USS

1 Cyclotron Road
Berkeley, CA, 94720

Tutorial for including an accelerator in the USS

August 04, 2022

Tile shell

The tile shell, shown in Figure 1, offers two interfaces to communicate with a custom accelerator:

- A lightweight NoC consisting of 5 4X1 switches
- An AXI interface.

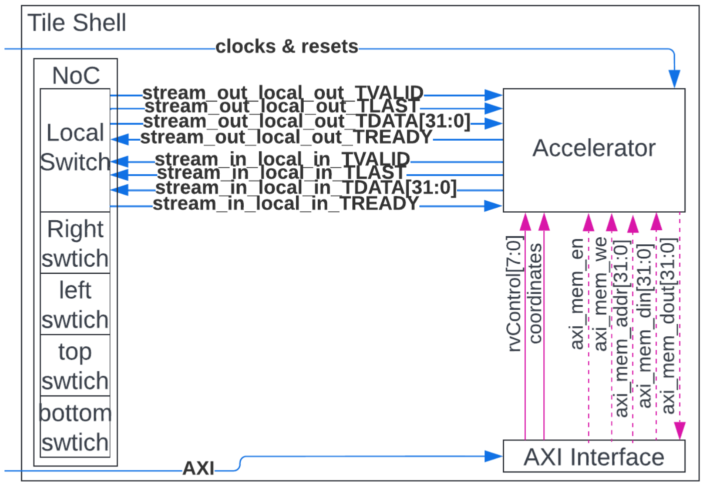


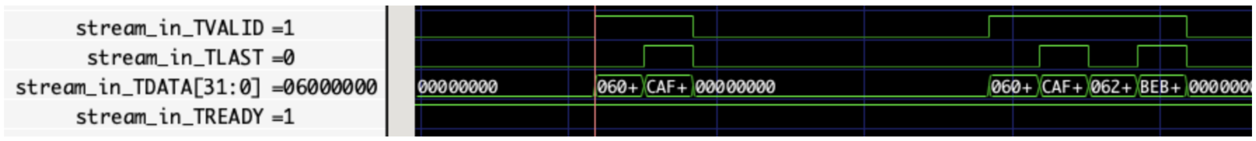
Figure 1. Tile Shell

Network on Chip (NoC)

The AXI-lite-inspired, NoC interface comprises the following signals:

- TVALID: Asserted by the source when valid information is available. It must remain asserted until the destination accepts the data and until the last word of the packet is sent (TLAST is asserted).
- TLAST: Indicates the last word of the packet.
- TDATA: Packet information.
- TREADY: Set by the destination, it indicates when the destination can receive information.

The timing diagram for the NoC signals is shown below:



For short packets, the packet is composed of two words, header and data. The diagram shows an isolated packet followed by two consecutive short packets.

AXI interface

A typical AXI interface has five channels: *Write Address* (AW), *Write Data* (WD), *Write Response* (B), *Read Address* (RA), and *Read Data* (RD).

The interface enables the communication between a manager and a subordinate. In this case, the manager is a host computer that initiates the AXI transaction, and the subordinates are the tiles. Below is a short description of the AXI interface taken from [1].

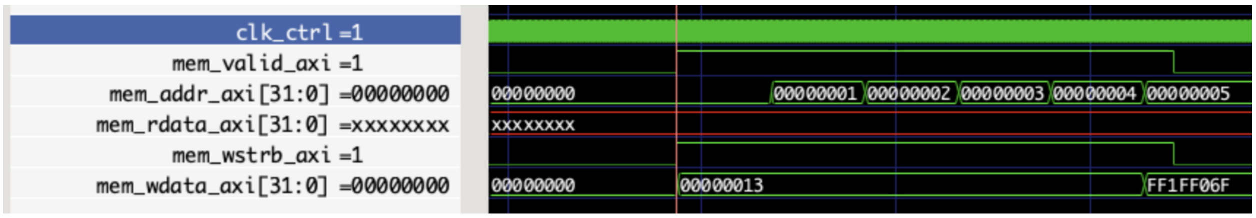
For write operations, the manager issue an address on the Write Address (AW) channel and transfers data on the Write Data (W) channel. The subordinate writes the data to the specified address. Once the subordinate has completed the write operation, it responds with a message to the sender on the Write Response (B) channel.

For reading operations, the manager sends the address it wants to read on the Read Address (AR) channel. The subordinate sends the data from the requested address to the manager on the Read Data (R) channel.

The tile shell provides the following AXI registers:

- An 8-bits configuration register *rvControl.* This register can be used to fit the accelerator configuration needs.
- Registers for writing/reading to/from a scratchpad in the accelerator.
- A coordinates register *tileID*, which is written before runtime. It can be used by the accelerator to form the packet header.

The figure below shows the memory interface, controlled by the AXI bus, used to initialize the scratchpad in the accelerator.



In this case, we write from addresses 0-4 the h’13 as the data and to address 5 h’FF1FF06F. This is the code for a NOP loop for the picorv32.

Example: 2x2 USS

Figure 2 shows a 2x2 USS system as a dummy example of a heterogeneous tiled system. The system has four tiles with coordinates (0,0)-(0,1)-(1,0)-(1,1). Tiles (0,0) and (1,1) at the corners have a PICORV32 with an ISA extension for message queues in place of the accelerator. In tile (0,1), the accelerator is a scratchpad (MEM) implemented as a dual port SRAM that receives write and read requests through the NoC interface. The scratchpad can also be accessed through the AXI interface. In Tile (1,0), the accelerator is a loop that receives packets and sends it back to the source without any modification to the header or data.

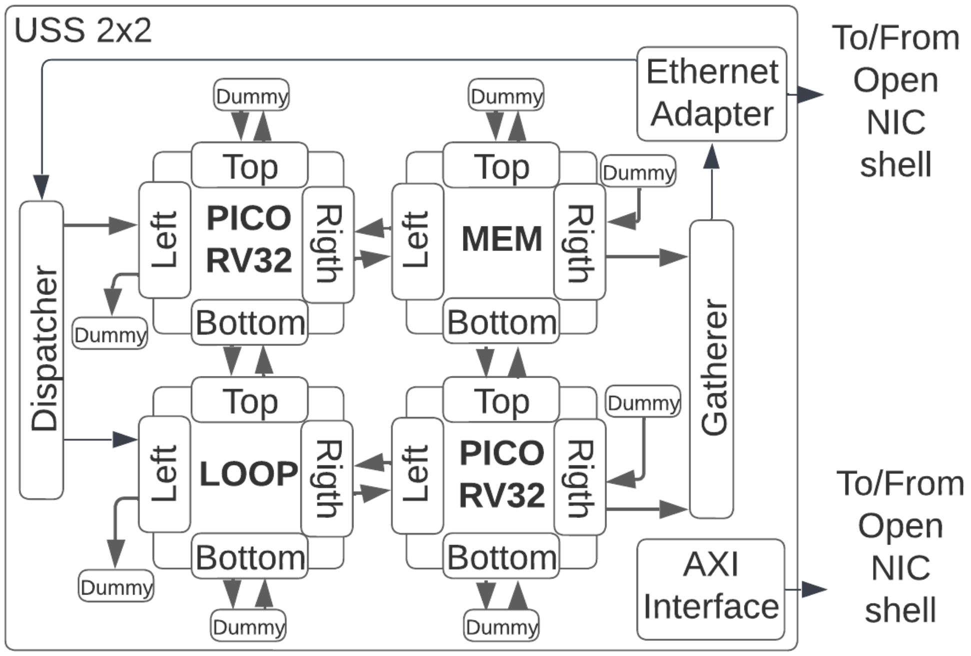
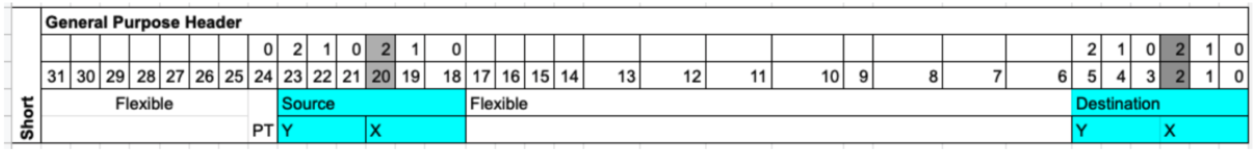


Figure 2. USS 2x2 block diagram. The system has three types of tiles: a tile with a PICORV32, a tile with a scratchpad memory, and a loop tile.

NoC Header

The header is 32 bits wide and is shown below:



	General Purpose Header																															
								0	2	1	0	2	1	0													2	1	0	2	1	0
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Short	Flexible								Source						Flexible												Destination					
								PT	Y			X															Y			X		

The *Source* and *Destination* identifiers are 6 bits wide and are encoded in the form of a (y,x) coordinate. These fields are fixed at bits 23:18 and 5:0 since the NoC uses them to route packets across the mesh. The meaning of the remaining bits can be assigned according to the accelerator's needs.

Accelerator example: Scratchpad Tile (MEM)

Figure 3 shows a simplified block diagram of a tile containing a scratchpad as an accelerator. The accelerator includes a shallow buffer to receive data from the NoC. A decoder (NoC Dec.)

identifies if the packet is a read/write request and starts the memory transaction. An encoder (NoC Enc.) generates response packets when appropriate.

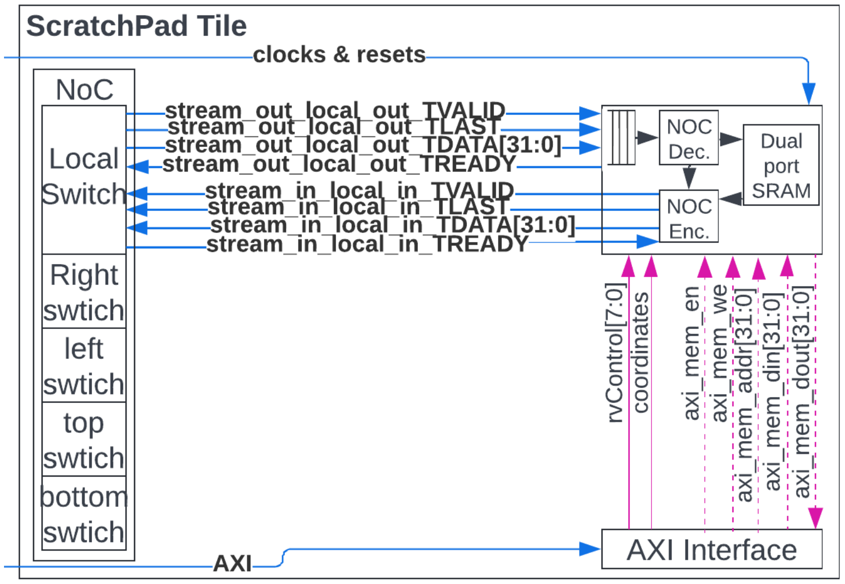
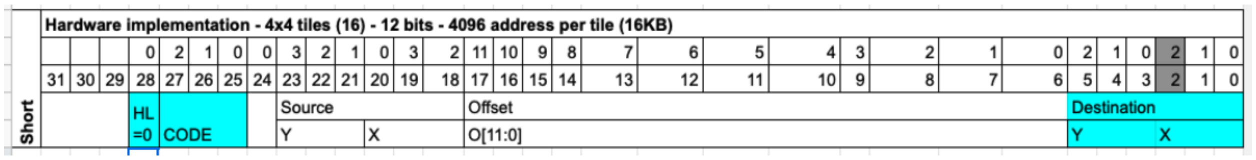


Figure 3. Simplified block diagram of scratchpad as an accelerator.

The header for this accelerator is mapped as follows:



	Hardware implementation - 4x4 tiles (16) - 12 bits - 4096 address per tile (16KB)																															
				0	2	1	0	0	3	2	1	0	3	2	11	10	9	8	7	6	5	4	3	2	1	0	2	1	0	2	1	0
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Short				HL					Source						Offset												Destination					
				=0	CODE				Y			X			O[11:0]												Y			X		

The scratchpad tile offers 16KB of memory, requiring 12 bits of address space <*Offset*>. The CODE field for the incoming packets in the scratchpad tile can take the following values:

- MPUT (4): The sender writes data in the scratchpad memory at address <*Offset*>.
- MGET (5): The sender reads from the scratchpad memory at address <*Offset*>.
- MLOAD (6): The sender reads from the scratchpad memory at address <*Offset*>. The sender is blocked until it receives a response.
- MSTORE (7): The sender writes data in the scratchpad memory at address <*Offset*>. The sender is blocked until this Tile sends an ACK.

In the case of MPUT or MSTORE the payload is data to be written on the scratchpad. For an MGET or an MLOAD, the payload contains the address (offset) to write the response at the origin.

The CODE field for the output packets from the scratchpad memory can take the following values:

- MACK (1): the scratchpad memory sends an ACK in response to an MSTORE packet.
- MDATA (2): the scratchpad memory sends the data in response to an MLOAD packet.
- MPUT (4): The scratchpad memory sends the data in response to an MGET packet.

Testcase

For testing the 2x2 USS system, we inject the following packets through the dispatcher at the left of the array, using the *Packet_in.txt* file.

Header	**Data**	**Code**	**Src.**	**Dest.**	**Offset**	**Description**
06000000	CAFECAFE	MQ	00	00	NA	Push CAFECAFE to message queue at *tile00-picorv32*.
08000181	BACABACA	MPUT	00	01	6	Write BACABACA to *tile01-scratchpad* at address 6.
06000008	BEBEBEBE	MQ	00	10	NA	Push BEBEBEBE to message queue at *tile10-loop*. Will be sent to *tile00-picorv32*.
06000009	DEADEADE	MQ	00	11	NA	Push DEADEADE to message queue at *tile11-picorv32*.
06000000	CAFECAFE	MQ	00	00	NA	Push CAFECAFE to message queue at *tile00-picorv32*.
080001C1	BACABACA	MPUT	00	01	7	Write BACABACA to *tile01-scratchpad* at address 7.
06240008	BEBEBEBE	MQ	11	10	NA	Push BEBEBEBE to message queue at *tile10-loop*. Will be sent to *tile11-picorv32*.

06000009	DEADEADE	MQ	00	11	NA	Push DEADEADE to message queue at *tile11-picorv32*.
0A000001	00000400	MGET	00	01	0	Read address 0 from *tile01-scratchpad and send it to tile00 at address h'400*
0A000041	00000401	MGET				Read address 1 from *tile01-scratchpad and send it to tile00-picorv32 at address h'401*

File Directory

The files that are relevant to add an accelerator are located in the following directories:

- Testbench: Contains the files for testing.
    - USS_tb.v: USS testbench.
- Tile.HDL: This directory contains files common to all the tiles and tiles examples. In particular, it contains
    - Tile_template.v: A template to be modified when adding a new accelerator.
    - Loop_tile: directory with a dummy tile.
        - Tile_loop.v: The tile modified to instantiate the dummy accelerator.
        - Acc_loop.v: The dummy accelerator.
    - Scratchpad_tile: directory with scratchpad accelerator
        - Tile_scratchpad.v: The tile modified to instantiate the scratchpad as an accelerator.
        - Acc_scratchpad.v: The scratchpad as an accelerator.

Other files that make up the system are organized as follows:

- Common.HDL: Contains files that are common to different blocks in the design.
- Dispatcher.HDL: Contains the RTL for the dispatcher. This block distributes injected packets to the first column in the tiled array. In the iverilog, the testbench injects the packets from the *Packet_in.txt* file.
- Gatherer.HDL: Contains the RTL for the gatherer. This block receives packets from the last column on the tiled array and sends them through the USS output interface.

- SOP.HDL: TBD
- S_CONTROLLERs.HDL: Contains the RTL for the AXI protocol decoder and encoder.
- S_PROTOCOL_ADAPTERS.HDL: Contains the RTL that converts the packets from AXI stream to TBD.
- S_RESETTER.HDL: Contains the RTL that generates reset signals for clk_control and clk_line.

Dependencies

- **Iverilog:** *Icarus Verilog,* a Verilog simulation that compiles source code written in Verilog (IEEE-1364) into some target format. The compiler can generate an intermediate form called vvp assembly for batch simulation. This intermediate form is executed by the "vvp" command.
- **vvp:** run time engine that executes the default compiled form generated by Icarus Verilog. The output from the *iverilog* command is not by itself executable on any platform. Instead, the *vvp* program is invoked to execute the generated output file.
- **GTKWave:** GTK based wave viewer for Unix, Win32, and Mac OSX that reads LXT, LXT2, VZT, FST, and GHW files as well as standard Verilog VCD/EVCD files and allows their viewing.

Launch a simulation

Run the bash script:

```
> ./launch_sim
```

Open Issues

- Dispatcher, controllers and gatherer don't support back-pressure.

Outstanding Work

- Adding tutorial to FPGA.

- DRAM support.
- Floating point support.
- RV64 support.
- For loops slow (~64 CC) due to memory accesses.
- DMA.

References

[1] AXI protocol overview.