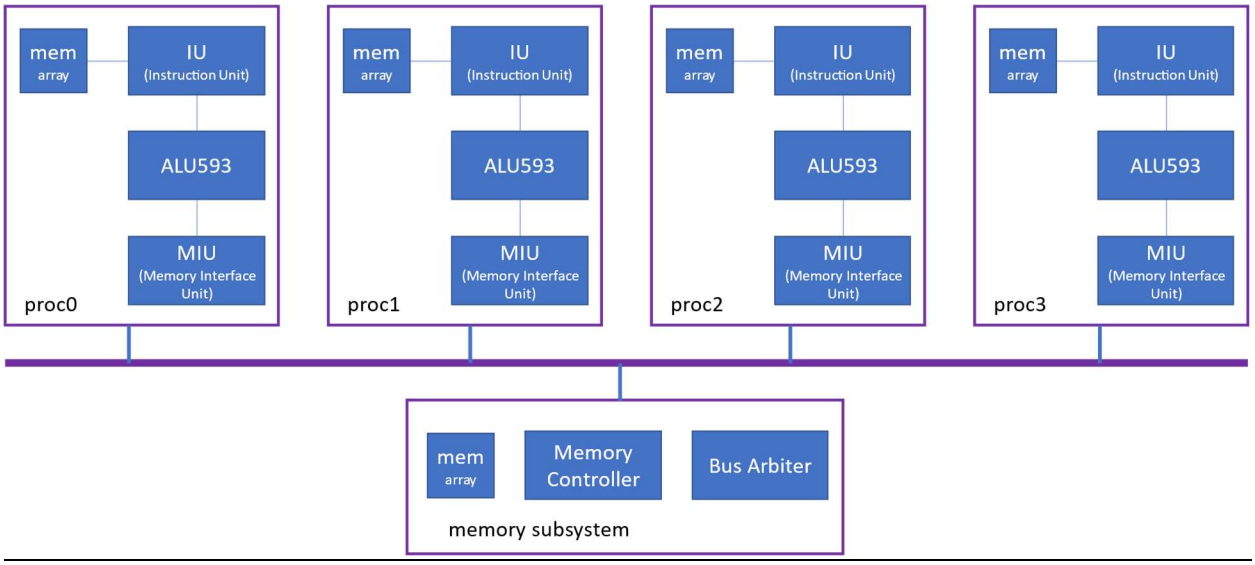
Project ECE593: Specification Rev0.7

This is an example for simplifying multi-processor design



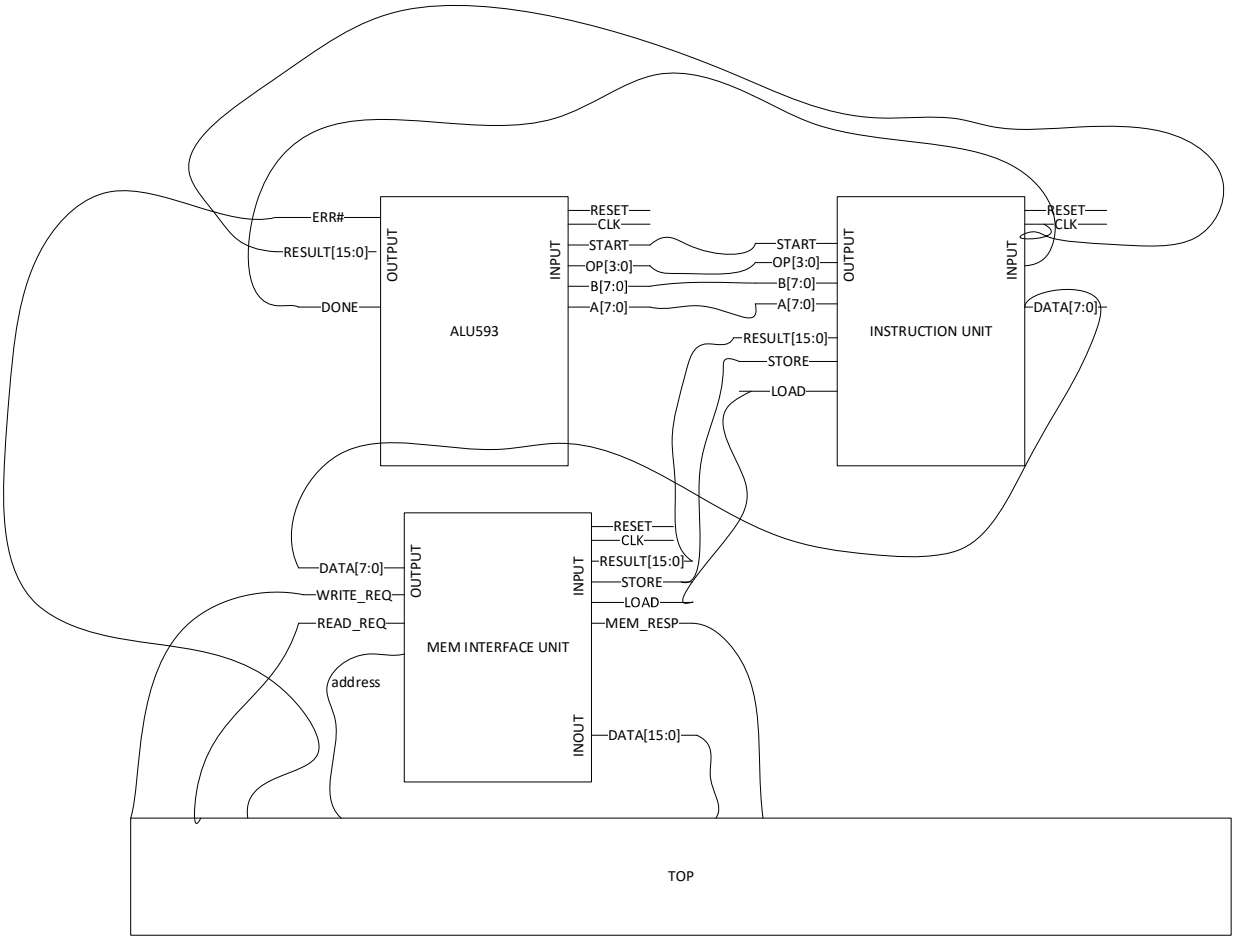
Processor Subsystem Specifications:

- Contains an ALU that executes the following instructions: Processor performs instrtuctions as specified in Assignment #2, table below (except Load and store instructions).
- Contains an Instruction Unit:
  - Contains local memory that loads instructions from a file using $fopen.
  - Instruction unit executes instructions read from the local memory one by one.
- Consists of a memory interface that is connected to a system memory.
  - The processor should fetch data from the system memory of size 16 KBytes.
  - The data size should be 2 Bytes. Max operand size is 8 bits.

Processor Design Specification:

- Arithmetic and Logic Unit (ALU):
  - Executes instructions for ALU593
- Instruction unit:
  - Each instruction is 4 bits
  - Loads instructions from a file using $fopen() into an array : array size is 1KBytes
  - Executes instructions read from local memory one by one
- Memory Interface Unit:
  - 16 KBytes system memory interface
  - 8 Bytes data size
  - Makes read/ write request to memory
    - Read request: Asserts request, waits for read data. Removes request upon receiving response.

- Write request: asserts request with write data, removes request upon receiving response.



ALU593 Interfaces:

Inputs:

- Reset
- Clk
- A[7:0]
- B[7:0]
- OP[3:0]; // operations specified in table
- Start; // once Start received, ALU starts OP on A and B

Outputs:

- Result[15:0];
- Done; // ALU is done, ready to receive next Opcode

- ERR; // Error occured

Intruction Unit:

- Inputs:
  - Reset;
  - clk;
  - DATA[7:0]; // receives data from Memory Interface unit
- outputs:
  - Start; //connects to ALU593 to start operation
  - A[7:0]; // connect to ALU593
  - B[7:0]; // connect to ALU593
  - Addr[13:0]; // address to main memory
  - RESULT[15:0]; //results to be sent to main memory - connect to mem interface unit
  - Store; // issue store signal to mem interface unit
  - Load; // issue load signal to mem interface

Memory Interface Unit:

- Inputs:
  - Reset
  - Clk
  - Result[15:0];
  - Store;
  - Load;
  - Addr[13:0]
  - Mem_resp;
- Inout:
  - DATA[15:0]; // Writes use 15:0/ reads use 7:0
- Output:
  - Done;
  - Data[7:0]
  - Write_req; to SMM
  - READ_REQ; to SMM
  - ADDROUT[13:0]; to SMM

Top Module (Proessor_System):

- Input:
  - Clk,
  - reset,
  - mem_resp;
- output:
  - addr[13:0];
  - WRITE_REQ;
  - READ_REQ;
  - ERR#;

- INOUT data[15:0];

Instruction Unit Functionality:

- Reads instructions from Mem array
- Keeps 2 registers A, B for loading data from main memory
- Sample of Instruction mem array content:
- Load 0x10 A_B=0; IU sends load request o mem interface unit by asserting Load=1, store=0, addr=0x10; when mem_done=1; wait for next mem_resp; mif.data gets loaded into register A.
- load 0x11 A_B=1; IU sends load request to MIU by asserting Load=1; store=0;addr=0x11; when mem_done=1; wait for next mem_resp; mif.data gets loaded into register B.
- Add A, B; send Opcode/a/b to ALU593; wait for done
- Store 0x12; send store request to MIU with result[15:0], addr[13:0]; wait for mem_resp

---

Memory Subsystem Specifications:

- Memory Size: 16Kbytes
- Data size = 2 Byte <what should reasonable size of data request>
- Memory Block size = 2 Bytes
- Read Access Latency: 10 cycles
- Write Access Latency: 10 cycles
- Memory connects to 4 Processors via shared bus
- Shared bus uses round robin arbitration
- Processors support Modified, Invalid (MI) Processor Coherency scheme
- Module Interfaces:
  - Inputs:
    - clk: Clock signal
    - reset: Reset signal
    - mem_read_req: Memory read request signal from the processor
    - mem_write_req: Memory write request signal from the processor
    - mem_write_data: Memory write data from the processor
    - processor_0_req, processor_1_req, processor_2_req, processor_3_req: Processor read/write request signals
    - processor_0_data, processor_1_data, processor_2_data, processor_3_data: Processor read/write data signals
  - Outputs:
    - mem_read_data: Memory read data to be returned to the requesting processor
    - processor_0_resp, processor_1_resp, processor_2_resp, processor_3_resp: Processor read/write response signals

Memory Subsystem Functionality:

- On receiving a memory read request, memory subsystem initiates the read operation and returns the read data after the specified latency of 10 cycles.
- On receiving a memory write request, memory subsystem initiates the write operation and completes the write operation after the specified latency of 10 cycles.
- Memory subsystem arbitrates between multiple processor requests using a round-robin algorithm.
- Memory subsystem ensures processor coherency using the Modified, Invalid (MI) Processor Coherency scheme. When a processor updates a data item, it marks the corresponding memory block as Modified (M) state. The memory subsystem ensures that the other processors accessing the same memory block are in the Invalid (I) state and their data is updated from memory before allowing any further write access to the block.

Memory Subsystem Design:

- The memory subsystem is designed as a module that contains :
    - memory array
    - a controller for handling read/write operations (receives requests from memory bus and performs operation with memory array)
    - a controller that maintains processor coherency (supports Modified, Invalid (MI) Processor Coherency schemes)
- The controller contains a round-robin arbiter for handling multiple processor requests.
- The memory array is partitioned into memory blocks that correspond to processor lines.
- Each memory block contains the data and the corresponding coherency state.
- The controller maintains the coherency state for each memory block and updates the states based on processor read/write operations.
- The controller ensures that processor read requests are served from memory only if the coherency state is Invalid (I) or Shared (S). If the coherency state is Modified (M), the controller denies the service request to the controller. Corresponding microprocessor keeps the request asserted till served.
    - Verification needs to check if a processor has unserviced request for very long time.
- The controller ensures that processor write requests are served from memory only if the coherency state is Invalid (I) or Shared (S). If the coherency state is Modified (M), the controller first updates the corresponding processor line and then invalidates the coherency states of other processors accessing the same memory block.

Here is a high-level outline of the main components and functionality that would be included in the SystemVerilog code for the shared memory subsystem:

- Memory controller module:
    - This module contains the memory array, coherency state table, arbiter for memory bus, and a separate controller for checking coherency.
    - The arbiter is implemented using a round-robin algorithm to handle multiple processor requests.
- Coherency state table:
    - This is a lookup table that tracks the coherency state for each memory block.

        - It contains one entry for each memory block, with fields for the block address, data, and coherency state.
    - Memory array:
        - This is an array of memory blocks that stores the data.
    - Processor interface:
        - Each processor has an interface to the memory controller module.
        - The processor interface includes input/output ports for read/write requests, data, and responses.
    - Processor module:
        - Each processor is implemented as a separate module.
        - It contains a processor array, which stores the processord data.
        - The processor module implements the Modified, Invalid (MI) processor coherency scheme to ensure that the processord data is up-to-date.

OP[0:3]	Operation	Description
0000	NOP	No operation
0001	A + B	Adds A, B
0010	A & B	And operation on A, B
0011	A ^ B	XOR operation on A, B
0100	A * B	Multiply A, B as Int
0101	A + 2* B	special function
0110	A * 2	special function
0111	A * 3	special function
1000	Load	Loads from Memory to register inside ALU
1001	Store	Loads from ALU register to Memory
1010	Reserved	designer may define what reserved do
1011	Reserved	designer may define what reserved do
1100	Reserved	designer may define what reserved do
1101	Reserved	designer may define what reserved do
1110	Reserved	designer should define what reserved do
1111	NOP	No operation

- Please refer to table from assignment #2