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

ESE532, Fall 2021 — Final — Thursday, December 16

- Exam ends at 9:00AM; begin as instructed (target 11:00AM)
  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. 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	6a	6b	6c	6d	7	Total
10	5	5	10	10	20	10	5	10	5	10	100

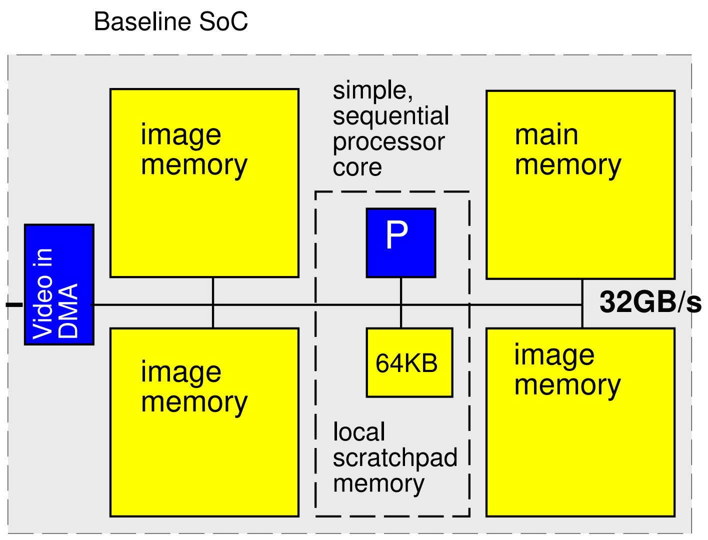
Average 61, Std. Dev. 14

Consider ...

```
#include<stdint.h>
#include<stdlib.h>
#include<stdbool.h>

#define HEIGHT 2048
#define WIDTH 2048
#define M 16
#define BH 16
#define BW 16
#define MATCH_BLOCK_THRESHOLD 8
#define SEND_BLOCK_THRESHOLD 8
#define MAX_MATCH_COST (BH*BW*1<<16)

#define END_MOTION_BLOCKS (1<<15)
#define END_DISCRETE_PIXELS ((1<<15)|1)

uint16_t current[HEIGHT][WIDTH]; // in an image memory
uint16_t previous[HEIGHT][WIDTH]; // in an image memory
// by default these live in main memory
uint16_t best_move_by[HEIGHT/BH][WIDTH/BW];
uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW];
uint16_t best_move_cost[HEIGHT/BH][WIDTH/BW];
bool sent[HEIGHT][WIDTH]; // assume packed densely into words in memory
   // so only uses HEIGHT*WIDTH/8 bytes of memory

void write_compressed(uint16_t value); // treat like .write on stream<uint16_t> *
// -- takes one cycle; account as memory operation

void get_image(uint16_t from_img[HEIGHT][WIDTH]); // assume take negligble time
   // changes pointers to reassign which memory holds which image

void update_previous(uint16_t from_img[HEIGHT][WIDTH],
                     uint16_t to_img[HEIGHT][WIDTH]);
   // changes pointers to reassign which memory holds which image

void compress_and_send(uint16_t previous[HEIGHT][WIDTH],
                       uint16_t current[HEIGHT][WIDTH]);
   // see next page

int main()
{
  while(true)
    {
      get_image(current); // assume comes form camera via DMA -- no time for this rou
      compress_and_send(previous,current);
      update_previous(previous,current);
    }
}
```

```
void compress_and_send(uint16_t previous[HEIGHT][WIDTH],
                       uint16_t current[HEIGHT][WIDTH]) {
 for (int ih=0;ih<HEIGHT;ih+=BH) // loop A
   for (int iw=0;iw<WIDTH;iw+=BW) // loop B
     {
       uint16_t best_offset_x=0;
       uint16_t best_offset_y=0;
       uint32_t best_offset_cost=MAX_MATCH_COST;
       // range adjustment to deal with out-of-bound references omitted for simplicit
       for(int voffset=-M;voffset<M;voffset++) // loop C
         for(int hoffset=-M;hoffset<M;hoffset++) // loop D
           {
             uint32_t cost=0;
             for(int by=0;by<BH;by++) // loop E
               for(int bx=0;bx<BW;bx++) // loop F
                 cost+=abs(current[ih+voffset+by][iw+hoffset+bx]
                           -previous[ih+by][iw+bx]);
             if (cost<best_offset_cost) {
                 best_offset_y=voffset; best_offset_x=hoffset;
                 best_offset_cost=cost;
               }
           }
       best_move_by[ih/BH][iw/BW]=best_offset_y;
       best_move_bx[ih/BH][iw/BW]=best_offset_x;
       if (best_offset_cost<MATCH_BLOCK_THRESHOLD)
         best_move_cost[ih/BH][iw/BW]=best_offset_cost;
       else
         best_move_cost[ih/BH][iw/BW]=MATCH_BLOCK_THRESHOLD+1;
   }

 for(int y=0;y<HEIGHT;y++) // loop G
   for(int x=0;x<WIDTH;x++) // loop H
      sent[y][x]=false; // assume runs at data streaming rate

 for (int ih=0;ih<HEIGHT;ih+=BH) // loop I
   for (int iw=0;iw<WIDTH;iw+=BW) // loop J
     {
       if (best_move_cost[ih/BH][iw/BW]<MATCH_BLOCK_THRESHOLD)  {
           uint16_t toadd=0;
           for(int by=0;by<BH;by++) // loop K
             for(int bx=0;bx<BW;bx++) // loop L
               if (sent[ih+by+best_move_by[ih/BH][iw/BW]]
                       [iw+bx+best_move_bx[ih/BH][iw/BW]]==false) toadd++;
           if (toadd>SEND_BLOCK_THRESHOLD) {
               write_compressed(ih); write_compressed(iw);
               write_compressed(best_move_by[ih/BH][iw/BW]);
               write_compressed(best_move_bx[ih/BH][iw/BW]);
               for(int by=0;by<BH;by++) // loop M
                 for(int bx=0;bx<BW;bx++) // loop N
                   sent[ih+by][iw+bx]=true; // ERROR -- should be with best_move offs
             }
         }
     }
 write_compressed(END_MOTION_BLOCKS);
 for (int y=0;y<HEIGHT;y++) // loop O
   for (int x=0;x<WIDTH;y++) // loop P
     if (sent[y][x]==false) {
         write_compressed(y); write_compressed(x);
         write_compressed(current[y][x]);
        }
 write_compressed(END_DISCRETE_PIXELS);
}
```

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, min, max, abs, divides, 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, 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 processor can execute one multiply, divide, compare, min, max, abs, or add per cycle and runs at 1 GHz.
- Data can be transfered from main memory and each of the 8MB image memories at 32 GB/s when streamed in chunks of at least 96B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to the main memory and each of the 8MB image memories takes 10 cycles.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 32 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- By default, all arrays live in the main memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, min, max, divide 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.
- Data can be transfered to accelerator local memory at the same 32 GB/s when streamed in chunks of at least 96B.
- image arrays (`current`, `previous`, one for input before becomes current) live in image memories; role of memories is changed each iteration using `get_image` and `update_previous` using a double-buffer technique.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for compute operations and memory access for outer loops inside `compress_and_send`.
   (Treat `write_compressed` cycle as a memory operation.)

loop	Compute	Memory
A	$128^2 \times 32^2 \times 16^2 \times 3$ $= 12{,}884{,}901{,}888$	$128^2 \times 32^2 \times 16^2 \times 2 \times 10$ $+128^2 \times 3 \times 10$ $= 85{,}899{,}837{,}440$
G	0 =0	$\frac{2048^2/8}{32\text{GB/s}}$ =16,384
I	$128^2 \times 16^2 \times 2$ $= 8{,}388{,}608$	$128^2 \times 16^2 \times 3 \times 10$ $+128^2 \times (2 \times 10 + 4)$ $+128^2 \times 16^2 \times 10$ =168,165,376
O	0 =0	$2048^2 \times (2 \times 10 + 3)$ =96,468,992
compress_and_send	12,893,290,496 13 seconds	86,164,488,192 86 seconds

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

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

   [✓ A]

   G

   I

   O

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

   $\frac{99{,}057{,}778{,}688}{273{,}039{,}360} = 362 \approx 360$

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/B	[✓ Data Parallel] Reduce Sequential	each block is independent
C/D	Data Parallel [✓ Reduce] Sequential	sum reduce across block at window offset
E/F	Data Parallel [✓ Reduce] Sequential	Min reduce across each offset
I/J	Data Parallel Reduce [✓ Sequential]	each iteration depends on **`sent`** as modified in previous iteration
K/L	Data Parallel [✓ Reduce] Sequential	sum reduce across each block to count pixels sent

4. What is the critical path `compress_and_send`?

AB	save block_move_X	10
CD	min reduce	$\log_2(32^2) = 10$
EF	read current, previous	10
	subtract, abs	2
	sum reduce	$\log_2(16^2) = 8$
GH	(run in parallel with above)	0
KL	best move reads (can do all at once)	10
IJ	(blocks sequentialized: 128^2) KL **`sent`** reads (all reads for block simultaneous) KL **`sent`** sum reduce for block MN **`sent`** writes (all simultaneous)	$128^2\times$ $(10$ $+\log_2(16^2)$ $+10)$ =458,752
	MN stream writes (run in parallel with **`sent`** write)	0
OP	read **`sent`**, **`current`** in parallel	10
	stream writes	$2048^2 \times 3 = 12,582,912$
Total		13,041,714

(This page intentionally left mostly blank for answers.)

5. Rewrite the body of `compress_and_send` to minimize the memory resource bound by exploiting the scratchpad 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)
   - Provide your modifications to the code.
     - Use **for** loops that only copy data to denote the streaming operations
   - Estimate the new memory resource bound for your optimized `compress_and_send`.

Variable	Size (Bytes)
match_block	$16 \times 16 \times 2 = 512$
search_window	$(16 \times 3)^2 \times 2 = 4,608$

`current` and **`previous`** reads reduce from 10 to 1, so first term for A becomes: $128^2 \times 32^2 \times 16^2 \times 2 \times 1 = 8,589,934,592$.

Since the search window is $48 \times 2 = 96$ bytes wide, we can stream the rows. Add in streaming for **`search_window`**: $128^3 \times \frac{48^2 \times 2}{32}$=2,359,296.

Since the match blocks are only $16 \times 2 = 32$ bytes, they cannot be streamed. Add in **`match_block`** reads: $128^3 \times 16^2 \times 10 = 41,943,040$.

Or make match_block larger (48×48) ... can stream. That comes out faster.

Leave G, I, O unchanged (also best_cost part of A):
491,510+16,384+168,165,376+96,468,992

Memory Resource Bound: 8,899,379,200

(This page intentionally left mostly blank for answers.)

```
void compress_and_send(uint16_t previous[HEIGHT][WIDTH],
       uint16_t current[HEIGHT][WIDTH]) {
 for (int ih=0;ih<HEIGHT;ih+=BH) // loop A
   for (int iw=0;iw<WIDTH;iw+=BW) // loop B
     {
       uint16_t best_offset_x=0;
       uint16_t best_offset_y=0;
       uint32_t best_offset_cost=MAX_MATCH_COST;
       uint16_t search_window[2*M+BH][2*M+BW];
       uint16_t match_block[BH][BW];
          for(int by=0;by<BH;by++)
            for(int bx=0;bx<BW;bx++)
              match_block[by][bx]=previous[ih+by][iw+bx];
       for(int voffset=-M;voffset<M+BH;voffset++)
         for(int hoffset=-M;hoffset<M+BW;hoffset++)  // stream read
           search_window[voffset+M][hoffset+M]=current[ih+voffset][iw+hoffset];
       // range adjustment to deal with out-of-bound references omitted for    simp
       for(int voffset=0;voffset<2*M;voffset++) // loop C
         for(int hoffset=0;hoffset<2*M;hoffset++) // loop D
          {
             uint32_t cost=0;
             for(int by=0;by<BH;by++) // loop E
               for(int bx=0;bx<BW;bx++) // loop F
                 cost+=abs(search_window[voffset+by][hoffset+bx]
                       -match_block[by][bx]);
             if (cost<best_offset_cost) {
                best_offset_y=voffset-M; best_offset_x=hoffset-M;
                best_offset_cost=cost;
            }
          }
       best_move_by[ih/BH][iw/BW]=best_offset_y;
       best_move_bx[ih/BH][iw/BW]=best_offset_x;
       if (best_offset_cost<MATCH_BLOCK_THRESHOLD)
         best_move_cost[ih/BH][iw/BW]=best_offset_cost;
       else
         best_move_cost[ih/BH][iw/BW]=MATCH_BLOCK_THRESHOLD+1;
   }
```

6. Considering a custom hardware accelerator implementation for loops A–F of `compress_and_send` 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 while minimizing area?
Use the following area model in units of mm^2:

- n-bit adder or absolute value: $n \times 10^{-5}$
- 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.

(a) Unrolling for each loop?

The difference, abs, sum will be pipelined, so without unrolling this takes $128^2 \times 32^2 \times 16^2 = 4,294,967,296$ computational cycles. We need to compute it in 30ms = 30,000,000 cycles at 1GHz. So, we need to accelerate by $\frac{4{,}294{,}967{,}296}{30{,}000{,}000} \approx$ 143. Acceleration by 256 will be sufficient, which we can do by unrolling the two innermost loops.

Loop	Unroll Factor
A	1
B	1
C	1
D	1
E	16
F	16

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

Absolute Value	256
Adders	512

(c) Array partitioning for each array used in local memories in the accelerator?

Note: local arrays may be ones added when optimizing memory in Question 5. If add additional memories, describe as necessary.

Array	Array Partition	Ports	Width	Depth per Partition (in Width words)
match_block	complete	1	16b	1
search_window	cyclic 16, 16	1	16b	9

Likely need to double buffer `match_block` and `search_window` so can load next while computing current.

(d) Estimate the area for the accelerator.

$(256+512)\times 10^{-5}+256\times 16\,(1+1)\,(1+6)\times 10^{-7}+256\times 16\,(1+1)\,(9+6)\times 10^{-7}$

$=768\times 10^{-5}+57,344\times 10^{-7}+122,880\times 10^{-7}\approx 0.026\text{mm}^2$

N.B. A separate model for a register would probably make more sense, in practice, than using the nodel for a 1 deep memory.

7. 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?

   (c) What data is being transfered in each such quanta? Identify the variable, array, or portion of an array that is needed for the consuming loop to operate.

   (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)	(c) Data
`A/B → I/J`	Y	6	`best_move_{bx,by,cost}`
`I/J→O/P`	N	524,288 (8MB)	`sent` (`current`)

Explain size choices for partial credit consideration.

`best_move` processed in order and independently in A/B and I/J.

Any iteration of I/J may change `sent`. O/P needs final version of `sent` to properly execute. It's possible you could send the entire `sent` for each block to O/P to get overlap in same frame. That would be sending much more total data between the loops.

Also need to keep another copy of the image if running I/J as a separate pipeline stage on an earlier image.

(This page intentionally left mostly blank for answers.)

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.