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:

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

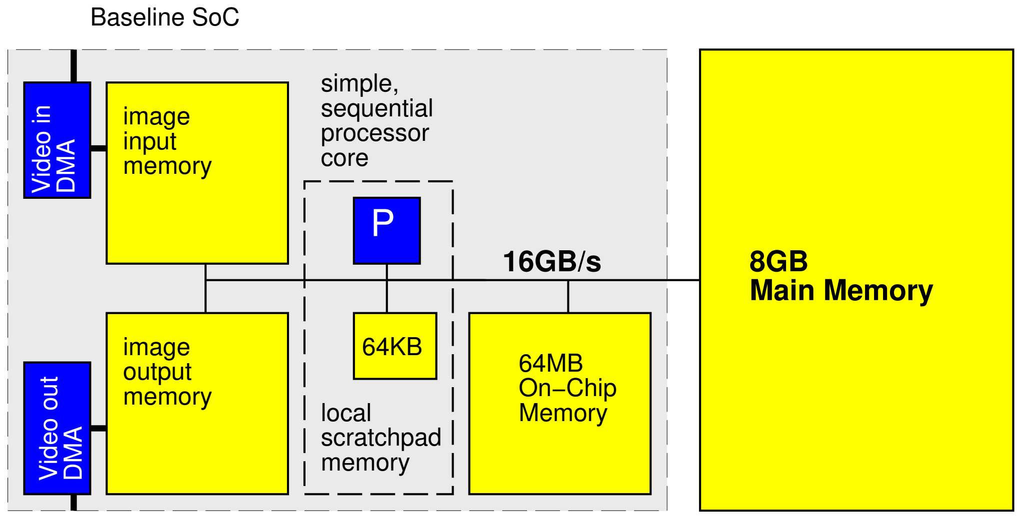
```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t 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`.

loop	Compute	Memory
`get_image`		
`compute_viewpoint`		
`render_augmentation`		
`merge_image`		
`send_image`		
`augment_frame`		

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`

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

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	
F	Data Parallel Reduce Sequential	
K	Data Parallel Reduce Sequential	
N	Data Parallel Reduce Sequential	
Z	Data Parallel Reduce Sequential	

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.)

Function Pair	(a) Same Image?	(b) Size (bytes)
`get_image` → `compute_viewpoint`		
`compute_viewpoint`→`render_augmentation`		
`render_augmentation` → `merge_frames`		
`merge_frames` → `send_image`		

Explain size choices for partial credit consideration.

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

(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)

(This page intentionally left mostly blank for answers.)

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.

   (a) Unrolling for each loop?

Loop	Unroll Factor
A	
B	
C	
D	
E	
F	
G	
H	

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

Multipliers	
Adders	

(c) Array partitioning for each array?

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

Array	Replicas	Array Partition	Ports	Width	Depth per Partition (in Width words)

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	
ALU (includes |, &, &&, +, - , ×,˜ , >>, >, <, >=, <=, ==)	2	1	
ports to memory containing overlay[]	2	1	
ports to memory containing image[]	2	1	
branch units	1	0	

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

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

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.