University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2020 — Final Solutions — Friday, December 18

See exam as given for code, Loop E design, baseline system.

1. I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity and the exam regulations `https://www.seas.upenn.edu/~ese532/fall2020/final_details.pdf` in completing this exam.

2. Assuming:

   - up to 6-input xors or ors can be packed into a single 1ns operation
   - a comparison followed by a mux can be performed in a single 1ns operation
     - mux alone can also be performed in a 0.2ns
   - each memory operation can be performed in a single 2ns operation
   - an add or increment can occur in a single 1ns operation
   - bitpos registers are preloaded

   What is the cycle-bound II for loop E in ns?

   7.2 ns

3. Explain your II answer to previous question.

- critical loop is state to newstate to state
- selection of val8 is not in critical loop; it can be pipelined in a prefix feeding input to the loop.
- 0.2 ns for bitpos mux selects (all in parallel)
- 2 ns for xor-reduce (20 inputs, can be done in 2 stages – first with 5-inputs per xor, second with 4 – general: $\lceil \log_6(20) \rceil$
- 2 ns for lookup memory read
- 1 ns for mux-compare to select value based on key match
- 1 ns for or-reduce (4 inputs, single stage)
- 1 ns for final selection of updated state from lookup or from init_lookup.
- write to results can happen pipelined after critical loop
- init_lookup memory happens in parallel with lookup and is a shorter path

4. Provide code for the body of Loop E based on the pipeline show.

   Recreate loops as flagged in the diagram.

```
for (b=0;b<8;b++) { // loop E
  int slot_addr=0;
  for (f=0;f<KEYLEN;f++) { // loop F -- unroll
      if ((key>>f)& 0x01)
        slot_addr^=bitpos[f];
    }
   ap_int<128> lookup_result=lookup[slot_addr];
   int found_val=0;
   for (int g=0;g<BUCKET_CAPACITY;g++)
     {
       int slot = (lookup_result>>(32*g))&BUCKET_MASK;
       int slot_val = slot & VAL_MASK;
       int slot_key = (slot >> VAL_MASK_MASK) & STATELEN;
       if (slot_key==key)
           found_val|=slot_val;
     }
     if (found_val==0)
        {
          result[result_count]=state;
          result_count++;
          state=init_lookup[val8];
        }
     else
        state=found_val;
}
```

5. Estimate the throughput in cycles per frame for loop A running on the baseline processor.

   2,637

6. Explain your throughput answer above.

   - 8 cycles to read 240b (4 64b transfers) from main memory
   - 2,624 cycles for 64 iterations of B at 41 cycles per iteration
     - 5 cycles to read bitloc
     - 32 cycles for 8 iterations of C at 4 cycles per iteration
       * 3 cycles for &, ==, divide by power of 2 (shift right)
       * 1 cycle for divide by 2 (shift right by one)
     - 3 cycles for &, ==, or for conditional update of result
     - 1 cycle for shift left (multiply by 2)
   - 5 cycles to write to temp at end

7. Where is the bottleneck in throughput processing frames?

   - Loop A compute
   - Loop A memory
   - Loop E compute
   - Loop E memory

8. What is the Amdahl's Law speedup if you were to accelerate the bottleneck identified in the previous question? Support your answer with calculations.

   7.9

   Of the 2637 cycles, $8+5+64\times5=333$ are for memory operations. The rest is compute. Loop E total runs 7.2×8 ns per frame, so both its memory and its compute is smaller than either of the Loop A times. $\frac{2637}{333} = 7.9$

9. What is the smallest granularity that you can profitably stream data between Loop A and Loop E?

   - Entire tmp[] (all NFRAMES words in tmp[], each of which is a 64b word)
   - single 64b word
   - no streaming possible

10. Use the scratchpad memory to accelerate memory operations in Loop A.

    Indicate which data you place in the scratchpad.

    Provide code or other clear description of how you modify the provided code for Loop A to exploit the scratchpad memory.

    Read bitlocs into the scratchpad memory before Loop A, and use from scratchpad memory.

    It's also possible to save a few cycles by loading segments of frames into the scratchpad. Scratchpad is too small to hold the whole frames array. Each frame is already only used once.

    Use part of this box to provide justification to the numerical answer in the next question.

    Moving bitloc saves 5-1=4 cycles per iteration of B, or 256 cycles.

    At best, moving frames saves 5-1=4 cycles per frame.

11. For your revised code in the previous question, what is the throughput in cycles per frame for the revised implementation of Loop A?

    Just bitloc: 2381

    bitloc and block reads of frames: 2377

12. Classify each loop as sequential, reduce, or data parallel:

    - Loop A Data Parallel
    - Loop B Reduce
    - Loop C Sequential
    - Loop D Sequential
    - Loop E Sequential
    - Loop F Reduce
    - Loop G Reduce

13. What is the cycle-bound II (unlimited hardware) for loop A (assuming no bottleneck on input frames or output tmp)?

    $II_{cylce\_bound} = 1$

14. What is the latency bound (unlimited hardware) for loop A executing all NFRAME frames (assuming no bottleneck on input frames or output tmp)?

    42 cycles

15. Support your numeric answers to the previous two questions.

    13: A is data parallel, there are no cycles = no dependencies to prevent the next iteration of A from beginning.

    14:

    - 5 cycles to read val from frame, bitloc
    - 24 cycles for C – 8 iterations of 3 cycles each – all 64 instances of C in parallel
      - 3 cycles for &, ==, div (shift)
      - bitloc shift (div) happen in parallel with above
    - 2 cycles for &, == for each val coming out of C (all 64 in parallel)
    - 6 cycles ($\log_2(64)$) for or-reduce at end of B
    - finalpos can happen in parallel
    - 5 cycles to write tmp

16. Describe a VLIW architecture (types of operators and numbers of each, custom memories (memory ports) as needed) for executing Loop A that has a Resource Bound throughput (cycles per frame) that is the same throughput as the pipeline for Loop E such that Loop A is no longer the bottleneck. For simplicity, you may assume a monolithic, multi-ported register file. Try to identify an architecture with minimum hardware achieving the target resource bound. Provide description and calculations to support your answer.

E completes one frame per 7.2 ns $\times 8 = 57.6$ ns or 57 cycles. That becomes our RB target.

Accounting operations needed:

- one read from frame
- 64 reads from bitloc
- 64 $\times$ 8=512 &, ==, shift-right, shift-right
- 64 &, ==, or, shift-left
- one write to tmp[]

Combining same type operations:

- one read from frame
- 64 reads from bitloc
- 576 &
- 576 ==
- 1152 shift-right
- 64 shift-left
- 64 bitwise ORs
- one write to tmp[]

Divide by 57 (and round up), the operators we need are:

- one read port for frame – fine to be from existing main memory
- 2 read ports from bitloc – probably from register file; or could be scratchpad
- 11 &

- 11 == (could combine & (previous) and this == into one specialized operator)
- 21 shift-right
- 2 shift-left
- 2 bitwise ORs
- 1 write port for tmp[] – fine to be existing bus to Loop E accelerator, even if shared with main memory for frame read

17. Given the resources identified in the previous question, how close to the target resource bound will a scheduled computation achieve? Explain your reasoning.

    Since the cycle-bound II is one, we can achieve the Resource Bound (RB) with software pipelining.