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

ESE532, Fall 2019 | Final | Thursday, December 19

- Exam ends at 11:00AM; begin as instructed (target 9:00AM).
  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.
- 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	2	3	4	5	6	7	8a	8b	8c	8d	Total
10	10	10	10	10	10	10	10	2	8	10	100

Average 59, Std. Dev. 14

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

```
code_one.c        Fri Dec 20 10:14:13 2019        1

int WIDTH 4096
int HEIGHT 2048
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 1
int YOFF 1
int ROT 1
int XSCALE 2
int XFACT 2 // typo for XSFACT in original
int XSFACT 2
int YSCALE 2
int YSFACT 2 // typo for YSFACT in original
int YFACT 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        Mon Dec 23 18:18:34 2019        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;i<HEIGHT;iy++) // loop F
                for (int ix=0;i<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;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<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   Thu Dec 19 05:26:56 2019   1

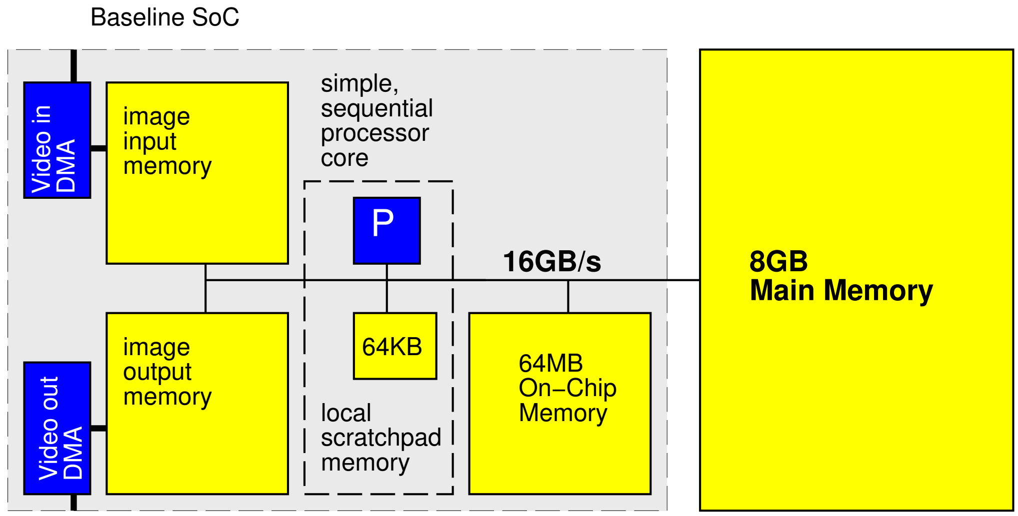
```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
      for (int ix=0;i<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;i<HEIGHT;iy++)
    for (int ix=0;i<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;i<HEIGHT;iy++)
    for (int ix=0;i<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 throughout, 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 indecies.)
- 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 2048B. 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 2048B.

1. Simple, Single Processor Resource Bounds

   (a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

   (b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096\times2048\times3\times2}{16\times10^9}+$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031
Total		59 s

0.017 frames/second

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

   (a) What function 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?

$\frac{64.9}{5.4} = 11$

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.

A is a min-reduce on best_score.

F is a sum-reduce for score.

Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

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	48 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 ($5 \times 2\text{B} = 10\text{B}$) to pass to **`render_augmentation`**. **`render_augmentation`** needs the viewpoint to process any image pixels.

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

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

5. Latency Bound

   (a) What is the critical path (latency bound) for the entire computation as captured in the `augment_frame` function?

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	$\log_2(4096 \times 2048 \times 3) = 25$
	min reduce	$\log_2(2^5) = 5$
render_ augmentation	read sintable, costable	1
	multiply by sine, cos	1
	add sin/cos terms	1
	scale	1
	(shifts for free)	0
	add offset	1
	read overlay	100
	(don't write image, just use it below)	0
merge_ frames	compute tx, ty with above	0
	(reads, if needed, happen with overlay above)	0
	subtract	1
	(abs for free)	0
	sum reduce	$\log_2(3) = 2$
	(don't write image, just use for output)	0
Total		244

(b) What is the latency bound Iteration Internal (II) for the `main` computation? (Hint: builds on part (a).)

136

Only need to compute new viewpoint.

6. Consider rewriting the body of `compute_viewpoint` to minimize the memory resource bound by exploiting the scratchpad memory and the 64 MB on-chip memory and streaming data tranfers.

   (a) Identify new temporary arrays allocated to scratchpad memory or 64MB on-chip memory (and specify which memory each new array is in).

```
uint16_t image_line[WIDTH][COLORS]; // scratchpad
uint16_t ref_copy[HEIGHT][WIDTH][COLORS]; // in 64MB on-chip memory
```

   (b) Describe how you use these arrays.

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

   Copy each line ($4096 \times 3 \times 2$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.

   (c) Account for total memory usage in the local scratchpad.

   24KB in image_line; 1440B in sintable and costable; 20B in old and current. Less than 26KB

   (d) Estimate the new memory resource bound for your optimized `compute_viewpoint`.

$$\frac{4096\times2048\times3\times2}{16\times10^9} + 2^5 \times 2048 \times \frac{4096\times3\times2}{16\times10^9} + \frac{2^5\times4096\times2048\times(10+1)}{10^9}$$

3.1 seconds

(This page intentionally left mostly blank for answers.)

7. Consider a multiprocessor design that included N copies of a vector processor with 16 vector lanes, each operating on 16b data. This is a single-issue vector processor that can either issue one vector or one scalar operation on each cycle. Assume the loops you identifed as data parallel or reduce operation in Question 3 are perfectly vectorizable. Each vector processor requires 2 mm^2 including a local 64KB data scratchpad.

   (a) Based on computational requirements alone, how many vector processors do you need to achieve a 30 frame per second frame rate? [for this problem, ignore memory and communication]

   82

   (b) Identify how the processors are used.

   Everything that takes significant time is data parallel or an associative reduce.

`compute_viewpoint`	81
`render_augmentation`	1
`merge_frames`	(shared)

(This page intentionally left mostly blank for pagination and answers.)

8. 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 while minimizing area? Use the following area model in units of mm^2:

- n-bit counters: $n \times 10^{-5}$
- n-bit adder: $n \times 10^{-5}$
- 16×16 multiplier: 2.5×10^{-3}
- p-port, w-bit wide memory holding d words: $w(1+p)(d+6) \times 10^{-7}$

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. $\frac{2^5\times4096\times2048}{A\times10^9} \leq \frac{1}{30}$, giving us A a little over 8. This suggests unrolling about a factor of 16 beyond H will be sufficient.

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	16
H	3

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

Multipliers	$6 \times 16 = 96$
Adders	$16 \times (4+3\times 2) = 160$

(note: for upcoming area calculation, you will need to break down adders by size.)

64b: 3×16=48, 16b: $(3+4) \times 7 = 112$

(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 16 dim 1, x complete dim 2 (and pack), c	1	48	256
ref_tmp[]	none	16	48	8,388,608

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

(d) Estimate the area for the accelerator.

Resource	Count	Area/resource	Area
16×16 multipliers	96	2.5×10^{-3}	0.24
64b-adders	48	64×10^{-5}	0.03072
16b-adders	112	16×10^{-5}	0.01792
3b-counters	5	3×10^{-5}	15×10^{-5}
8b-counters	1	8×10^{-5}	8×10^{-5}
12b-counters	1	12×10^{-5}	12×10^{-5}
memory (1,16,10)	2	5.1×10^{-5}	1.0×10^{-4}
memory (1,16,360)	2	1.2×10^{-3}	2.4×10^{-3}
memory (1,48,256)	16	2.5×10^{-3}	0.0402
memory(16,48,8388608)	1	680	680
Total			680

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 students 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 persons paper, article, or computer work and submitting it for an assignment, cloning someone elses 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 students 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 ones 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 students 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 ones 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 students responsibility to consult with the instructor to clarify any ambiguities.