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

ESE5320, Fall 2024 — Final — Friday, December 13

- Exam ends at 5:00PM; begin as instructed (target 3:00PM)
  Do not open exam until instructed.
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration. All answers here.
- Unless otherwise noted, answers to two significant figures are sufficient.
- Sign Code of Academic Integrity statement (see last page for code).

____

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this exam.

Name: Solution

1	2a	2b	3	4	5	6	7a	7b	7c	8a	8b	8c	Total
10	5	5	10	10	10	20	10	2	8	3	4	3	100

Consider the following code to render augmented reality features on a real-time video stream

```
code_one.c        Sun Dec 08 12:38:47 2024        1

int WIDTH 2048
int HEIGHT 1024
int COLORS 3
int MASK 3

int VPARAMS 5
int VP_X 0
int VP_Y 1
int VP_XS 2
int VP_YS 3
int VP_ROT 4

int XOFF 2
int YOFF 2
int ROT 2
int XSCALE 2
int XSFACT 2
int YSCALE 2
int YSFACT 2


uint16_t reference[HEIGHT][WIDTH][COLORS];
uint16_t overlay[HEIGHT][WIDTH][COLORS+1]; // +1 for mask
int16_t sintable[360]; // -1 to 1 -- scaled by 2^14
int16_t costable[360];

void main() {
  while (true) { // loop Z
    augment_frame();
  }
}

void augment_frame() {
  uint16_t raw[HEIGHT][WIDTH][COLORS]; // uint16_t for 16b (2 byte) color per pixel
  uint16_t augment[HEIGHT][WIDTH][COLORS];
  uint16_t augmented[HEIGHT][WIDTH][COLORS];
  uint16_t old_viewpoint[VPARAMS];
  uint16_t viewpoint[VPARAMS];
  uint16_t *tmp_viewpoint;
  get_image(raw);
  tmp_viewpoint=old_viewpoint;
  old_viewpoint=viewpoint;
  viewpoint=tmp_viewpoint;
  compute_viewpoint(raw,reference,old_viewpoint,viewpoint);
  render_augmentation(viewpoint,overlay,augment);
  merge_frames(reference,viewpoint,raw,augment,augmented);
  send_image(augmented);
}
```

code_two.c   Sun Dec 08 11:55:19 2024   1

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;iy<HEIGHT;iy++) // loop F
                for (int ix=0;ix<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                    uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}

void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;iy<HEIGHT;iy++) // loop I
    for (int ix=0;ix<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;iy<HEIGHT;iy++) // loop K
    for (int ix=0;ix<WIDTH;ix++) // loop L
        {
          uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
          uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
              && (overlay[ty][tx][MASK]>0))
               for (int c=0;c<COLORS;c++) // loop M
                  image[iy][ix][c]=overlay[ty][tx][c];
        }

}
```

code_three.c   Sun Dec 08 11:44:25 2024   1

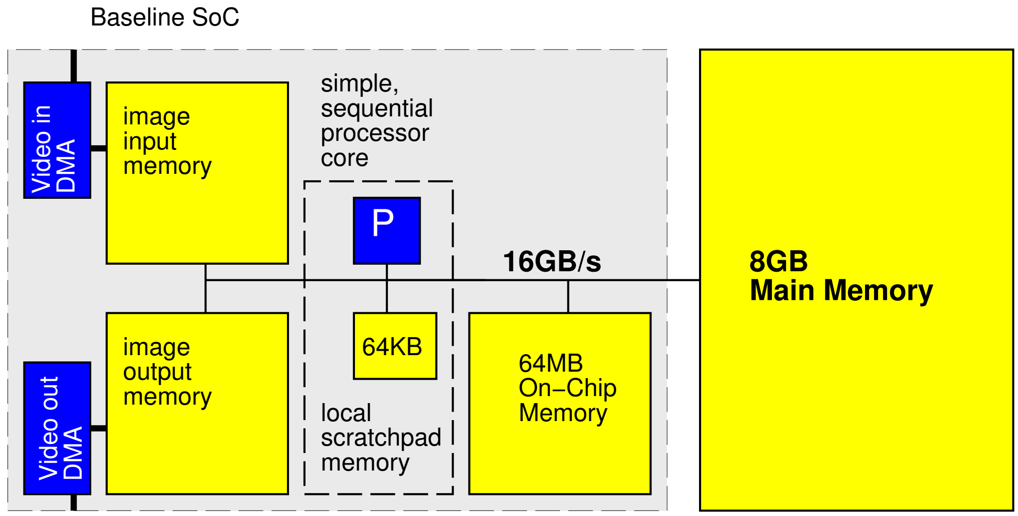
```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;iy<HEIGHT;iy++) // loop N
      for (int ix=0;ix<WIDTH;ix++) // loop O
        {
          uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
          uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
              && (augment[iy][ix]!=UNMAPPED))
            {
              uint32_t diff=0;
              for (int c=0;c<COLORS;c++) // loop P
                diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              if (diff<THRESH)
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
              else
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
            }
          else
            for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
        }
}

void get_image(uint16_t ***image)
{
  for (int iy=0;iy<HEIGHT;iy++)
    for (int ix=0;ix<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image[iy][ix][c]=image_in[iy][ix][c];
}

void send_image(uint16_t ***image)
{
  for (int iy=0;iy<HEIGHT;iy++)
    for (int ix=0;ix<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image_out[iy][ix][c]=image[iy][ix][c];

}
```

We start with a baseline, single processor system as shown.



- For simplicity (except problem 8), we will treat non-memory indexing adds (subtracts count as adds), compares, abs, shifts, and multplies as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with the adds, abs, shift, multiplies, and memory operations. (Some consequences: You may ignore loop and conditional overheads in processor runtime estimates; you may ignore computations in array indices.)
- Baseline (simple, sequential) processor can execute one multiply, compare, shift, abs, or add per cycle and runs at 1 GHz.
- Data can be transfered between pairs of memory (including main memory) at 16 GB/s when streamed in chunks of at least 1024B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to the main memory takes 100 cycles and can move 8B.
- Non-streamed access to image and 64 MB on-chip memories takes 10 cycles and can move 8B.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 16 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- Baseline processor is 1 mm^2 of silicon including its 64KB local scratchpad.
- By default, all arrays live in the 8 GB main memory.
- `image_in` and `image_out` live in the respective image input and image output memories.
- Arrays for `sintable`, `costable` and viewpoints (`old_viewpoint`, `viewpoint`) live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz. A compare-mux operation can also be implemented in 1 ns. Consider abs and shift free in hardware.
- Data can be transfered to accelerator local memory at the same 16 GB/s when streamed in chunks of at least 1024B.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time (in cycles) for compute operations and memory access for the computing components of `augment_frame`.

function	Compute	Memory
`get_image`	all DMA	$\frac{2048\times1024\times3\times2}{16} = 7.9\times10^5$
`compute_viewpoint`	$4^3\times2^2\times2048\times1024\times(12+3\times3)$ $= 1.13\times^{10}$	$4^3\times2^2\times2048\times1024\times(2\times100)$ $= 1.1\times10^{11}$ overlay[ty][tx] including mask is single read image[iy][ix] is single write
`render_augmentation`	$2048\times1024\times12 = 2.5\times10^7$	$\frac{2048\times1024\times3\times2}{16} + 2048\times1024\times2\times100$ $= 4.2\times10^8$ overlay[ty][tx] including mask is single read image[iy][ix] is single write
`merge_image`	$2048\times1024\times(12+3\times3)$ $= 4.4\times10^7$	$2048\times1024\times4\times100$ $= 8.4\times10^8$
`send_image`	all DMA	$\frac{2048\times1024\times3\times2}{16} = 7.9\times10^5$
`augment_frame`	1.1×10^{10}	1.1×10^{11}

2. Based on the simple, single processor mapping from Problem 1:

   (a) What loop is the bottleneck? (circle one)

`get_image`
(`compute_viewpoint`)
`render_augmentation`
`merge_frames`
`send_image`

   grade consistency with problem 1

   (b) What is the Amdahl's Law speedup if you only accelerate the identified function?

   $\frac{1.2\times10^{11}}{1.5\times10^{9}} \approx 81$

   grade consistency with problem 1

3. Parallelism in Loops

   (a) Classify the following loops as data parallel, reduce, or sequential?
   (b) Explain why or why not?

Loop	circle one			Why?
A	Data Parallel	(Reduce)	Sequential	min-reduce on best_score
F	Data Parallel	(Reduce)	Sequential	sum-reduce for score
K	(Data Parallel)	Reduce	Sequential	Computation for image[iy][ix] are each independent of other elements in array
N	(Data Parallel)	Reduce	Sequential	Computation for augmented[iy][ix] are each independent of other elements in array
Z	Data Parallel	Reduce	(Sequential)	must compute new viewpoint from one iteration/image before starting computation on next image

4. Data Streaming:

(a) Can the producer and consumer operate concurrently on the same input image? or must the consumer work on a different (earlier) input image? ("Same Image?" column)

(b) How big (minimum size) does the buffer (or other data storage space) need to be between the identified loops in order to allow the loops to profitably execute concurrently?

(Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Loop Pair	(a) Same Image?	(b) Size (bytes)
`get_image` → `compute_viewpoint`	N	12 MB
`compute_viewpoint`→`render_augmentation`	N	10 B
`render_augmentation` → `merge_frames`	Y	6 B
`merge_frames` → `send_image`	Y	6 B

Explain size choices for partial credit consideration.

Must hold onto an entire image from `get_image` to perform the search in `compute_viewpoint`.

Need to process entire search in `compute_viewpoint` before have a new viewpoint ($5 \times 2B = 10B$) to pass to `render_augmentation`. `render_augmentation` needs the viewpoint to process any image pixels.

As `render_augmentation` completes a pixel ($3 \times 2B = 6B$), it is ready to use, in the same order, in `merge_frames`.

As `merge_frames` completes a pixel ($3 \times 2B = 6B$), it is ready to be sent by `send_image` in the same order produced.

5. What is the critical path (latency bound) for `compute_viewpoint`?

read sintable, costable	1
multiply by sine, cos	1
add sin/cos terms	1
scale	1
(shifts for free)	0
add offset	1
read image and reference	100
subtract	1
(abs for free)	0
sum reduce	$\log_2(2048 \times 1024 \times 3) = 23$
min reduce	$\log_2(4^3 \times 2^2) = 8$
Total	137

full points for basic idea, including long memory, just a few serial, and reduces ... should get small number < 300

-4 for getting reduces wrong.

-1 for overlooking memory read; -3 for putting memories in series.

-3 for missing other data parallel operations.

(This page intentionally left mostly blank for answers.)

6. Rewrite the body of `compute_viewpoint` to minimize the memory resource bound by exploiting the scratchpad memory and the 64MB on-chip memory and streaming memory operations.

   - Annotate what arrays live in the local scratchpad
   - Account for total memory usage in the local scratchpad (use provided table)
   - Describe how you modify the code
     - You do not need to rewrite the entire function, but you can use code snippets as necessary to clarify your answer.
     - Use **for** loops that only copy data to denote the streaming operations
   - Estimate the new memory resource bound for your optimized `compute_viewpoint`.

Variable	Size (Bytes)
image_line[WIDTH][COLORS]	$2048 \times 3 \times 2 = 12,288$
sintable[360]	720
costable[360]	720
old[5]	10
current[5]	10

Put a copy of `reference` in `uint16_t ref_copy[HEIGHT][WIDTH][COLC` (12MB) in 64MB on-chip memory

Copy reference image into 64MB on-chip memory at beginning of function and operate on it from there.

Copy each line ($2048 \times 3 \times 2$B) into image_line in the body of F before starting G. All references to image[iy][ix] now go to image_line.

Common Problem: `reference` is accessed randomly. A line buffer will not work for it.

(This page intentionally left mostly blank for answers.)

New memory resource bound:

$$\frac{2048\times1024\times3\times2}{16}$$
$$+\,4^3\times2^2\times1024\times\frac{2048\times3\times2}{16}$$
$$+\,4^3\times2^2\times2048\times1024\times(10+1)$$
$$=6.1\times10^9$$

Roughly: 10 points for describing correct strategy that will get reasonable speedup; 5 points for summarize memory usage; 5 points for resource bound.

Strategy should get at least 4× speedup and not exceed stated memory capacities. Otherwise no greater than 10 points total.

Similarly, strategy based on incorrect assumption (like can use line-buffer for reference) gets at most 10 points total.

7. Considering a custom hardware accelerator implementation for `compute_viewpoint` where you are designing both the compute operators and the associated memory architecture. How would you use loop unrolling and array partitioning to achieve guaranteed throughput of 30 frames per second of throughput.

   Make the (probably unreasonable) assumption that reads from these memories can be completed in one cycle.

   Start by assuming we unroll H; we need to understand how much unrolling of the rest of the loops is required. Since the loops are associative reduce, the inner loop can be pipelined to II=1. $\frac{4^3 \times 2^2 \times 2048 \times 1024}{A \times 10^9} \leq \frac{1}{30}$, giving us A a little over 16. This suggests unrolling about a factor of 32 beyond H will be sufficient.

   Smaller unroll factors 17–32 are acceptable. There's just a question of how well data is distributed.

   Common Problem: Not accounting for the operations that can be pipelined.

   (a) Unrolling for each loop?

Loop	Unroll Factor
A	1
B	1
C	1
D	1
E	1
F	1
G	32
H	3

   (b) For the unrolling, how many multipliers and adders?

Multipliers	$6 \times 32 = 192$
Adders	$32 \times (4 + 3 \times 2) = 320$

   Grade for consistency with answer to (a)

(c) Array partitioning for each array?

Note: blank rows left for local arrays you may have added when optimizing memory in Question 6.

Array	Array Partition	Ports	Width	Depth/partition
old[]	none	1	16	10
current[]	none	1	16	10
sintable[]	none	1	16	360
costable[]	none	1	16	360
image[]	n/a			
reference[]	n/a			
image_line[]	cyclic 32 dim 1, x complete dim 2 (and pack), c	1	48	64
ref_tmp[]	none	32	48	2,097,152

Common Problem: `reference` needs ports rather than partitioning since it is accessed randomly.

2 points for old/current/sintable/costable

3 points for image/image_line; 3 points for reference/ref_temp

partial credit within each group.

8. VLIW: Define the composition of a custom VLIW datapath for `render_augmentation` loop L achieving an II of 1.

   Assume:

   - Monlithic register file supporting all operators and memories.
   - The memory is wide enough so the color/mask dimension in overlay[][] and image[][] can be packed into a single memory operation.
   - Here, since we're handling the VLIW directly, we do need to consider looping and indexing.

   An equivalent statement of Loop L showing loop, conditional, indexing, and wide memory operations is:

```
int ix=0;
uint16_t *iaddr_base=image; // no instruction cost
uint16_t *oaddr_base=overlay; // no instruction cost
#define MASK48 ((1<<48)-1);
while (ix<WIDTH) // loop L
  {
      uint16_t tx=((ix*cr+iy*sr)*xs)>>22+x;
      uint16_t ty=((ix*sr+iy*cr)*ys)>>22+y;
      uint16_t oaddr=oaddr_base+(ty*WIDTH+tx)*4;
      uint16_t iaddr=iaddr_base+(ty*WIDTH+tx)*3;
      uint64_t oval=*((uint64_t *)oaddr);
      int tcnd=((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
        && ((oval>>48)>0));
      oval=oval&MASK48;
      *((uint48_t *)iaddr=(oval&tcnd)|(*((uint48_t *)iaddr)&~tcnd);
      ix++;
   }
```

   (a) How many operators of each type so the Resource Bound II is 1.

Operator	Inputs	Outputs	**Number**
incrementers/decrementers	1	1	1
ALU (includes |, &, &&, +, - , ×, ˜ , >>, >, <, >=, <=, ==)	2	1	34
ports to memory containing overlay[]	2	1	1
ports to memory containing image[]	2	1	2
branch units	1	0	1 or 2

   take anything 25–40 for ALU

(b) What is the latency of the loop L body? Identify Critical Path and give length.

Critical Path 18

i. ix*cr, iy*sr, ix*sr, iy*cr, conditional branch
ii. + for terms above, ix++
iii. *xs, *ys, ix¡WIDTH
iv. >>
v. +x, +y
vi. tx, ty comparisons, ty*WIDTH
vii. && combine tx compares, && combine ty compares, +tx
viii. && combine tx and ty components, *4, *3
ix. add iaddr_base, oaddr_base
x. oaddr, iaddr dereference
xi. oval>>48, oval&MASK48
xii. compare for shifted ovall
xiii. && finish tcnd
xiv. ˜ tcnd, oval & tcnd
xv. & for ˜ tcnd
xvi. |
xvii. iaddr writeback
xviii. branch

Solution with single conditional branch at end back to top; that's one that only requires one branch unit. That solution probably works better with II=1 software pipeline.

Mostly looking for them to see path is deep and understand basic dependencies. Path of > 10 with plausible dependencies ok for full credit.

(c) Can you schedule to achieve the resource bound II of 1? Why or why not?

Yes. Loops K, L are data parallel. There is no dependence between loop iterations. There are no cycles in the flow graph. Software pipeline the loop across multiple iterations to get II of 1.

Code of Academic Integrity

Since the University is an academic community, its fundamental purpose is the pursuit of knowledge. Essential to the success of this educational mission is a commitment to the principles of academic integrity. Every member of the University community is responsible for upholding the highest standards of honesty at all times. Students, as members of the community, are also responsible for adhering to the principles and spirit of the following Code of Academic Integrity.*

Academic Dishonesty Definitions

Activities that have the effect or intention of interfering with education, pursuit of knowledge, or fair evaluation of a student's performance are prohibited. Examples of such activities include but are not limited to the following definitions:

A. Cheating Using or attempting to use unauthorized assistance, material, or study aids in examinations or other academic work or preventing, or attempting to prevent, another from using authorized assistance, material, or study aids. Example: using a cheat sheet in a quiz or exam, altering a graded exam and resubmitting it for a better grade, etc.

B. Plagiarism Using the ideas, data, or language of another without specific or proper acknowledgment. Example: copying another person's paper, article, or computer work and submitting it for an assignment, cloning someone else's ideas without attribution, failing to use quotation marks where appropriate, etc.

C. Fabrication Submitting contrived or altered information in any academic exercise. Example: making up data for an experiment, fudging data, citing nonexistent articles, contriving sources, etc.

D. Multiple Submissions Multiple submissions: submitting, without prior permission, any work submitted to fulfill another academic requirement.

E. Misrepresentation of academic records Misrepresentation of academic records: misrepresenting or tampering with or attempting to tamper with any portion of a student's transcripts or academic record, either before or after coming to the University of Pennsylvania. Example: forging a change of grade slip, tampering with computer records, falsifying academic information on one's resume, etc.

F. Facilitating Academic Dishonesty Knowingly helping or attempting to help another violate any provision of the Code. Example: working together on a take-home exam, etc.

G. Unfair Advantage Attempting to gain unauthorized advantage over fellow students in an academic exercise. Example: gaining or providing unauthorized access to examination materials, obstructing or interfering with another student's efforts in an academic exercise, lying about a need for an extension for an exam or paper, continuing to write even when time is up during an exam, destroying or keeping library materials for one's own use., etc.

* If a student is unsure whether his action(s) constitute a violation of the Code of Academic Integrity, then it is that student's responsibility to consult with the instructor to clarify any ambiguities.