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

ESE5320, Fall 2023 — Final — Tuesday, December 19

- 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	6a	6b	6c	7a	7b	7c	7d	Total
10	5	5	10	10	20	10	5	10	5	3	2	5	100

Consider the following code for video compression:

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

#define HEIGHT 1024
#define WIDTH 1024
#define M 32
#define BH 16
#define BW 16
#define MAX_MATCH_COST (BH*BW*1<<16)

// by default these live in main memory
uint16_t yweight[BH][BW]; // static declartion of contents not shown
uint16_t xweight[BH][BW]; // static declartion of contents not shown

// by default these live in main memory
uint16_t current[HEIGHT][WIDTH]; // in an image memory
uint16_t previous[HEIGHT][WIDTH]; // in an image memory
uint16_t transform[HEIGHT][WIDTH]; // also in an image memory
uint16_t best_move_by[HEIGHT/BH][WIDTH/BW];
uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW];

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 motion_estimate(uint16_t previous[HEIGHT][WIDTH],
                     uint16_t current[HEIGHT][WIDTH],
                     uint16_t best_move_by[HEIGHT/BH][WIDTH/BW],
                     uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW]);
   // see next page
void transform_difference(uint16_t previous[HEIGHT][WIDTH],
                          uint16_t current[HEIGHT][WIDTH],
                          uint16_t best_move_by[HEIGHT/BH][WIDTH/BW],
                          uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW],
                          uint16_t transform[HEIGHT][WIDTH]
                          );
   // see next page
void send_difference(uint16_t transform[HEIGHT][WIDTH]);
   // see next page

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

```
void motion_estimate(uint16_t previous[HEIGHT][WIDTH],
                      uint16_t current[HEIGHT][WIDTH],
                      uint16_t best_move_by[HEIGHT/BH][WIDTH/BW],
                      uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW]) {
 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;
   }
}

void transform2d (uint16_t block[BH][BW],
                 uint16_t tblock[BH][BW]){

  uint16_t xblock[BH][BW];
  for(int by=0;by<BH;by++) // loop K
    for(int wx=0;wx<BW;wx++) // loop L
      {
        xblock[by][wx]=0;
        for(int bx=0;bx<BW;bx++) // loop M
          xblock[by][wx]+=block[by][bx]*xweight[wx][bx];
      }
  for(int bx=0;bx<BW;bx++) // loop N
    for(int wy=0;wy<BH;wy++) // loop O
      {
        tblock[wy][bx]=0;
        for(int by=0;by<BH;by++) // loop P
          tblock[wy][bx]+=block[by][bx]*yweight[wy][by];
        // should be: tblock[wy][bx]+=xblock[by][bx]*yweight[wy][by];
      }
}
```

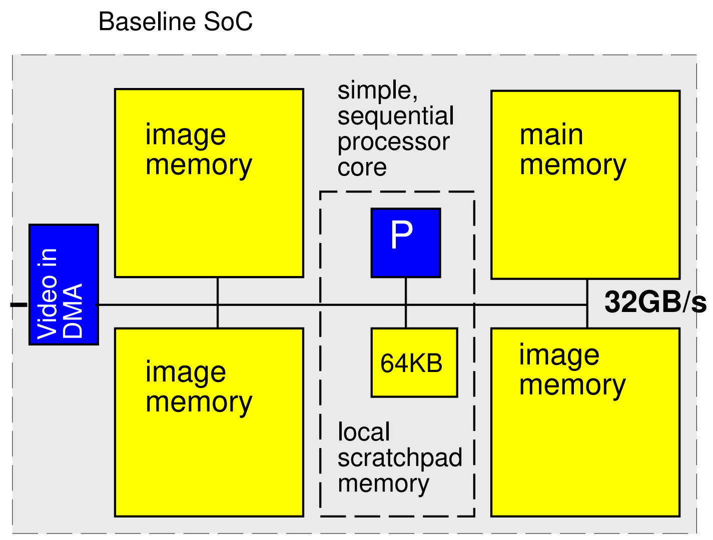
```
void transform_difference(uint16_t previous[HEIGHT][WIDTH],
                          uint16_t current[HEIGHT][WIDTH],
                          uint16_t best_move_by[HEIGHT/BH][WIDTH/BW],
                          uint16_t best_move_bx[HEIGHT/BH][WIDTH/BW],
                          uint16_t transform[HEIGHT][WIDTH]
                          ) {

 for (int ih=0;ih<HEIGHT;ih+=BH) // loop G
   for (int iw=0;iw<WIDTH;iw+=BW) // loop H
     {
       uint16_t block[BH][BW];
       uint16_t voffset=best_move_by[ih/BH][iw/BW];
       uint16_t hoffset=best_move_bx[ih/BH][iw/BW];
       for(int by=0;by<BH;by++) // loop I
         for(int bx=0;bx<BW;bx++) // loop J
           block[by][bx]=current[ih+voffset+by][iw+hoffset+bx]
             -previous[ih+by][iw+bx];
       uint16_t tblock[BH][BW];
       transform2d(block,tblock);
       for(int by=0;by<BH;by++) // loop Q
         for(int bx=0;bx<BW;bx++) // loop R
           transform[ih+by][iw+bx]=tblock[by][bx];
     }
}

void send_difference(uint16_t transform[HEIGHT][WIDTH])
{
 for (int ih=0;ih<HEIGHT;ih+=BH) // loop S
   for (int iw=0;iw<WIDTH;iw+=BW) // loop T
     {
       uint16_t count=0;
       uint16_t zzblock[BH*BW];
       uint16_t zzpos=0;
       for (int sy=0;sy<BH;sy++) // loop U
         for (int bx=0;bx<=sy;bx++) // loop V
           {
             uint16_t next=transform[ih+sy-bx][iw+bx];
             zzblock[zzpos]=next;
             if (next!=0) count=zzpos;
             zzpos++;
           }
       write_compressed(count+1);
       for (int i=0;i<(count+1);i++) // loop W
         write_compressed(zzblock[i]);
     }
}
```

(This page intentionally left blank for pagination.)

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 2MB 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 2MB 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, abs, 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, `transform`) 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 the computing components of `main`.
   (Treat `write_compressed` cycle as a memory operation.)

loop	Compute	Memory
`motion_estimate`	$64^2 \times 64^2 \times 16^2 \times 3$ $= 12{,}884{,}901{,}888$	$64^2 \times 64^2 \times 16^2 \times 2 \times 10$ $+64^2 \times 2 \times 10$ $= 85{,}899{,}427{,}840$
`transform_difference`	$64^2 \times 16^2 \times 1$ $+64^2 \times 16^3 \times 2 \times 2$ 0 $= 68{,}157{,}440$	$64^2 \times 16^2 \times 3 \times 10$ $+64^2 \times 16^3 \times 2 \times 3 \times 10$ $+64^2 \times 16^2 \times 2 \times 10$ $= 1{,}059{,}061{,}760$
`send_difference`	$64^2 \times 136 \times 2$ $=1{,}114{,}112$	$64^2 \times 136 \times (3 \times 10 + 1)$ $=17{,}268{,}736$
`main`	12,954,173,440	86,975,758,336

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

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

   [✓ `motion_estimate`]

   `transform_difference`

   `send_difference`

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

   $$\frac{12{,}954{,}173{,}440+86{,}975{,}758{,}336}{68{,}157{,}440+1{,}059{,}061{,}760+1{,}114{,}112+17{,}268{,}736} \approx 87$$

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
G/H	[✓ Data Parallel] Reduce Sequential	each block is independent
K	[✓ Data Parallel] Reduce Sequential	each xblock entry is computed independently
L	[✓ Data Parallel] Reduce Sequential	each xblock entry is computed independently
M	Data Parallel [✓ Reduce] Sequential	sum reduce across x dimension

4. What is the critical path (latency bound) `transform_difference`?

G/H	read best_move_by, best_move_bx (all parallel)	10
`transform2d`: K/L	read xweight parallel with above	0
`transform2d`: N/O	read yweight parallel with above	0
I/J	read current, previous (all parallel)	10
I/J	subtract	1
`transform2d`: K/L/M	multiply	1
`transform2d`: K/L/M	add reduce for xblock[by][wx]	4
`transform2d`: N/O/P	multiply	1
`transform2d`: N/O/P	add reduce for block[wy][bx]	4
Q/R	write transform (all parallel)	10
Total		41

(This page intentionally left mostly blank for answers.)

5. Rewrite the body of `motion_estimate` 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	$(32 \times 2 + 16)^2 \times 2 = 12,800$

`current` and **`previous`** reads reduce from 10 to 1, so first term for A becomes: $64^2 \times 64^2 \times 16^2 \times 2 \times 1 = 8,589,934,592$.

Since the search window is $80 \times 2 > 96$ bytes wide, we can stream the rows. Add in streaming for **`search_window`**: $64^2 \times \frac{80^2 \times 2}{32}$=2,359,296.

Since the match blocks are only $16 \times 2 = 32$ bytes, they cannot be streamed, but they only need to be read once. This would add in **`match_block`** reads: $64^2 \times 16^2 \times 10 = 10,485,760$. But, if we instead make match_block larger (48×48 – just read more than necessary), then can stream: $64^2 \times \frac{48^2 \times 2}{32} = 589,824$. This takes less time, so we use it.

Leave **`transform_difference`**, **`send_difference`** unchanged (also best_cost part of A):
81,920+1,059,061,760+17,268,736

Memory Resource Bound: 9,666,936,832

(This page intentionally left mostly blank for answers.)

```
void motion_estimation(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;
       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;
   }
```

6. Considering a custom hardware accelerator implementation for loops A–F of `motion_estimate` 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?

   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 $64^2 \times 64^2 \times 16^2 = 4,294,967,296$ computational cycles. We need to perform 30 of these per second. We get 10^9 cyclesa per second, so we need to compute in $\frac{10^9}{30}$ cycles. This means we need to accelerate by $\frac{4{,}294{,}967{,}296}{\frac{10^9}{30}} \approx 129$.

   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	Replicas	Array Partition	Ports	Width	Depth per Partition (in Width words)
match_block	1	complete	1	16b	1
search_window	1	cyclic 16, 16	1	16b	25

7. VLIW: Define the composition of a custom VLIW datapath for `motion_estimate` loop F achieving an II of 1, which can also be expressed as:

```
 // cptr=&(current[ih+voffset][iw+hoffset]);
 // pptr=&(previous[ih][iw]);
 // for(int yi=0;yi<BH;yi++) { // loop E
 //   cptr=cptr+WIDTH; pptr=pptr+WIDTH;
 //   xi=BW
 //   cost=0;
      while(xi>0) { // Make this loop II=1 (loop F)
        xi--;
        cptr++;
        pptr++
        cval=*cptr;
        pval=*pptr;
        diff=cval-pval;
        adiff=abs(diff)
        cost=cost+adiff;
     } // close on loop F
//    } // (close on loop E)
```

For full credit, minimize area of your implementation.
Assume:

- Monlithic register file supporting all operators and memories.
- pieces of current and previous exist in scratchpads that can be accessed in one cycle by this VLIW; (loading into those scratchpads occurs outside of loop F.)

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

Operator	Inputs	Outputs	**Number**
incrementers/decrementers	1	1	3
abs	2	1	1
ALU (includes |, &, +, - , >, <, ==	2	1	3
ports to memory containing cscratch[]	1	1	1
ports to memory containing pscratch[]	1	1	1
branch units	1	0	1

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

1	xi>0, (branch to skip), xi–, cptr++, pptr++
2	cval=*cptr; pval=*pptr
3	diff=cval-pval
4	adiff=abs(diff)
5	cost=cost+adiff

Latency=5

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

Yes. Only cycle is cost=cost+diff of length 1 (or callout that the cycle bound II is 1). So II=1 and rest is pipelineable. Or callout only dependence between loops is the cost, and cost update can be performed in one cycle.

Common Problem: Not being specific about dependencies or cycle.

(d) Provide a schedule minimizing II. Make sure schedule clearly denotes steady-state behavior and II. You do not need to show prologue and epilogue.

Operator→	Label with your selected operators									
Cycle	inc0	inc1	inc2	abs	ALU1	ALU2	ALU3	cscratch read	pscratch read	br
0	xi -4	cptr -4	pptr -4	adiff -1	xi> 0	diff - 2	cost 0	cval -3	pval -3	br -4
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										

Label cells with the variable assigned by the operation (or array entry written) and the iteration offset.

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.