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

ESE532, Fall 2018 | Midterm | Monday, October 22

- Exam ends at 11:50AM; begin as instructed (target 10:30AM)
- 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

1a	1b	2a	2b	3ab	3c	4	5a	5b	5c	6	Total
7	8	5	5	6	4	15	15	5	15	15	100

`warn_collide` is a typo for `warn_collision`.

`ostream[i]` and `tstream[i]` should be `ostream` and `tstream`.

Consider the following code to track objects and predict potential collisions.

You want to consider `warn_collision` as a separate thread that must execute at 100 frames per second. So, it is called once every 10ms with a new image. For this exam, we are only concerned with this `warn_collision` task.

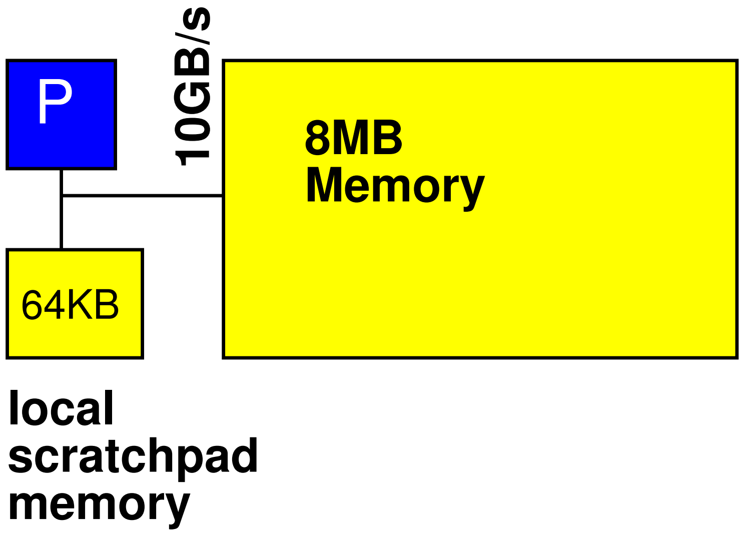
```
#define XMAX 1024
#define YMAX 1024
#define MAX_OBJECTS 1000
#define LOOKAHEAD 1000
#define XPOS 0
#define YPOS 1
#define WMIN -50
#define WMAX  50
#define MAXDIST 10000
#define OSIZE 30
uint16_t ocare[MAX_OBJECTS][OSIZE][OSIZE], oval[MAX_OBJECTS][OSIZE][OSIZE];
uint16_t dfx[MAX_OBJECTS], dfy[MAX_OBJECTS], x[MAX_OBJECTS], y[MAX_OBJECTS];
int speed[MAX_OBJECTS], angle[MAX_OBJECTS];
uint16_t collide[MAX_OBJECTS];
uint16_t collide_at_f[LOOKAHEAD][MAX_OBJECTS];
void findobj(uint16_t **image, int o, int gx, int gy, int result_xy[2]) {
     for (int y=gy+WMIN; y<gy+WMAX; y++) // Loop A
       for (int x=gx+WMIN; x<gx+WMAX; x++) { // Loop B
         int dist=MAXDIST;
         for (int i=0;i<OSIZE;i++) // Loop C
            for (int j=0;j<OSIZE;j++) // Loop D
                dist+=ocare[o][i][j]*abs(oval[o][i][j]-image[y+i][x+j]);
          if (dist<bestdist) {
               bestx=x; besty=y; bestdist=dist;
              }
          }
      result_xy[XPOS]=bestx;
      result_xy[YPOS]=besty;
  }
int distance(int x1, int x2, int y1, int y2) {
   int dx=x1-x2;
   int dy=y1-y2;
   return(sqrt(dx*dx+dy*dy));
}

// assume sin, cos, getangle, and sqrt
//   can each be performed with 20 multiplies and 20 adds;
//   critical path is 6 instructions long.
```

```
void warn_collisons(uint16_t image[XMAX][YMAX],
                    hls::stream<int> ostream, hls::stream<int> tstream) {
  for (int i=0;i<omax;i++)  // omax<MAX_OBJECTS // Loop E
     collide[i]=LOOKAHEAD;
  for (int f=0;f<LOOKAHEAD;f++) // Loop F
     for (int i=0;i<omax;i++) collide_at_f[f][i]=LOOKAHEAD; // Loop G
  int future[XMAX][YMAX];
  for (int y=0;y<YMAX;x++) // Loop H
     for (int x=0;x<XMAX;x++) future[y][x]=0; // Loop I
  for (int i=0;i<omax;i++) { // Loop J
    int newloc_guess_x=x[i]+dfx[i];
    int newloc_guess_y=y[i]+dfy[i];
    int xyloc[2];
    findobj(image,i,newloc_guess_x,newloc_guess_y,xyloc);
    speed[i]=distance(x[i],xyloc[XPOS],y[i],xyloc[YPOS]);
    angle[i]=getangle(x[i],xyloc[XPOS],y[i],xyloc[YPOS]);
    x[i]=xyloc[XPOS];
    y[i]=xyloc[YPOS];
    }
  for (int i=0;i<omax;i++) { // Loop K
     dfx[i]=speed[i]*cos(angle[i]);
     dfy[i]=speed[i]*sin(angle[i]);
     }
  for (int f=0;f<LOOKAHEAD;f++) { // Loop L
    for (int i=0;i<omax;i++) { // Loop M
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       future[fy][fx]++;
       }
    for (int i=0;i<omax;i++) { // Loop N
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       if (future[fy][fx]>1) collide_at_f[f][i]=f;
       }
    for (int i=0;i<omax;i++) { // Loop O
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       future[fy][fx]=0;
       }
     } // end of L
  for (int i=0;i<omax;i++) { // Loop P
     for (int f=0;f<LOOKAHEAD;f++) // Loop Q
        collide[i]=min(collide[i],collide_at_f[f][i]);
  for (int i=0;i<omax;i++) // Loop R
     if (collide[i]!=LOOKAHEAD)
        { ostream[i].write(i); tstream[i].write(collide[i]); }
}
```

```
// used in a pipeline like:
while(true) {
   getImage(image);
   warn_collisions(image,collision_objects,collision_times);
   steer(collision_objects,collision_times);
   }
```

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



- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds) 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 indecies.)
- Baseline processor can execute one multiply or add per cycle and runs at 1 GHz.
- Data can be transfered from the 8MB main memory at 10 GB/s when streamed in chunks of at least 256B.
- Non-streamed access to the main memory takes 5 cycles.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 10 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- By default, all arrays (image, future, ocare, oval, x, y, speed, angle, dfx, dfy, collide, collide_at_f) live in the main memory.
- Assume scalar (non-array) variables and xyloc can live in registers.
- Assume all additions are associative.
- Assume adds and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz.

1. Resource Bound

(a) Assuming the scalar processor can perform one add or one multiply on a cycle, what is the compute resource bound in seconds for the time taken by `warn_collide`?

E, F, G, H	(no adds or multiplies here)	0
J:findobj	$MAX\_OBJECTS \times (WMAX - WMIN)^2 \times OSIZE^2 \times 3$ 3 for -, *, + $1000 \times 100^2 \times 30^2 \times 3$	2.7×10^{10}
J:not findobj	$MAX\_OBJECTS \times (2 + 40 + 40 + 5)$	87,000
K	$MAX\_OBJECTS \times 2 \times (40 + 1)$	82,000
L	$LOOKAHEAD \times MAX\_OBJECTS \times (5 + 4 + 4)$	1.3×10^7
P	(no adds or multiplies here)	0
R	(no adds or multiplies here)	0
Total		2.7×10^{10}

$$\frac{2.7\times10^{10}}{10^9}=27 \text{ seconds}$$

(b) Assuming all array data located in the 8MB memory by default can be streamed at the full 10 GB/s rate, what is the memory resource bound in seconds for the time taken by `warn_collide`?

Here, we assume everything is streamed, so we count memory operations on the data in arrays.

		uint16_t
E	$MAX\_OBJECTS$	1000
F	$LOOKAHEAD \times MAX\_OBJECTS$	10^6
H	$YMAX \times XMAX$	10^6
J:findobj	$MAX\_OBJECTS \times (WMAX - WMIN)^2 \times OSIZE^2 \times 3$ 3 for oval, ocare, image $1000 \times 100^2 \times 30^2 \times 3$	2.7×10^{10}
J:not findobj	$MAX\_OBJECTS \times (10 + 2 \times 2)$ angle and speed are int instead of uint16_t, multiply by 2	14,000
K	$MAX\_OBJECTS \times 6$	6,000
L	$LOOKAHEAD \times MAX\_OBJECTS \times (6 + 6 + 5)$	1.7×10^7
P	$LOOKAHEAD \times MAX\_OBJECTS \times 3$	3×10^6
R	$MAX\_OBJECTS$	1000
Total		2.7×10^{10}

$$\frac{(2\text{Bytes/uint16\_t})\times2.7\times10^{10}\text{uint16\_t}}{10^{10}\text{GB/s}}=5.4 \text{ seconds}$$

2. Amdahl's Law

   (a) Based on resource bound estimates, which outer loop is the bottleneck?

   Circle One:

E	F	H	J	K	L	P	R

   Why?

   J contains both the most data access and the most computational operations.

   (b) Will accelerating the identified outer loop be sufficient to achieve the real-time goal of 10 ms per invocation of `warn_collide`? Explain why or why not.

No.

We can afford 10^7 cycles in the 10 ms budget.

We perform 13 M compute operations in L, which alone exceeds the budget. We perform 4 M accesses to **`future`** in L that cannot be streamed, accounting for 20 M cycles.

3. Parallelism in Loops

   (a) Classify the following loops as data parallel or not? (loop bodies could be executed concurrently)

   (b) Explain why or why not?

Loop	Data Parallel?	Why or why not?
J	Y	Computation for every object is independent.
L	Y	Computation for every future (LOOKAHEAD) time step is independent.
M	N	`future[fx][fy]` could change with every object.

   (c) Identify a loop or loop nest that performs data parallel operations followed by an associative reduce.

Loop	Reduce Operation
C,D	add
A,B	min
Q	min

M could be considered an add-reduce in a very sophisticated way. It would require effectively assuming there was a `future_for_object[YMAX][XMAX][MAX_OBJECTS]` array, computing M data parallel for each object, then performing a sum reduce over the object component for each y, x position to get the `future` array that appears in the program.

4. Focusing on the computation, and assuming unlimited computing resources, what is the latency bound for `warn_collide`? (for this problem, assume `min(a,b)` takes 1 ns just like add and multiply).

E, F, G, H	define initial values	0
J	newloc_guess (both, parallel)	1
	AB min reduce $\log_2(100^2)$	+14
	CD sum reduce $\log_2(30^2)$	+10
	D body pipeline of 3 operations	+3
	speed and angle run in parallel distance: subs in parallel, * in parallel, add, 6 for sqrt	+9
K	all objects in parallel dfx and dfy in parallel multiply, 6 for cos or sin critical path	+7
L	all iterations run concurrently	
M	fx, fy compuations all in parallel	+2
	serial future[fy][fx]++ (alternately reduce (see answer 3c))	+1000 (+10 instead)
N	`future[fy][fx]` access must wait for M to complete	
	count 1 for test/assignment	+1
O	defining initial value of `future` for loop	
P	all objects run concurrently	
Q	min reduce $\log_2(1000)$	+ 10
R	must put out in order worst case, all collide	+1000
Total	 or (reduce M case)	2057 (1067)

Latency Bound = 2.1μs (or 1.1μs)

(this page left mostly blank for pagination; can use for calculations or answers)

5. Accelerator

(a) Considering an accelerator target (e.g. FPGA mapping) for the bottleneck outer loop (Problem 2a), which of its loops need to be pipelined and unrolled to achieve the real-time goal of 10 ms per invocation of `warn_collide`? If unrolled less than completely, indicate the unroll factor.
Answer here likely assumes your solution to part (c); you will explain how you support data movement in part (c).

Loop	Unroll? (factor)	Pipeline?	II	Comments
C	Y			
D	Y			
A	N	Y	100	
B	N	Y	1	
J	N	Y	100^2	

`findobj` has 2.7×10^{10} cycles due to compute operations, but we can only afford 10^7 to meet our 10 ms target. Unrolling C and D gives us $3\times30^2-2700$ operations, to bring **`findobj`** down to 10^7 sequential cycles. Pipeline A, B allows us to start one of the unrolled C-D computation and reduces per cycle. Pipeline rest of J so can operate concurrently with **`findobj`** accelerator task.
Pipelining J outside of **`findobj`** and including in accelerator not strictly necessary. Could run as separate, concurrent thread on a processor.

(b) How many adders and multipliers does this design contain?

Multipliers	$1\times30^2+1$=901
Adders	$2\times30^2+1$=1801

+1 for computations in J not in CD. Not necessary for 2 significant figures.

(c) What memory organization do you need to support the accelerator? Assume you have 4KB, 2-port memory blocks (similar to BlockRAMs).

- What local memories do you need to allocate?
- What do these memories hold?
- What is the strategy for moving data into these local memories?
- How do these memories satisfy the data needs for the accelerator to operate as identified above? (answer may involve describing what data lives in pipeline registers as well)
- What bandwidth does this use from the main memory?

`oval` and `ocare` for a single object must be completely partitioned and placed in registers in order to support the C, D unrolling.

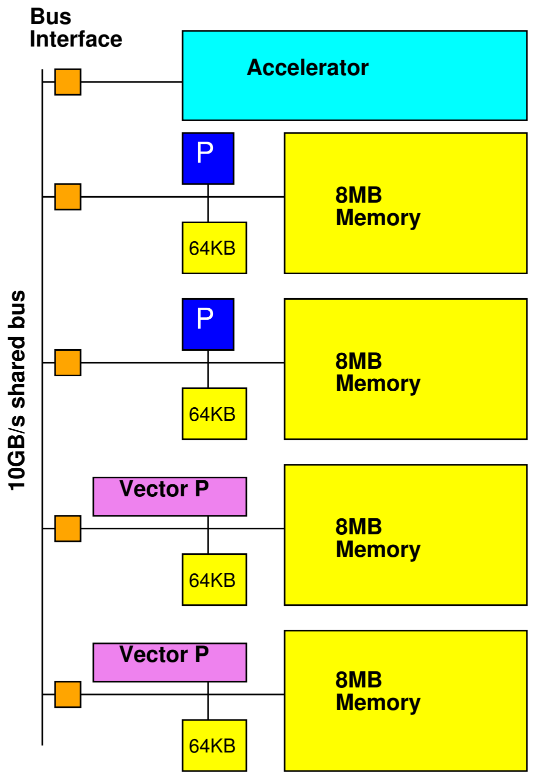
`findobj` needs a (WMAX-WMIN+OSIZE)^2 =130^2 window of the image in order to compute. We use OSIZE=30 local memories as line buffers (maybe 30-1) to hold lines reused in the C-D window filter. We use 30 length 30 shift registers to hold reused values as x increments. Together these provide the 900 image pixels that the unrolled C, D datapath needs on every cycle. With the line buffers and shift registers in place, the `findobj` accelerator needs only one image pixel per cycle during operation.

To setup a findobj call, we need to send $2 \times 2 \times 30^2$ bytes for `oval` and `ocare` and 2×130^2 for the image. 1800 bytes for each of `oval` and `ocare` are larger than 256, so streamable. Each line is $2 \times 130 = 260$ bytes, so also, independently streamable. We need a total of 3,600+33,800=37,400 bytes for every object or 37M bytes every 10 ms, which is 3.7GB/s.

To hit the 10 ms goal with this unrolling, the accelerator must run continuously. So, we must move the `ocare` and `oval` for an invocation of `findobj` during the previous invocation. This means we'll need a second set of `ocare`, `oval` registers to shift in and hold the new values during operation. Similarly, we must move the first 30 lines of the image window during the previous invocation. With care, it might be possible to use the same line buffers, but a simpler solution would be another set of line buffers for alternating invocations.

6. Assuming you use the accelerator from Problem 5 to accelerate the bottleneck loop, how do you implement the remaining loops to achieve the real-time goal of 10 ms per invocation of `warn_collide`?

   - Assume you can have any number of processors like the baseline processor, each with their own pool of 8MB of memory.
   - Assume you can have any number of vector processors that can perform 8 identical operations in a cycle, als with their own pool of 8MB of memory.
     - you can stream full vectors into the vector register file at the full streaming rate for the memory
   - Assume the vector processor is twice the size of the baseline processor.
   - Assume the communication among processors, vector processors, and accelerator is a single bus that can support one 10GB/s transfer at a time.
   - Minimize the Hardware you use.
   - Sample system with two baseline processors, two vector processors, and accelerator shown below.



   (a) Summarize the number of processor of each type you use.
   6 baseline processors

(b) Describe what loops (or portions of loops) run on which hardware.

Stragey: split L data parallel across all 6 processors. Run rest on Processor 0.

			seconds
E	Processor 0	(2×1000)/(10GB/s)	2×10^{-7}
F	All processors for own portion	(2(1000/6)×1000)/(10GB/s)	3.3×10^{-5}
H	All processors for own copy (not really executed each time)		
J	(on accelerator)		
K	Processor 0	$(82+4\times5)\times1000$	10^{-5}
	Processor 0 copy data for L	$(2\times1024\times1024+3\times2\times1000)$/(10GB/s)	0.0002
L	All processors	(4×1000)/(10GB/s) (stream x, y, dfx, dfy)	4×10^{-6}
		$(1000/6)\times1000\times(49)$ M(18): 2×5 for `future`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, * L(18): 2×5 for `future`, `collide_at_f`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, * O(13): 5 for `future`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, *	0.0082
P	All processors with per processor reduce	$(1000/6)\times1000\times(8)$ 5 `collide_at_f`, 1 for each `collide` and min	0.0014
	Processor 0 stream back per processor collide	$(5\times1000\times2)$/(10GB/s)	10^{-6}
P	Processor 0	finish collide reduce $1000\times6\times8$	0.0000048
R	Processor 0	1000×3 (`collide`, `ostream`, `tstream`)	2×10^{-6}
		Total	0.0098562

9.9 ms just under 10 ms requirement.

(c) Describe when and how data is moved (to various memories for these processors).

- Clear E, F, and H as streaming operations. Setup 256B of all zeros in the 64KB local memory (and 256B of LOOKAHEAD) and use streaming to copy over `collide`, `collide_at_f`.
- After K, stream `image`, `x`, `y`, `dfx`, `dfy` to other 5 processors.
- After L, stream per processor `collide` back to first processor.
- Within each processor, stream `x`, `y`, `dfx`, `dfy` into 64KB local memory for operation within L.

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.