EEEE-521-621-Lab8

1) From mycourses download on your desktop this pdf file, fmlRISC-CodeAssignments-version2.pdf, and the *.qar of interest.
2) The goal of the next two labs is to convert your "monolithic" blocks of memory (PM and DM or MM) into a hierarchical memory blocks as follows:
    a. Harvard:
        i. PM ➔ Instruction Cache and PM
        ii. DM ➔ Data Cache and DM.
        iii. The two hierarchical memory blocks are identical.
    b. Von Neumann: MM ➔ Unified Cache and MM.
3) Towards this end, in this lab you'll be designing and verifying a Content Addressable Memory – CAM memory. I am providing the VHDL and the Verilog versions of a CAM.
4) Its symbol and a snapshot of its diagram are shown at the end of this document. Read the comments in the code, your lecture notes, and corroborate with these figures to thoroughly understand how this CAM works.
5) Before adapting this CAM to your specifications, you need to infer the latter from the corresponding table at the end of this lab. An example will be run in the lab on the board.
6) Your current memory doesn't need changes.
7) If you don't have a working pipelined CPU, it is OK for this (0 points penalty) and next (-10 points penalty) labs to continue with the non-pipelined CPU. However, if you'll finish the pipelined CPU you can still get partial credit for it.
8) In the case of a cache HIT, the access time is one cycle, exactly as it was before with the monolithic memory block. However, in the case of a miss, either read or write, the block will have to be transferred into the cache. In principle, the speed penalty is as follows:
    a. First cycle to determine that there is a miss.
    b. Next 8(16) cycles write back the block chosen for replacement into the memory. This is only necessary if the *"Dirty Bit"* is set.
    c. Next 8 (16) cycles read the required block into the cache.
    d. Last cycle to repeat the access to the cache and obviously HIT and complete it.
9) Therefore, the only change you'll need to make to your CPU (non-pipelined or pipelined) is the ability to STALL for the number of cycles necessary to handle a miss.
10) Once the tag memory determines a match, it generates an address that points to the respective block location in the cache. This address value concatenated with the word address field points finally to the right word in the cache.
11) As you know, the wizard allows you to create only clock synchronous memory blocks. Because the CAM I'm providing is asynchronous, you should be able to use the same phase of the clock you used so far to register the address into the cache block in the case of a HIT access.
12) The rest of the hierarchical memory block will be further explained during the next lab.
13) If time permits, show your simulation to your TA. Note: only the TA will assign a grade.
14) Archive your project.

15) Write your report and upload it along with your archived project(s) in the dropbox on mycourses, as described in the lab policy.
16) This concludes this week's lab.
17) **Grading**:
    a. 30 points for a complete working CAM or nothing.

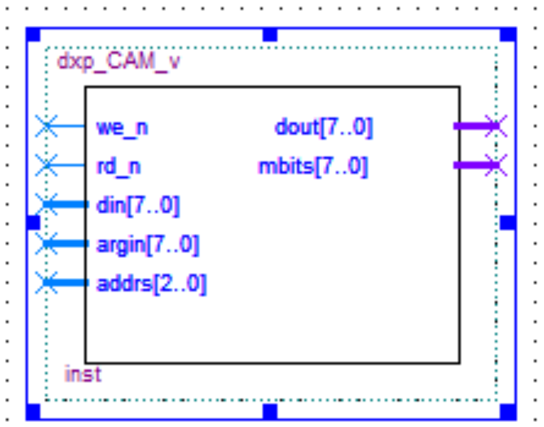
521 fmlRISC

Your binary number code		dxp_RISC	1	0	Comments
B5		Both			
B4		Both	8 Block Locations	16 Block Locations	Cache Size
B3		Both	8 Words	16 Words	Block Size
B2		16-bit	Fully Associative	Direct	Mapping Function
B1		Both	von Neumann	Harvard	Program and Data Memory Organization
B0		Both	14-bit	12-bit	Word Size
	Note: All architectures are Register/Register, i.e. Load/Store				

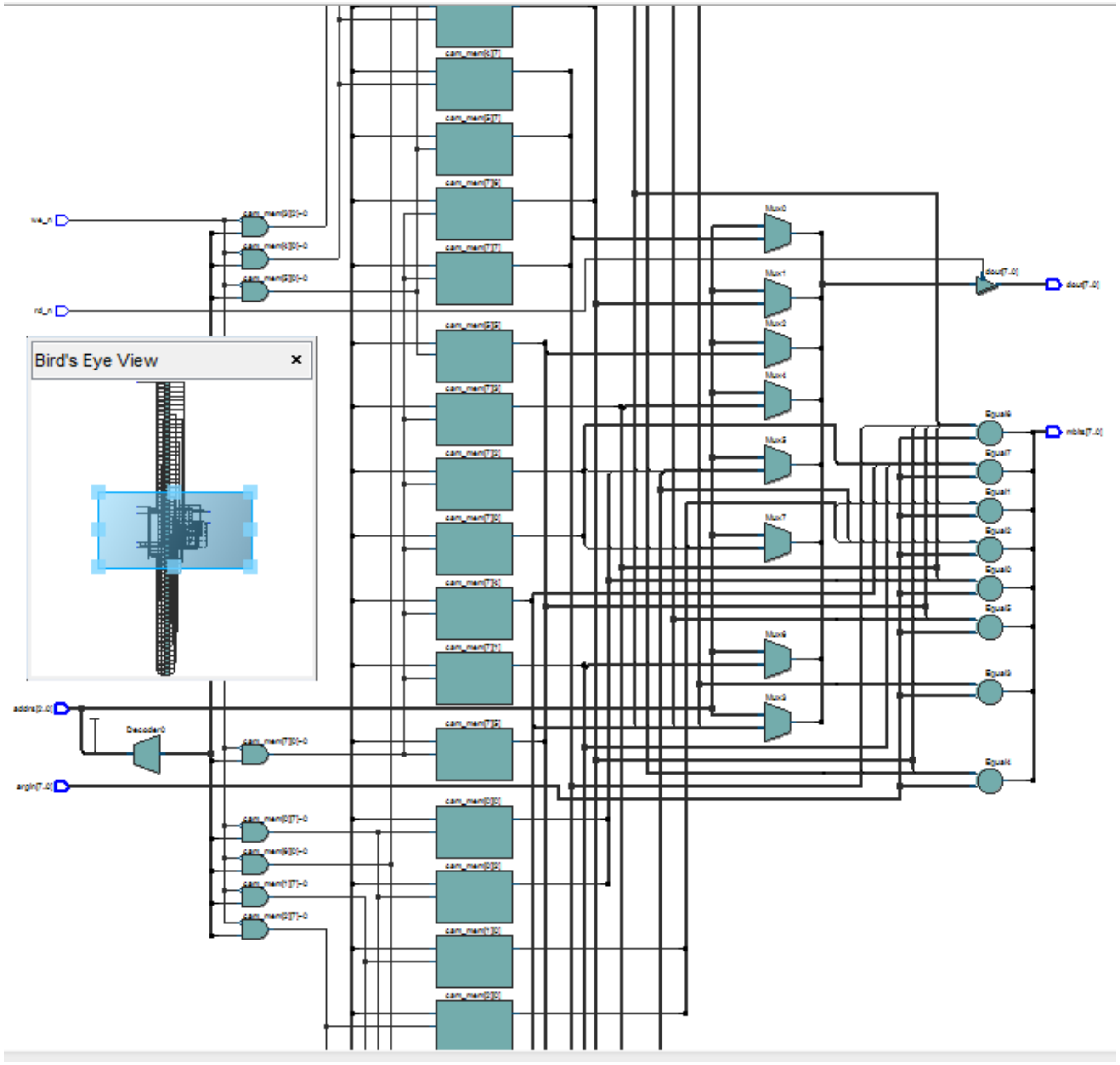
621 fmlRISC

Your binary number code		dxp_RISC	1	0	Comments
B5		Both			
B4		Both	8 Block Locations	16 Block Locations	Cache Size
B3		Both	8 Words	16 Words	Block Size
B2		16-bit	2 – Way	4 – Way	Mapping Function
B1		Both	von Neumann	Harvard	Program and Data Memory Organization
B0		Both	14-bit	12-bit	Word Size
	Note: All architectures are Register/Register, i.e. Load/Store				

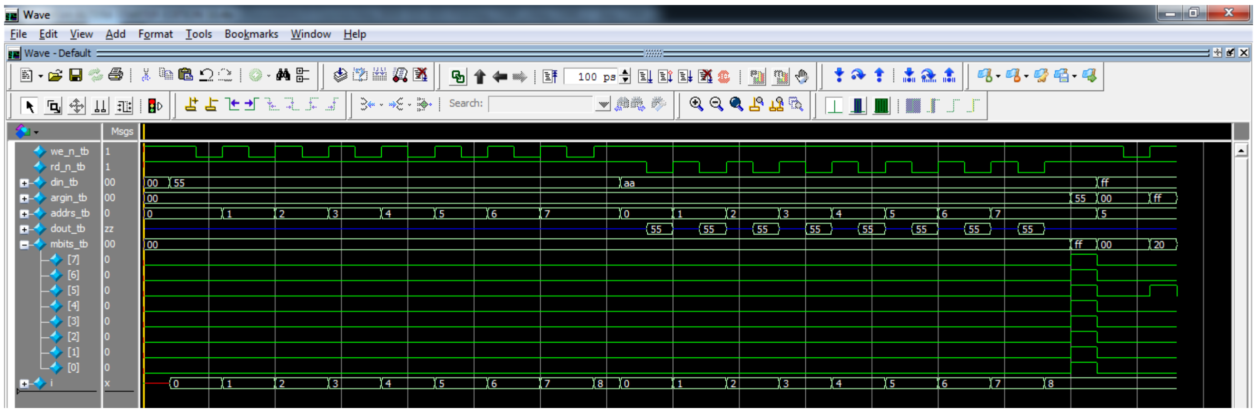
- Write cache hit policies
    - Write back–updates only cache. Updates MM only upon block removal.
        - "Dirty bit" is set upon first write to indicate block must be written back.
- Read miss - bring block in from MM
    - Wait until entire line is filled, then repeat the cache request.
- Write miss
    - Write allocate - bring entire block into cache, then update

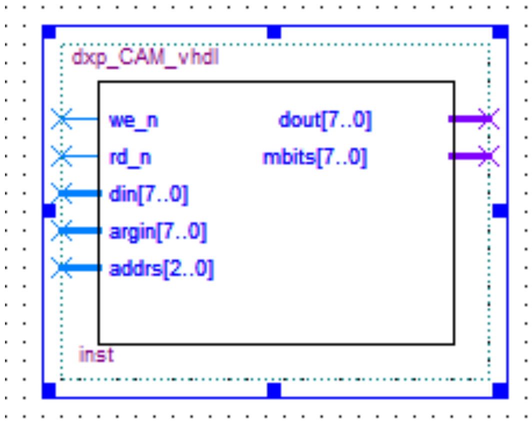


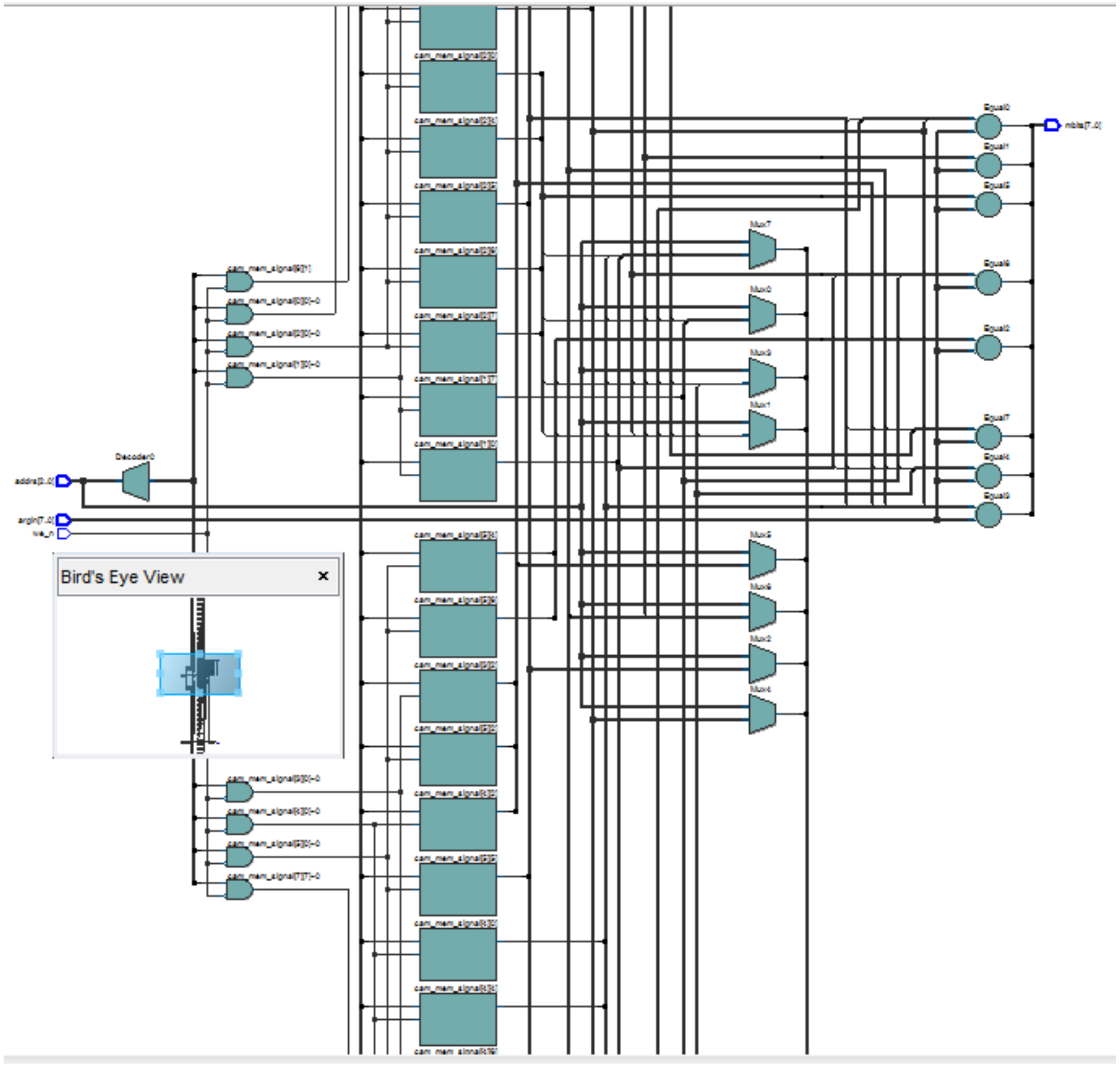
dxp_CAM_v
we_n
rd_n
din[7..0]
argin[7..0]
addrs[2..0]
dout[7..0]
mbits[7..0]
inst

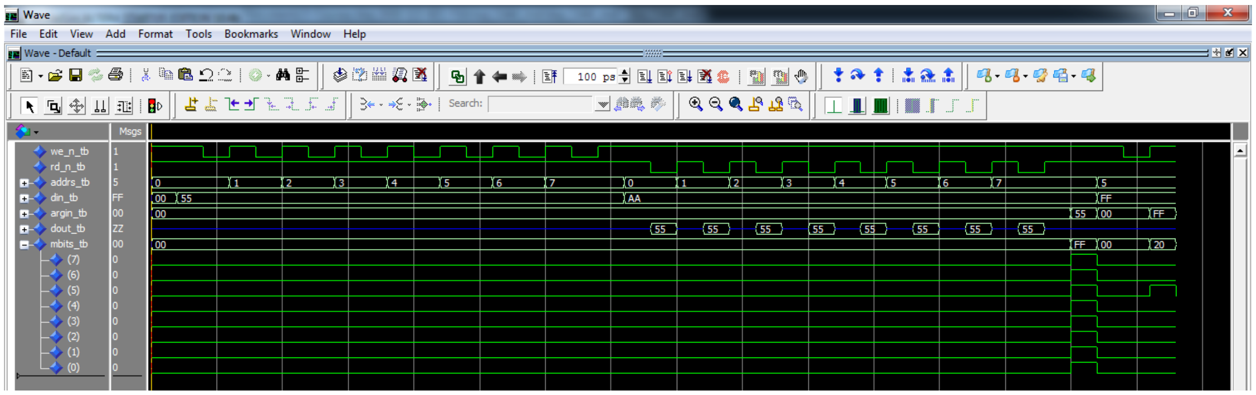


we_n
rd_n
addrs[2..0]
argin[7..0]
Decoder0
Bird's Eye View
dout[7..0]
mbits[7..0]
Mux0
Mux1
Mux2
Mux4
Mux5
Mux7
Mux6
Mux3


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
100 ps
Search:
Msgs
we_n_tb 1
rd_n_tb 1
din_tb 00
argin_tb 00
addrs_tb 0
dout_tb zz
mbits_tb 00
[7] 0
[6] 0
[5] 0
[4] 0
[3] 0
[2] 0
[1] 0
[0] 0
i x
00 55 aa ff
00 55 00 ff
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 5
55 55 55 55 55 55 55 55
00 ff 00 20
0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8


dxp_CAM_vhdl
we_n
rd_n
din[7..0]
argin[7..0]
addrs[2..0]
dout[7..0]
mbits[7..0]
inst


Decoder0
addrs[2..0]
argin[7..0]
we_n
Equal0
Equal1
Equal5
Equal6
Equal2
Equal7
Equal4
Equal3
mbits[7..0]
Mux7
Mux0
Mux3
Mux1
Mux5
Mux6
Mux2
Mux4
Bird's Eye View
×


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
100 ps
Search:
Msgs
we_n_tb 1
rd_n_tb 1
addrs_tb 5
din_tb FF
argin_tb 00
dout_tb ZZ
mbits_tb 00
(7) 0
(6) 0
(5) 0
(4) 0
(3) 0
(2) 0
(1) 0
(0) 0
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 5
00 55 AA FF
00 55 00 FF
55 55 55 55 55 55 55 55
00 FF 00 20