ECE 445

Computer Organization and Design

SPRING 2009

MP3: LC-3b Processor with Cache

v 1.0

Introduction:

For this MP, you will be extending your existing LC3b processor to include a direct mapped cache with 8 blocks of 16 words each. You can either choose to extend your finished MP2 or build on the solution provided on Blackboard. If you choose to build on your version of MP2, please make sure that the new memory file and the included package is compatible to your design before starting MP3.

The summary of tasks are as followed :

- Build cache based on the given diagrams and codes
- Cache Control : Create an update cache sequence
- Cache Datapath : Create three components as specified in the following pages
- Comparison between MP2 (with the new memory) and MP3

Note : If you decide to build MP3 based on your existing MP2, you may need to modify the new memory file or your control on how they interact with upper byte.

What you CANNOT do in MP3:

No delay values should be taken-off in any part of the design. All delays that have been specified in the MP2 solution should be retained and any new blocks or statements that are added should have appropriate delays specified explicitly.

WebCT and TA Assistance:

Like it has been in MP1 and MP2, please check Blackboard regularly for any updates or discussions posted. Also, please do post on the Blackboard discussion, any additional/important detail that you have understood while you work on MP3 (Of course not solutions). It may help your colleagues or may even help you if you have misunderstood some part.

The TA will be available for guidance during the hours posted on the class webpage. Also, note that the TA is going to guide you during the project and not provide ready solutions to implement your design.

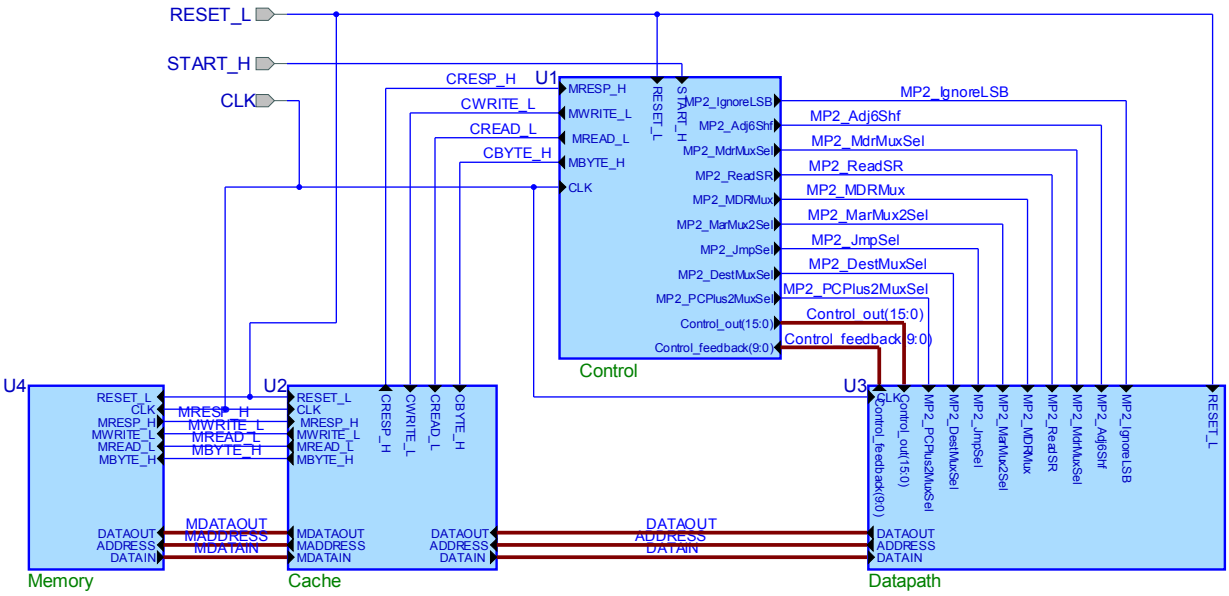
Files:

See blackboard for all necessary files.

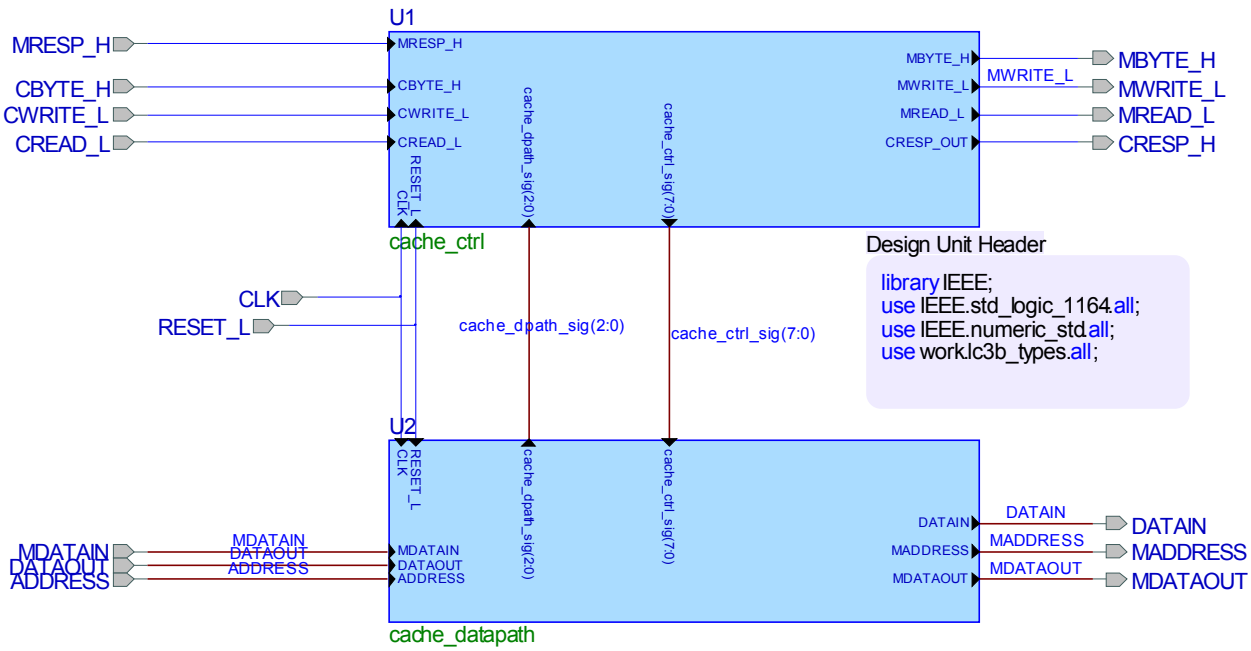
Top-Level (LC3b_bd.bde)

Design Unit Header

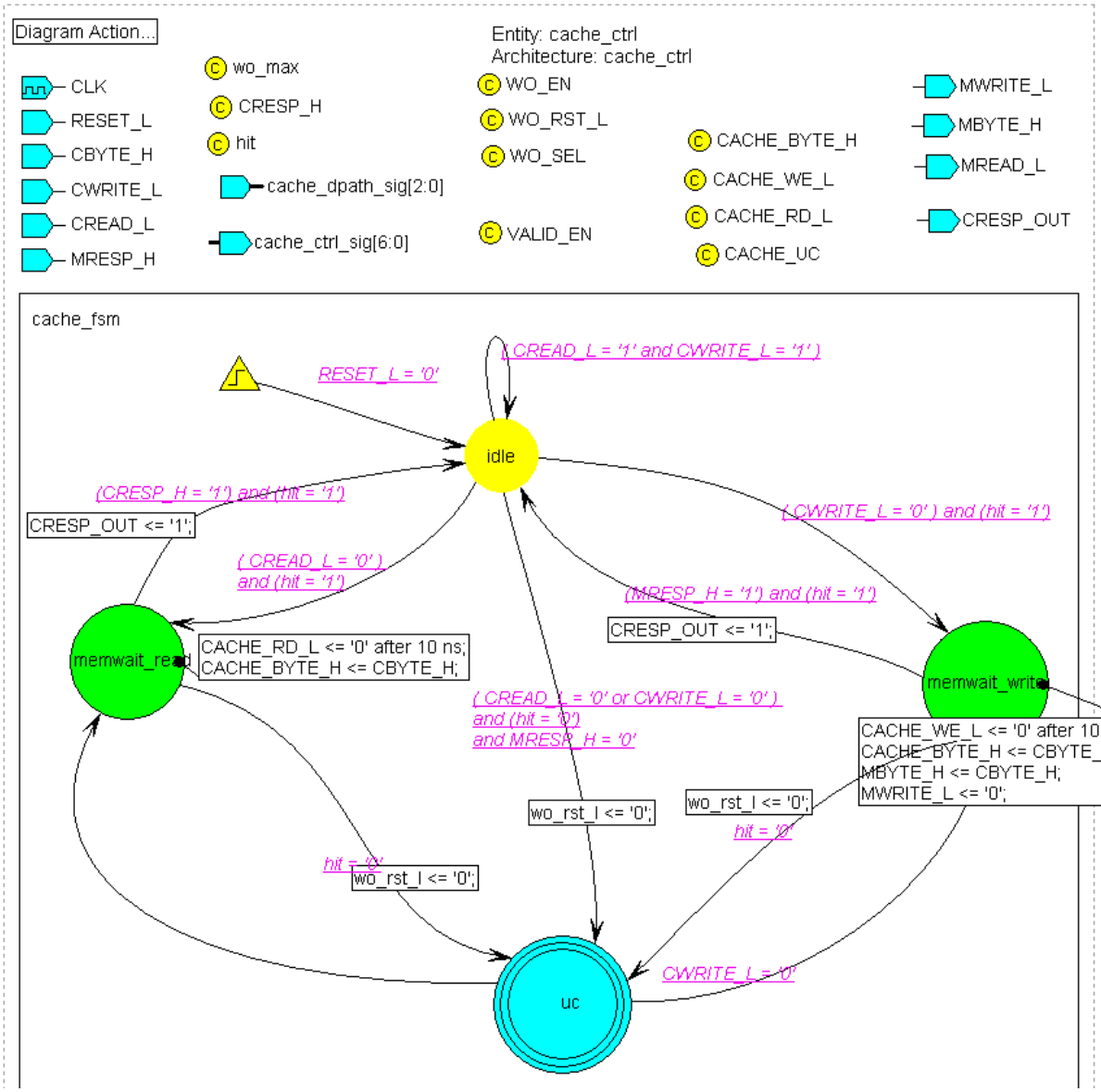
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.lc3b_types.all;
```



Cache (Cache.bde)



Cache control (cache_control.asf)

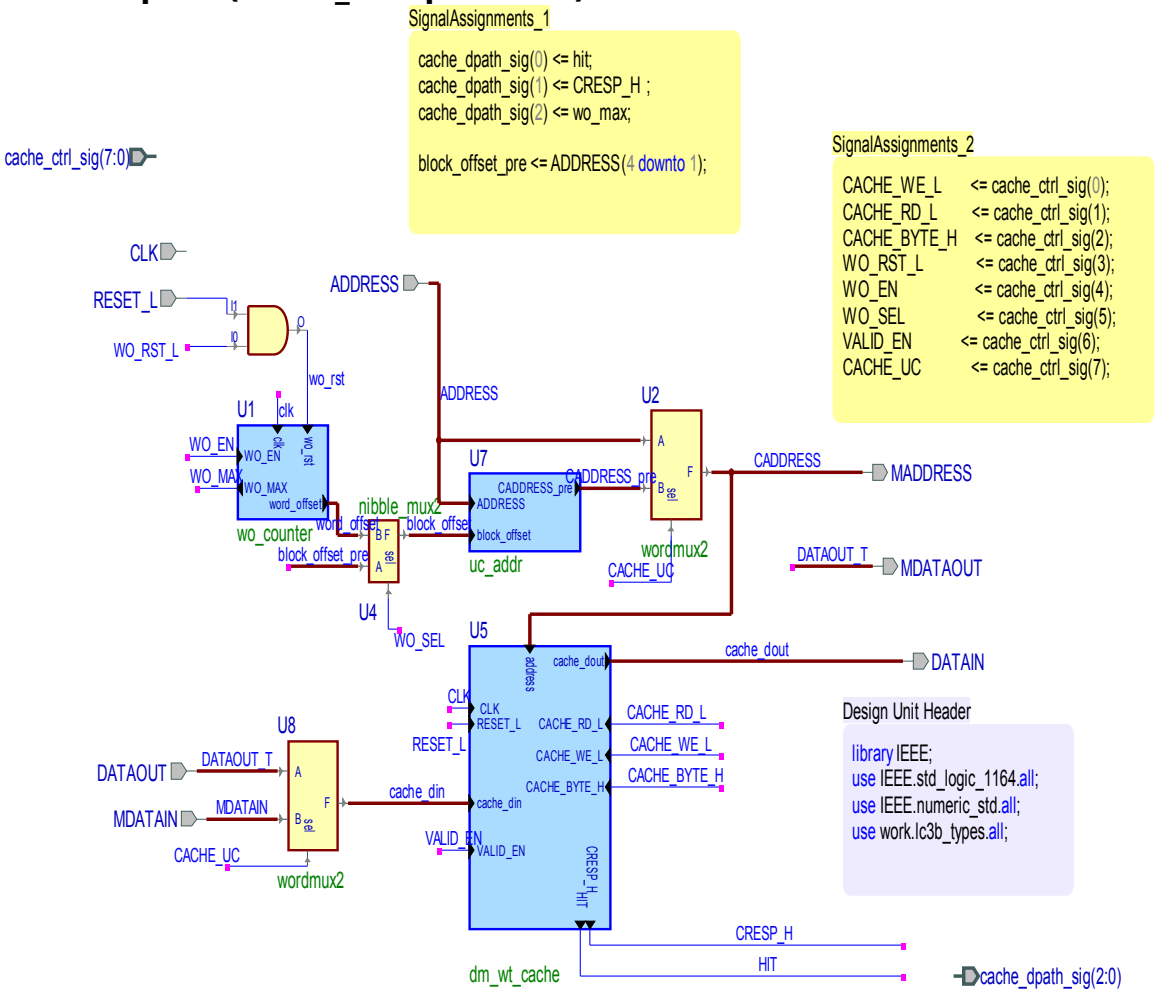



```
-- diagram ACTION
hit      <= cache_dpath_sig(0);
CRESP_H <= cache_dpath_sig(1);
wo_max   <= cache_dpath_sig(2);


cache_ctrl_sig(0) <= CACHE_WE_L;
cache_ctrl_sig(1) <= CACHE_RD_L;
cache_ctrl_sig(2) <= CACHE_BYTE_H;
cache_ctrl_sig(3) <= WO_RST_L;
cache_ctrl_sig(4) <= WO_EN;
cache_ctrl_sig(5) <= WO_SEL;
cache_ctrl_sig(6) <= VALID_EN;
cache_ctrl_sig(7) <= CACHE_UC;
```

For uc (update cache) state, you will need to create an algorithm to update the blocks inside a specific cache location.

Cache datapath (cache_datapath.bde)



Note : All blue lines are std_logic. All thick red lines are bus size with the width of LC3b_word. With the following exceptions that contain the bus width of LC3b_nibble ➔ word_offset, block_offset_pre and block_offset

Cache datapath specification

You will need to create 3 files to make the datapath work, which are

- wo_counter (wo_counter.vhd)
  - wo_counter is a 4-bit (LC3b_nibble) counter with a built-in MAX comparator. The counter has synchronous active low reset and should be sensitive to clk signal only. word_offset is the current value of the counter. wo_max is the comparator, it should be high when a the counter reaches its maximum value.
- uc_addr
  - uc_addr stands for update cache address. It takes in an address and a block_offset. It outputs a new address (CADDRESS) based on inserted block_offset.
- nibble_mux2
  - 4-bit, 2 input mux