITON_ROOT/build
$ protosyn -b vcu118 -c ariane --no-ddr --bram-test hello_world_uart.c --define
OXAGENT_EN,ILA_ENABLE --skip-test

#If protosyn fails with errors, the project may be opened with Vivado to assist in
diagnosing and resolving errors.
```
4. Convert compiled program instructions endianness for compatibility with Endpoint
```
$ python3 ../../scripts/byte_flip.py -z 134512 -o <output file> diag.o

#The "-z" argument pads the end of file with zeros to clear a portion of endpoint
 memory for program use.
```

6.2.4 Adding New Source Files (Optional)

For Simulation: For source files to be used by a simulation, they must be listed in an Flist file, which is in turn listed in the simulation's configuration file located in the directory `piton/tools/src/sims/`. For an example, refer to the following files.

- `piton/design/oxbridge/rtl/oxbridge.v` – Source file
- `piton/design/oxbridge/rtl/Flist.oxbridge` – Flist File
- `piton/tools/src/sims/manycore.config` – Simulation Configuration

For Synthesis: For source files and IPs to be used in a synthesis, they must be listed in the file `piton/tools/src/proto/common/rtl_setup.tcl`. Listed files are grouped by component and category as described below.

- Components
    - `SYSTEM`: Overall system sources. Always in use.
    - `CHIP`: CPU Core and Cache sources. Only used when CPU is present in design.
    - `PASSTHRU`: Pass-through module sources. Only used when pass-through is enabled.
    - `CHIPSET`: CPU chipset sources. Only used when chipset is present in design.
- Categories
    - `RTL_IMPL_FILES`: List of all RTL source files used by the component
    - `INCLUDE_FILES`: List of all include files used by the component
    - `IP_FILE_PREFIXES`: List of all IPs used by the component
    - `COE_IP_FILES`: List of all coefficient files used by IPs in the component
    - `PRJ_IP_FILES`: List of all `.prj` files used by IPs in the component

6.3 OmniXtend Endpoint (Alveo U50)

~Pending release of OmniXtend Endpoint~

6.4 System Operation

1. Ensure hardware devices are connected as described in Section 6.1.
2. Synthesize OX Agent design for VCU118 & program board using Vivado (see Section 6.2).
3. Program and configure Endpoint (see Section 6.3).
4. Connect to UART serial terminal from VCU118 programming machine.
```
$ ls /dev/ttyUSB*
$ sudo screen <device> 115200

#The desired device is typically the last of those listed by the 'ls' command.
```
5. Start any ILA or network capture as desired.
6. Reset the OX Agent system using either button 'SW5: CPU_RESET' on the VCU118 board, or HW VIO signal "reset_all_" through Vivado.
7. To rerun the program, reset connections on the endpoint before resetting the OX Agent system.
```
#NOTE: The program data on the Endpoint may be changed/reloaded without having to re-
program any other system components. Simply repeat this step as many times as
desired.
```

7. References

[LMAC3] LMAC Core, LeWiz Communications, Inc., https://github.com/lewiz-support/LMAC_CORE3/blob/master/DOCS/LMAC_CORE3_UserManual_rev1p03_OSrc_24may18.pdf

[TLINK] TileLink Specification Rev 1.8.0, SiFive Inc., August 9, 2019

[OXTEND] OmniXtend Specification Rev 1.0.3, SiFive Inc., Western Digital Corp, November 12, 2019