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

ESE532, Spring 2017 — Final — Monday, May 1

- Exam ends at 11:00AM; begin as instructed (target 9:00AM)
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- 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

Problem 1 (35 pts) (20–30 minutes)					Problem 2 (35 pts) (40–50 minutes)				Problem 3 (30 pts) (30–40 minutes)								
a	b	c	d	e	a	b	c	d	a	b	c	d	e	f	g	h	Total
5	5	5	10	10	5	20	5	5	3	6	3	3	3	3	6	3	100

Average 67, Std. Dev. 11

1. Consider the following computation:

```
int x[256], y[256], w[256][256], s[3];

while (true) {
  for (i=0;i<256;i++) { // loop A
     x[i]=input();
     y[i]=0;
    }
  for (i=0;i<256;i++)   // loop B
   for (j=0;j<256;j++)
     y[j]+=x[i]*w[i][j];
  for (i=0;i<256;i++)   // loop C
     s[2]=max(y[i],s[2]);
     s[1]=max(s[1],s[2]);
     s[0]=max(s[0],s[1]);
  for (i=0;i<2;i++)    // loop D
     output(s[i]);
}
```

Original intent was to output all 3; typo in given problem with loop bound of 2 instead of 3.

- The initial input() provides a new input every 100 ns
- multiply is 5 ns operation, pipelineable to start one multiply every 1 ns
- local memory access (load, store) to w[][], x[], y[], s[] is 1 ns
- add and `max` are 1 ns operations
- ignore loop and indexing costs for this problem

(a) How many operations (load, store, add, max, multiply) in each labelled (A, B, C, D) for loop?

Loop	A	B	C	D
Operations	512 or 256	5×2^{16}	$4 \times 3 \times 2^8$	2 or 3

(b) Where is the bottleneck in this computation?

Loop B

(c) What is the Amdahl's law speedup if only the bottleneck is accelerated?

Assuming 100 ns is just time before guaranteed there is a next input: $\frac{5\times256+2+12}{2+12} \approx 92$

If you assume that input operation takes 100 ns: $\frac{5\times256+101+12}{101+12} \approx 12$

(d) What parallelism can be exploited in this task (both within and among loops)? Describe all applicable options where appropriate.

Loop	Parallelism Options
among loops	coarse-grained pipeline parallelism (across all)
A	(nothing really, bottlenecked on input)
B	data parallel (both task and SIMD) pipeline, VLIW
C	pipeline VLIW with software pipelining
D	could SIMD read but not really have any need

(e) Describe how you would speedup this task so that it can consume one input every 100 ns, limited only by the input rate.

- Input gives us a goal of handing 1 input per 100 ns
- Start by running each loop as a separate, coarse-grain dataflow pipeline task.
- This gives a goal of performing each (outer) loop iteration in less than 100 ns.
- Only loop B needs acceleration
  - A: Clearing y[i] can occur while waiting for next input to show up
  - C: 12 operations take 12ns, so can be pefrormed sequentially within the 100 ns cycle of the input
  - D: The output here is also not a rate limiter (needs even less than one output per 100 ns input)
- Loop B
  - Need to perform 256×5 operations every 100 ns to match the rate of the input. $\frac{256 \times 5}{100} = 12.8$ operations per nanosecond (timing on each operator).
  - So, we need to perform at least 13 operations per 1 ns cycle. Probably best to round that to 16.
  - Operations are SIMD and a multiple of 16, so a 16-lane vector unit used to accelerate the inner loop would do the trick.
  - Alternately, could break this into 16 parallel tasks, each computing a different set of 16 y[i]'s.

This page nearly blank for pagination.

(Feel free to use for answer to 1e, but it is probably not necessary.)

2. Consider the following computation:

```
int Image[1024][1024], Model[3][1024][1024], wpixel[1024][1024];
boolean mpixel[1024][1024];
for (y=0;y<1024;y++)
  for (x=0;x<1024;x++) {
     int pixel=Image[y][x];
     int M0=Model[0][y][x];
     int M1=Model[1][y][x];
     int M2=Model[2][y][x];
     mpixel[y][x]=f(pixel,M0,M1,M2); // 10 mpy, 6 adds
     int mupdate=g(pixel,M0,M1,M2); // 4 mpy, 10 adds
     int updateval=h(pixel,M0,M1,M2); // 16 mpy, 8 adds
     Model[mupdate][y][x]=updateval;
     }
for (i=0;x<1024;i++) { // ERRORs: x<1024 should be i; assignments should be wpixel
  if (mpixel[0][i]) mpixel[0][i]=1 else mpixel[0][i]=0;
  if (mpixel[i][0]) mpixel[i][0]=1 else mpixel[i][0]=0;
  }
for (y=1;y<1024;y++)
  for (x=1;x<1024;x++) {
     int imax=max(wpixel[y-1][x-1],max(wpixel[y-1][x],wpixel[y][x-1]));
     if (mpixel[y][x]) wpixel[y][x]=imax+1; else wpixel[y][x]=0;
     }
int xmax=0;
int ymax=0;
int maxval=0;
for (y=1;y<1024;y++)
  for (x=1;x<1024;x++)
     if (wpixel[y][x]>maxval) {maxval=wpixel[y][x]; xmax=x; ymax=y;}
int sy=max(0,ymax-16);
int sx=max(0,xmax-16);
for (y=sy;y<sy+16;y++)
  for (x=sx;x<sx+16;x++)
    output(Image[y][x]);
```

- Main memory is 256 M 32b ints; has a read and write latency of 100 ns, but can stream sequential data at 1 ns per cycle for blocks up to 512 words.
  `streamIn(MainAddr,LocalAddr,n)` – copy n≤512 32b ints to local memory in 100+n ns.
  `streamOut(LocalAddr,MainAddr,n)` – copy n≤512 32b ints to main memory in 100+n ns.
- Local memory is 4K 32b ints and has a read/write latency of 1 ns.
- multiply, add, max, compare each take 1 ns.
- As written Pixel, Model, mpixel, and wpixel live in main memory.
- Ignore loop and indexing costs for this problem.

(a) With no memory streaming operations or local memories,

i. estimate runtime
$2^{20}\,(6 \times 100 + 54 \times 1) + 2^{10}\,(2 \times 2 \times 100) + 2^{20}\,(5 \times 100 + 3 \times 1) + 2^{20}\,(2 \times 100 + 1) + 4 + 2^8 \times 100 \approx 1.4\text{B}$

ii. identify bottleneck (which loop? memory or compute?) and support your answer using your runtime estimate
First loop, memory

(b) Rewrite the code to localize and stream data You may combine loops where you find it beneficial.

i. Identify the local variables you define and how they are laid out in the local memory:

Address		
begin	end	Variable
0	2047	localM[0]
2048	4095	localM[1]
4096	6143	localM[2]
6144	10239	local_wpixel_row
10240	12287	localImage
12288	12291	prevx
12292		prevy
12296		prevxy
12300		curr_mpixel
12304		curr_wpixel
12308		lmpixel
12312		mupdate
12316		updateval
12320		pixel
12324		M0
12328		M1
12332		M2
12336		imax
12340		xmax
12344		ymax
12348		maxval
12352		sx
12356		sy
12360		x
12364		y

ii. Show how the code is revised to use these local variables and stream fetch operations.

```
int Image[1024][1024], Model[3][1024][1024];
int localM[3][512]; // was erroneously 1024 on original solution
int local_wpixel_row[1024];
int xmax=0;
int ymax=0;
int maxval=0;
int curr_mpixel, curr_wpixel;
int prevx, prevy, prevxy;
for (int y=0;y<1024;y++)
  for (int xb=0;xb<1024;xb+=512) {
    streamIn(&Image[y][xb],localImage,512);
    streamIn(&Model[0][y][xb],&localM[0],512);
    streamIn(&Model[1][y][xb],&localM[1],512);
    streamIn(&Model[2][y][xb],&localM[2],512);
    for (int xoff=0;xoff<512;xoff++) {
       int x=xb+xoff;
       int pixel=localImage[xoff];
       int M0=localM[0][xoff];
       int M1=localM[1][xoff];
       int M2=localM[2][xoff];
       curr_mpixel=f(pixel,M0,M1,M2); // 10 mpy, 6 adds
       int mupdate=g(pixel,M0,M1,M2); // 4 mpy, 10 adds
       int updateval=h(pixel,M0,M1,M2); // 16 mpy, 8 adds
       localM[mupdate][xoff]=updateval;
       if ((y==0) || (x==0)) {
             prevy=0;
             curr_wpixel=0;
             imax=0;
          }
       else {
          prevy=local_wpixel_row[x];
          int imax=max(prevxy,max(prevy,prevx));
          }
       if (curr_mpixel) curr_wpixel=imax+1; else curr_wpixel=0;
       local_wpixel_row[x]=curr_wpixel;
       prevx=curr_wpixel;
       prevxy=prevy;
       if (curr_wpixel>maxval) {maxval=curr_wpixel; xmax=x; ymax=y;}
       }
    streamOut(&localM[0],&Model[0][y][xb],512);
    streamOut(&localM[1],&Model[1][y][xb],512);
    streamOut(&localM[2],&Model[2][y][xb],512);
```

```
     }
int sy=max(0,ymax-16);
int sx=max(0,xmax-16);
for (int y=sy;y<sy+16;y++)
  {
     streamIn(&Image[y][x],&localImage,16);
     for (int x=sx;x<sx+16;x++)
         output(localImage[x]);
   }
```

10 points for streaming in/out first loop
10 points for optimizing rest; full points for localizing mpixel, wpixel row; 5 points if keep image size mpixel/wpixel and use streaming on those.

(c) What is the runtime of your optimized design?

$2^{20}(69) + 2^{11}(612 \times 7) + 4 + 16 \times 116 + 2^8 \times 1 \approx 81\text{M}$

(d) Where is the bottleneck now?

Computation in first loop

This page intentionally left nearly blank for pagination.
(or, additional code and calculations)

3. Consider a function from $A00$, $A01$, $A10$, $A11$, $B0$, $B1$ to $B2$, $B3$:

$$t0 = \frac{A00}{A10} \quad (1)$$
$$t1 = \frac{A01}{A11} \quad (2)$$
$$t2 = t1 * B1 \quad (3)$$
$$t3 = B0 - t2 \quad (4)$$
$$t4 = t1 * A10 \quad (5)$$
$$t5 = A00 - t4 \quad (6)$$
$$t6 = t0 * B1 \quad (7)$$
$$t7 = B0 - t6 \quad (8)$$
$$t8 = t0 * A11 \quad (9)$$
$$t9 = A01 - t8 \quad (10)$$
$$t10 = \frac{t3}{t5} \quad (11)$$
$$t11 = \frac{t7}{t9} \quad (12)$$
$$t12 = A20 * t10 \quad (13)$$
$$t13 = A21 * t11 \quad (14)$$
$$t14 = A30 * t10 \quad (15)$$
$$t15 = A31 * t11 \quad (16)$$
$$B2 = t12 + t13 \quad (17)$$
$$B3 = t14 + t15 \quad (18)$$

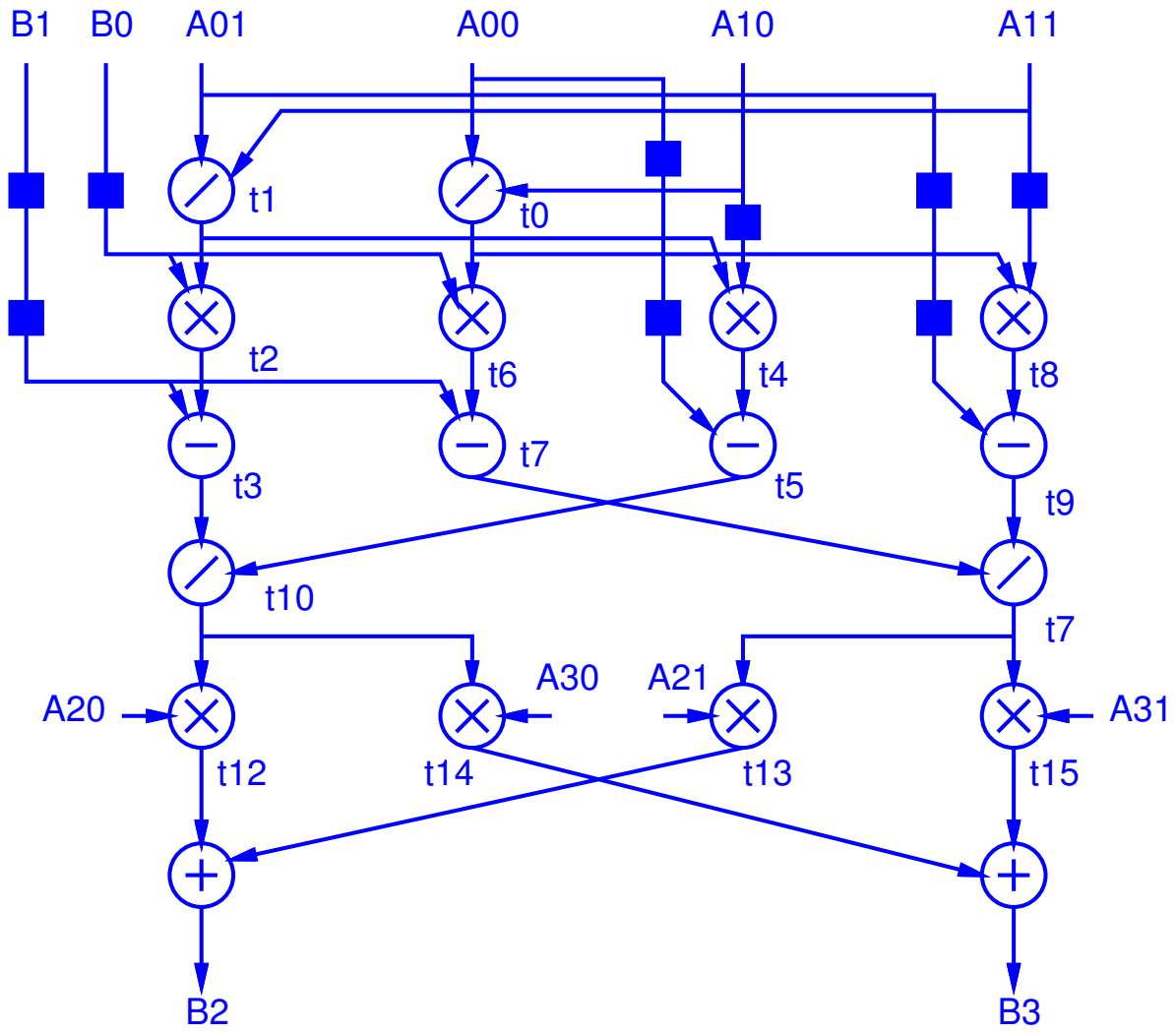
Assume:

- $A00$, $A01$, $A10$, $A11$, $B0$, $B1$ available on inputs at beginning of cycle
- output $B2$, $B3$ on designated output port
- $A20$, $A21$, $A30$, $A31$ already in operator memories; you choose which
- add/subtract, multiply, divide are single-cycle operations
- add/subtract unit costs 1 units of area
- multiply unit costs 10 units of area
- divide unit costs 10 units of area
- memory bank costs 5 units of area
- $i \times o$ crossbar costs $0.5 \cdot i \cdot o$ units of area
- word-wide pipeline register costs 0.5 units of area
- 2 or 3 input mux is 1 unit of area

(a) What is the critical path bound for this computation?

6 cycles

(b) Show a pipelined datapath for this operation.

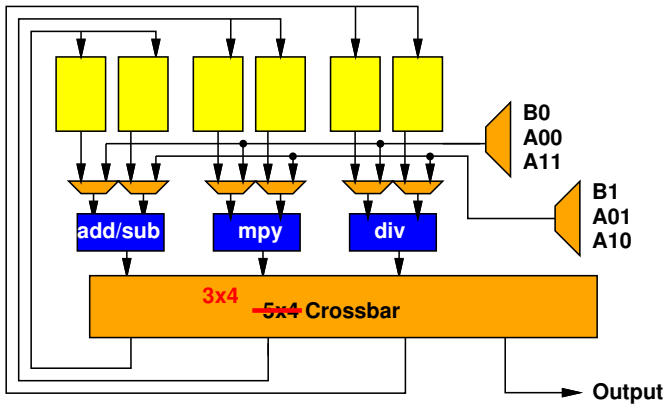


Note registers added (small squares) to balance path delays from inputs. As shown, assumes a register on the output of each operator.

(c) Estimate the area for the pipelined datapath.

$(8+4)\,10 + 1 \times 6 + \frac{9}{2} = 130.5$

Problem statement wasn't clear if there were registers associated with each operator, so may add 16 more registers for the ones associated with the operators. Maybe also add registers for A20, A21, A30, A31. So, +16 or +20 register (+8 or +10 area units) also reasonable.



(d) What is the resource bound for this computation on a VLIW datapath with a single add/subtract unit, a single multiplier, and a single divider (as shown)?

$\max\left(\frac{6}{1}, \frac{8}{1}, \frac{4}{1}\right) = 8$

(e) Schedule the computation on the VLIW datapath with a single add/subtract unit, a single multiplier, and a single divider (as shown) to minimize computation cycles.
Mark each "operator" with the variable computed on the operator on that cycle; mark each "input" with the variable being stored into the data memories on each cycle (note: only one value can be stored into the data memories associated with an operator on each cycle).

Cycle	**add/sub**		**multiply**		**divide**		**mux0**	**mux1**	**output**
	operator	input	operator	input	operator	input			
0				t1	t1		A11	A01	
1				t0	t0		A00	A10	
2		t2	t2				B0		
3		t4	t4					A10	
4	t3	t6	t6			t3	B0		
5	t5	t8	t8			t5	A00	A11	
6	t7			t10	t10	t7	B0		
7	t9	t12	t12			t9		A01	
8		t14	t14	t11	t11				
9		t13	t13						
10	B2	t15	t15						B2
11	B3								B3
12									
13									
14									
15									

(f) Estimate the area of the VLIW datapath with a single add/subtract unit, a single multiplier, and a single divider (as shown).

$5 \times 6 + 10 + 10 + 1 + 8 + \frac{3 \times 4}{2} = 65$

(g) Using no more than 100 units of area, provision a customized VLIW datapath for this unit – how many operators of each type? total area?

Operator	add/sub	mpy	div	mux2 or mux3	Area
Number	1	2	1	2 (10)	91

2 muxes as input select; 10 total including the pair for each of the 4 operators.

$5 \times 8 + 10 + 2 \times 10 + 1 + 10 + \frac{5 \times 4}{2} = 91$

(h) Justify your choice of operators.

Resource Bound $\max\left(\frac{6}{1}, \frac{8}{2}, \frac{4}{1}\right) = 6$. Multiply is the bottleneck resource, so priority to add. There is only enough space to add one operator.

A schedule would even be better, but was not expected.

Arguments about maximum parallelism available (maximum number of operators that could be usefully employed) were good, but those only tended to be applicable for designs that do not meet the area constraint (and are larger than the fully pipelined design).

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.