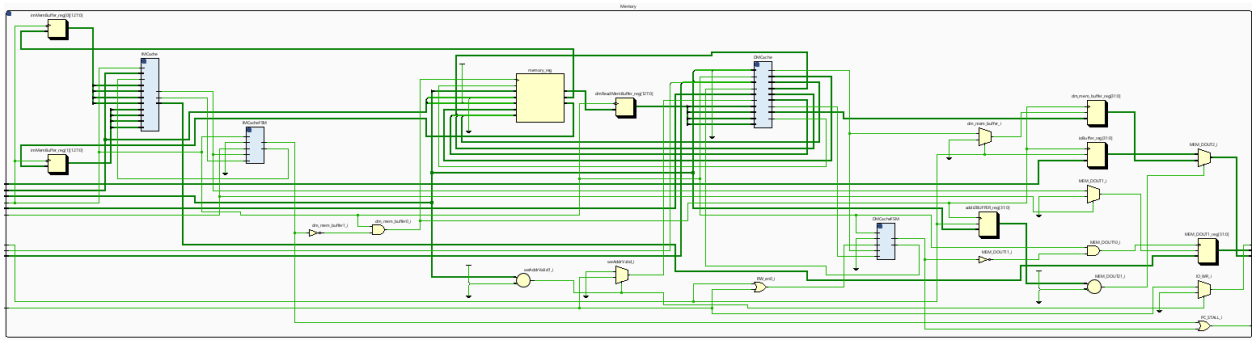
CPE 333 Lab 5: Caches

Contributions:

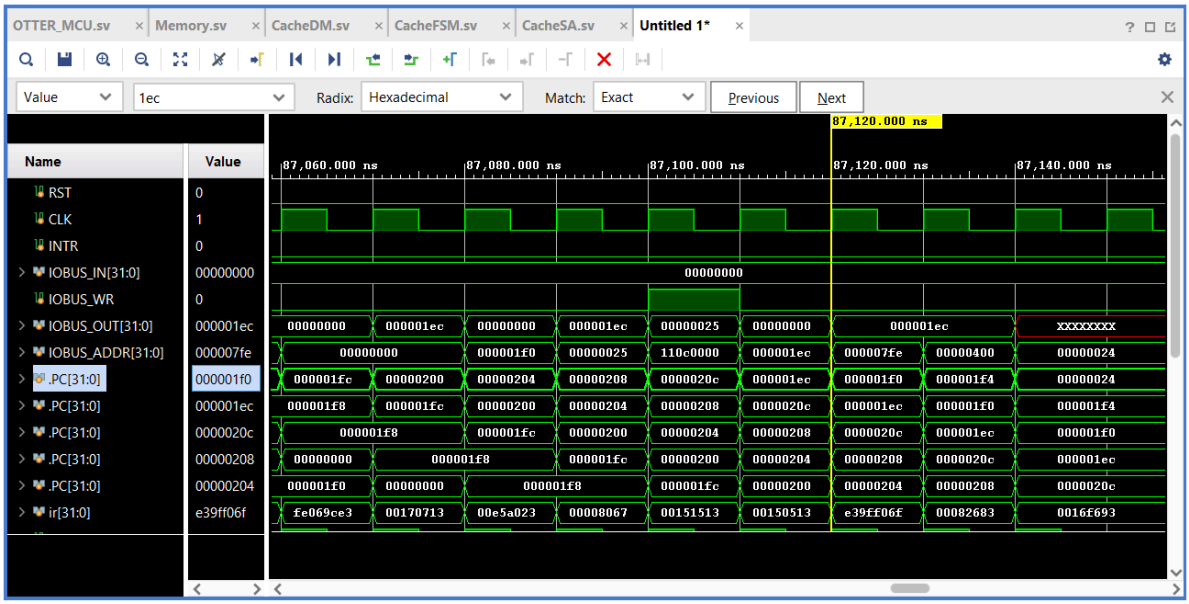
- Wyatt Tack: Wrote new verilog code
- Justin Rosu: Helped debug Cache
- Brayden Daly: Helped debug Cache
- Tyler Hamilton: Helped design portions of Cache

Design:

As per lab instructions, the cache was implemented onto the Data Memory section, with a size of 4x4x16 bytes, Set Associative. The design entailed a similar initial process of the direct map cache, with a synchronous decoding of index to narrow down the first block. The second block is a large decoder which detects if the sub-block is found (through a verilog for loop). From there either the cache flags a miss, and the FSM handles the update (with the input of a new block, and the output of the old block (under the condition of a dirty bit in that block). The Replaced block is done through a 2 bit counter for the LRU (least recently used) bit, in which a used block is set to 3, through subtracting every block that had a greater LRU by 1, and setting itself to 3. Replaced blocks are given a 3, through subtracting all blocks by 1 (0 overflows to 3). Due to issues in hardware with having a memory block with multiple read busses (as to read from the main memory for the cache is for multiple words per block), the word size in the main memory BRAM block was increased to 128 bits, 16 bytes instead of 4. This does not impact the 32 bit address architecture provided on the MCU, yet stores the values a little differently due to resource constraints in the FPGA.

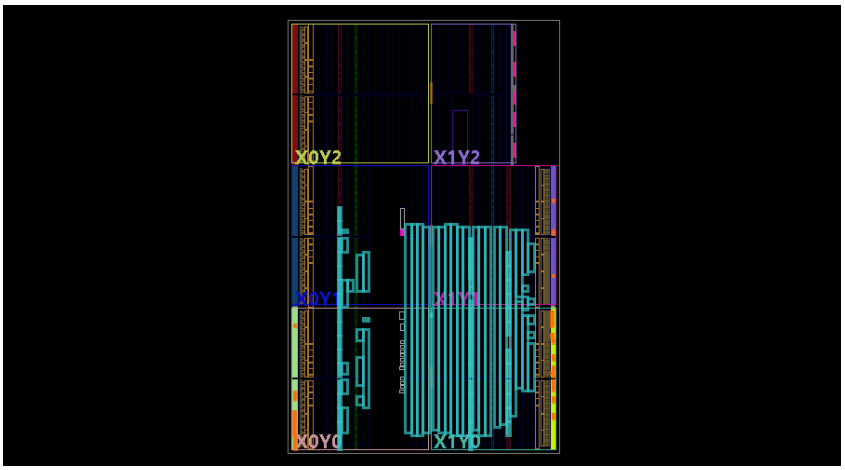


Vivado Simulation:



Above is shown the TEST_ALL simulation, with a clock count of 8,712 cycles. This is compared to the CPU without a data memory cache, with an execution of 8,691 cycles. Although there is a 21 clock cycle loss, we do in fact verify the cache conceptually, as well as have the ability for more cost-effective memory. Adding a cache shows that we can replace our currently distributed RAM block with a cheaper memory option, which allows for a more cost-efficient design.

Implementation:



The cache addition provides a larger increase in resources than its pipelined counterpart. The total device uses 27% of LUTs, 16% of FFs, 43% IO, and a whopping 87% BRAM (due to multiple read/writes of large bit counts, the memory is cloned and shared across multiple blocks). The device additionally uses a decent 0.167W of power with a 62/38 dynamic/static split.