Benjamin Kalnbach, Tongyu Liu

bak7, tongyul3

12/05/2023

CP3:

**Progress Report:** The main objective of CP3 was to design and instantiate advanced features that are not necessary to run coremark(i), but could greatly improve performance. Of the available features on the recommended list we designed and implemented: advanced multiplier, basic divider, branch target buffer, and basic prefetching. We had intended to also complete multilevel caches and victim cache, however we lost one of our teammates towards the end of cp2 and couldn’t complete them within the allotted time. Benjamin completed the advanced multiplier and basic divider. Tongyu completed the branch target buffer. Prefetching was half Tongyu and half Benjamin. Tongyu also spent a large amount of time getting the caches to work since they were not working completely correctly, which caused coremark to not run correctly. Both Tongyu and Benjamin spent a lot of time debugging various aspects from cp2 to get coremark to run. Multiplier was based on Karatsuba’s divide and conquer algorithm. The design using recursive calls to a function netted a smaller area and a 3 cycle multiply vs the hardware implementation, which was larger and was a 5 cycle multiply. Division was the basic shift and subtract algorithm for binary division. This process took 32 cycles to complete a division. Branch target buffer in it’s first version was 16 target predictions and 8 jump predictions. Each prediction was based on the previous time the btb saw the same jump and if it took it last time, it would predict taken, else not taken. The later version of btb was cut down to 8 branch target and 4 jump target predictions to reduce the area btb took up. Prefetch buffer stored two cachelines. When i\_cache and d\_cache are not using pmem arbiter reads the next cacheline into the prefetch buffer. Then when(or if) the i\_cache misses and wants one of the line we prefetched, arbiter would give the i\_cache the cacheline in prefetch, if prefetch didn’t have the cacheline however, arbiter would request it from pmem as normal. Multiplication and division decreased the ipc(by -.02), but would halve the time required to complete coremark(im). Btb netted the largest ipc gain of .03, while prefetch gained a more modest .0015 ipc gain.

**Roadmap:** Reducing the area by 60000 units is required in order to meet the 75000 unit requirement. SRAM using negedge clocking, and the critical path created by generating the control word all need to be reduced in order to better the timing. That being said timing is already met at the baseline requirement of 10000ps. The method of choice would be reducing the caches to a single way and 16 sets. SRAM for caches takes up ~80000 units, so cutting down the caches size to a quarter should get us very close to the required area. Further area savings could be found by reducing prefetching to a single cacheline, and if it doesn’t perform well just get rid of it. Also, btb can be cut down to 4 branch predictions and 2 jump predictions. Better power would also be nice, but give that we are only sipping 1.38 mW, it likely will not be a main concern for us. One potential fix for the timing issues is instantiating our caches as flip-flops which would also lower the time required to hit. Another fix would be separating the really big always\_comb block in control to several smaller always comb blocks. That way those blocks could be done in parallel rather than sequentially, hurting our overall critical path. These fixes would both allow us to meet area requirements, but also to decrease our clock period, allowing the processor to run faster. In fact if we used the flip-flops for SRAM we could see a large increase in ipc.