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

ESE532, Spring 2017 Midterm Wednesday, March 1

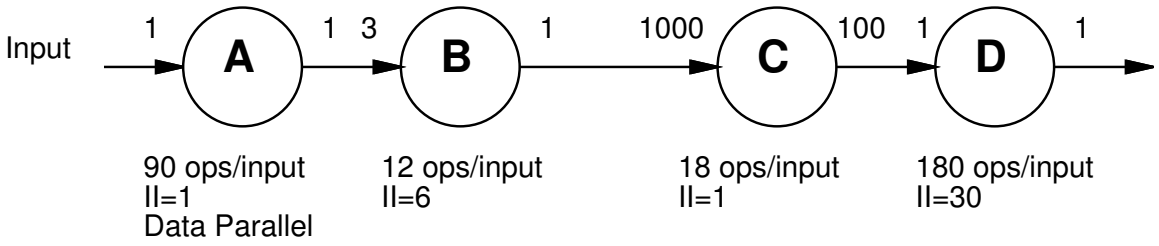
- Exam ends at 4:20PM; begin as instructed (target 3:00PM)
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- Sign Code of Academic Integrity statement (see last page for code).

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this exam.

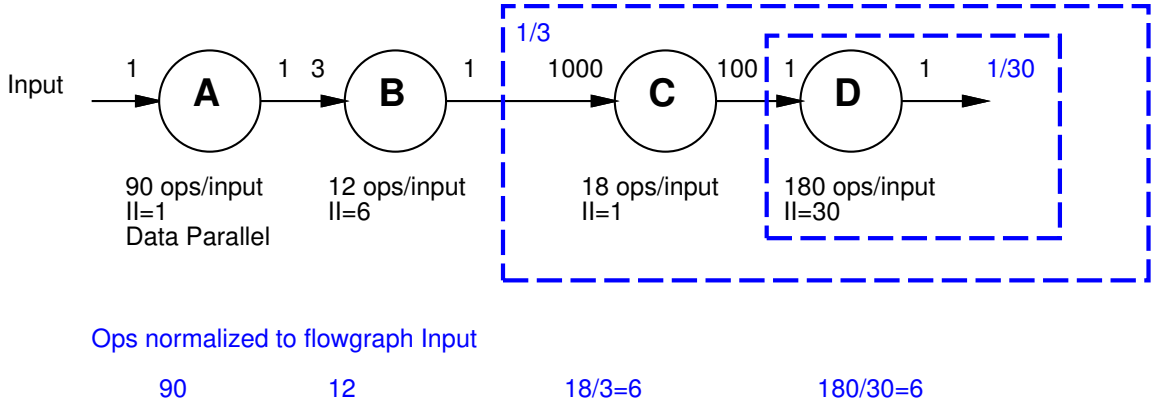
Name: Solution

Problem 1			Problem 2							
a	b	c	a	b	c	d	e	f	g	Total
10	10	10	5	10	10	10	15	15	5	100

1. Consider the Data Flow Graph



[Reminder: the consumer of an operator that consumes n inputs and produces m outputs will see only $\frac{m}{n}$ of the tokens seens by its predecessor. ops/input annotation is relative to the inputs to the operator. An operator shown consuming n inputs and executing c ops/input, will take $n \cdot c$ ops to process those n inputs.]



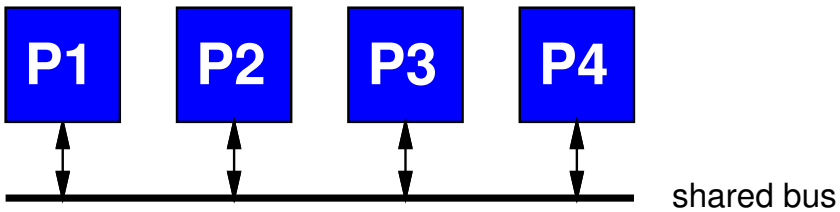
(a) Running on a single 100 MHz processor (assuming it successfully completes one of the "ops" required by a flowgraph node per cycle)

i. What is the throughput achieved in terms of inputs processed per second?

$1/(90+12+6+6) = \frac{100\text{MHz}}{114} = 0.88\text{M/s}$

ii. What is the Amdahl's Law maximum speedup possible if one could accelerate one operator (identify operator)? $\frac{114}{24} = 4.75$ (A)

(b) Running on 4 identical 100 MHz processors:



Assume processor-to-processor communication of one token takes one cycle on the shared bus.

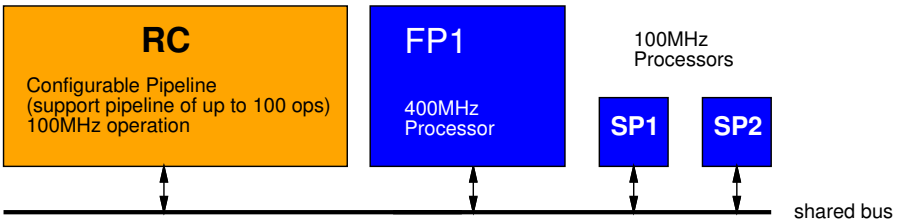
i. How would you map operators to processors?

P1	P2	P3	P4
A1	A2	A3	B, C, D

A1, A2, A3 are the result of dividing A into 3 equal pieces; this is viable since A is data parallel.

ii. Throughput achieved? $\frac{100\text{MHz}}{30}$=3.3M/s

(c) Running on a heterogeneous platform with a configurable accelerator pipeline, a fast processor, and two slower processors as shown below (one cycle transfer per token over bus):



i. How would you map operators to processors?

RC	FP1	SP1	SP2
Configurable Pipeline 100 datapath operations/cycle @ 100MHz	Processor 400 MHz	Processor 100 MHz	Processor 100 MHz
A	B	C	D

A is our bottleneck. With an II of 1 and fewer "ops" in the computation than the configurable pipeline supplies, we can build a complete pipeline and achieve a throughput of one result per cycle.
With an II of 6, the configurable pipeline will not be very useful for B. It would still take six 10 ns cycles to complete each input. We can run it on the faster processor to get it down to a throughput of one input every 30 ns.
B and C remain one input per 60 ns, and hence the rate limiting bottlenecks.

ii. Throughput achieved? $\frac{100\text{MHz}}{6}$=16.6M/s

2. Consider the following code:

```
#define WSIZE 5
#define FSIZE 1024

uint16_t window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t frame_in[FSIZE][FSIZE];
uint16_t frame_out[FSIZE][FSIZE];

for(int y=0;y<(FSIZE-WSIZE+1);y++)
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {
     frame_out[y][x]=0;
     for (int xoff=0;xoff<WSIZE;xoff++)
         for (int yoff=0;yoff<WSIZE;yoff++)
             frame_out[y][x]+=window[yoff][xoff]*frame_in[y+yoff][x+xoff];
     }
```

We run this on a system with a frame memory that is 32MB. It allows one 16b read or write at a time with a 10 ns latency and no pipelining. The single processor performs multiply-accumulate (Y=A+B*C) operations in 5 ns (unpipelined). For simplicity, assume other operations (e.g., loop management, index calculations) take no time (are dominated by the memory and multiply-accumulate operations).

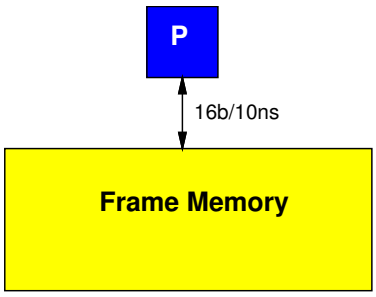
(a) How many multiply-accumulate operations need to be performed per frame?
$5 \times 5 \times 1020 \times 1020 \approx 26\text{M}$

(b) As written running on a single processor as shown.

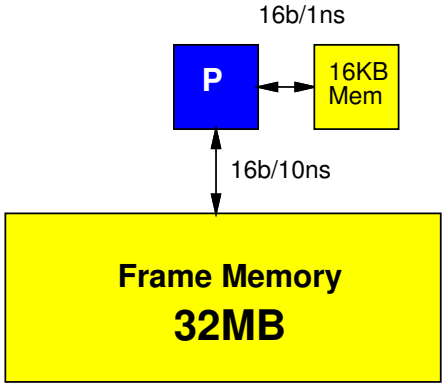
i. Assuming `window`, `frame_in` and `frame_out` live in the 32MB frame memory, how many memory operations are performed per frame?
$4 \times 5 \times 5 \times 1020 \times 1020 \approx 104\text{M}$
(1) read from window, (2) read from frame_in, (3) read from frame_out, (4) write to frame_out.
Also accepted 2 or 3.

ii. Where is the bottleneck on this single processor platform? Memory

iii. Assuming the code snippet above is called repeatedly, what is the throughput achieved in frames per second? $\frac{1}{26\text{M} \times 5\text{ns} + 104\text{M} \times 10\text{ns}} = 0.85$



(c) We add a local scratchpad memory that allows 16b access (read or written) in a 1 ns and holds 16KB of memory. We rewrite the code as shown on facing page. Variables with the `local_` are allocated to the scratchpad memory.



i. How much data must be read from the frame memory?

$$\boxed{2\text{B} \times (1024 \times 1024 + 5 \times 5) \approx 2\text{MB}}$$

ii. How many data operations are performed on the local memory?

$$\boxed{5 \times 5 + (2 \times 5 \times 5 + 1) \times 1020 \times 1020 + 1020 \times 1020 \approx 52\text{M}}$$

iii. Where is the bottleneck now?

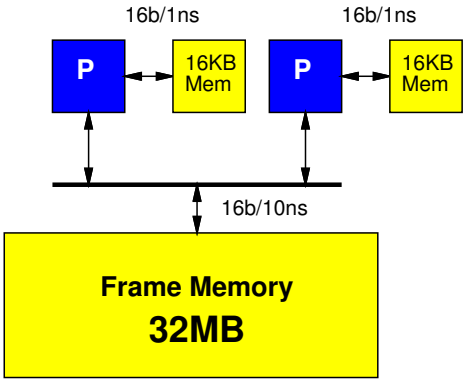
$$\boxed{\text{Computation (Multiply Accumulate Operations)}}$$

iv. Throughput achieved in frames per second?

$$\boxed{\frac{1}{26\text{M} \times 5\text{ns} + 2 \times 1\text{M} \times 10\text{ns} + 52\text{M} \times 1\text{ns}} \approx 5.0}$$

```
#define WSIZE 5
#define FSIZE 1024
uint16_t window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t local_window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t frame_in[FSIZE][FSIZE];
uint16_t frame_out[FSIZE][FSIZE];
uint16_t local_0[FSIZE];
uint16_t local_1[FSIZE];
uint16_t local_2[FSIZE];
uint16_t local_3[FSIZE];
uint16_t local_4[FSIZE];
uint16_t *local_line[WSIZE];
for (int xoff=0;xoff<WSIZE;xoff++)
  for (int yoff=0;yoff<WSIZE;yoff++)
      local_window[yoff][xoff]=window[yoff][xoff];
for(int x=0;x<(FSIZE-WSIZE+1);x++) { // ERROR -- should range to FSIZE not FSIZ
  local_0[x]=frame_in[0][x];
  local_1[x]=frame_in[1][x];
  local_2[x]=frame_in[2][x];
  local_3[x]=frame_in[3][x];
  local_4[x]=frame_in[4][x];
   }
local_line[0]=local_0;
local_line[1]=local_1;
local_line[2]=local_2;
local_line[3]=local_3;
local_line[4]=local_4;
for(int y=0;y<(FSIZE-WSIZE+1);y++) {
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {
     int tmp=0;
     for (int xoff=0;xoff<WSIZE;xoff++)
         for (int yoff=0;yoff<WSIZE;yoff++)
             tmp+=local_window[yoff][xoff]*((local_line[yoff])[x+xoff]);
     frame_out[y][x]=tmp;
   }
  uint16_t *tmp_line=local_line[0];
  local_line[0]=local_line[1];
  local_line[1]=local_line[2];
  local_line[2]=local_line[3];
  local_line[3]=local_line[4];
  local_line[4]=tmp_line;
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {// ERROR -- should range to FSIZE not FSI
    local_line[4]=frame_in[y+4][x]; // here (and above) hardcoded for WSIZE=5
    }
 }
```

(d) Given two processors as shown, how would you use both processors to accelerate the task?



i. Describe how you would divide the work between the two processors; show snippets of code that changes from part (c) as appropriate.

Give the computation of `frame_out` for y=0 to 509 to one processor and y=510 to 1019 to the second.

First processor, only change is:

```
for(int y=0;y<(FSIZE-WSIZE+1);y++) {
```

to:

```
for(int y=0;y<(FSIZE-WSIZE+1)/2;y++) {
```

Second processor, change to:

```
for(int y=(FSIZE-WSIZE+2)/2;y<(FSIZE-WSIZE+1);y++) {
```

Also need to change setup of `local_line`:

```
for(int x=0;x<(FSIZE-WSIZE+1);x++) {
  local_0[x]=frame_in[(FSIZE-WSIZE+2)/2+0][x];
  local_1[x]=frame_in[(FSIZE-WSIZE+2)/2+1][x];
  local_2[x]=frame_in[(FSIZE-WSIZE+2)/2+2][x];
  local_3[x]=frame_in[(FSIZE-WSIZE+2)/2+3][x];
  local_4[x]=frame_in[(FSIZE-WSIZE+2)/2+4][x];
   }
```

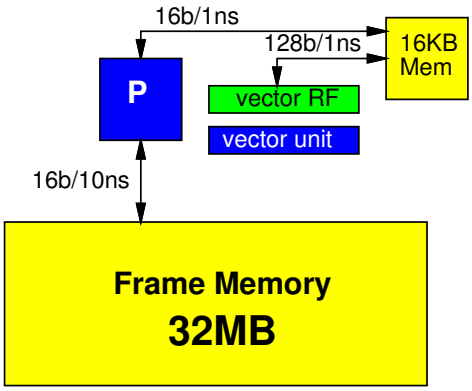
ii. Throughput achieved in frames per second?

$$\frac{1}{\frac{26}{2}\text{M}\times 5\text{ns}+2\times 1\text{M}\times 10\text{ns}+\frac{52}{2}\text{M}\times 1\text{ns}} \approx 9$$

This cuts the multiply-accumulates (bottleneck) and the local memory reads roughly in half, but doesn't change the frame memory reads. Nonetheless, accepted 2× solution in (c).

Left blank for pagination.

(e) Given a processor with a vector unit capable of processing eight 16b values in a cycle. Assume the local memory is widenned to allow transfers up to 128b of data in 1 ns (for simplicity of the problem, we will assume the memory system can support accesses that are not 128b aligned). For simplicity, we provide the following vector instruction set. All vector operations (including multiply) complete in 1 ns. The vector register file holds 16 vector registers.



- VADD sz, V1, V2, Vdst – add corresponding elements from V1 to V2 and store in Vdst (sz=16 says treat the 128b as 8 16b words)
- VMUL sz, V1, V2, Vdst – multiply corresponding elements from V1 to V2 and store in Vdst; perform operation on sz data items (sz=16 says treat the 128b as 8 16b words)
- VLD Rsrc, Vdst – load 128b at local memory addressed by Rsrc into Vdst
- VADDREDUCE sz, len, V1, Rdst – Perform a sum reduce add on the first len values in V1 and store into Rdst; sz is the values being reduced (sz=16 says treat the 128b as 8 16b words)

i. Show (on facing page) how you replace the inner two loops of the computation in (c) with vector operations. (Inner loop may not show how you setup some vector and scalar (normal processor) registers; summarize your use of registers to make the code clear.)

ii. Where is the bottleneck now?
Reading/Writing Frame Memory

iii. Throughput achieved in frames per second? $\frac{1}{15\times 1\text{M}\times 1\text{ns}+2\times 1\text{M}\times 10\text{ns}} \approx 29$

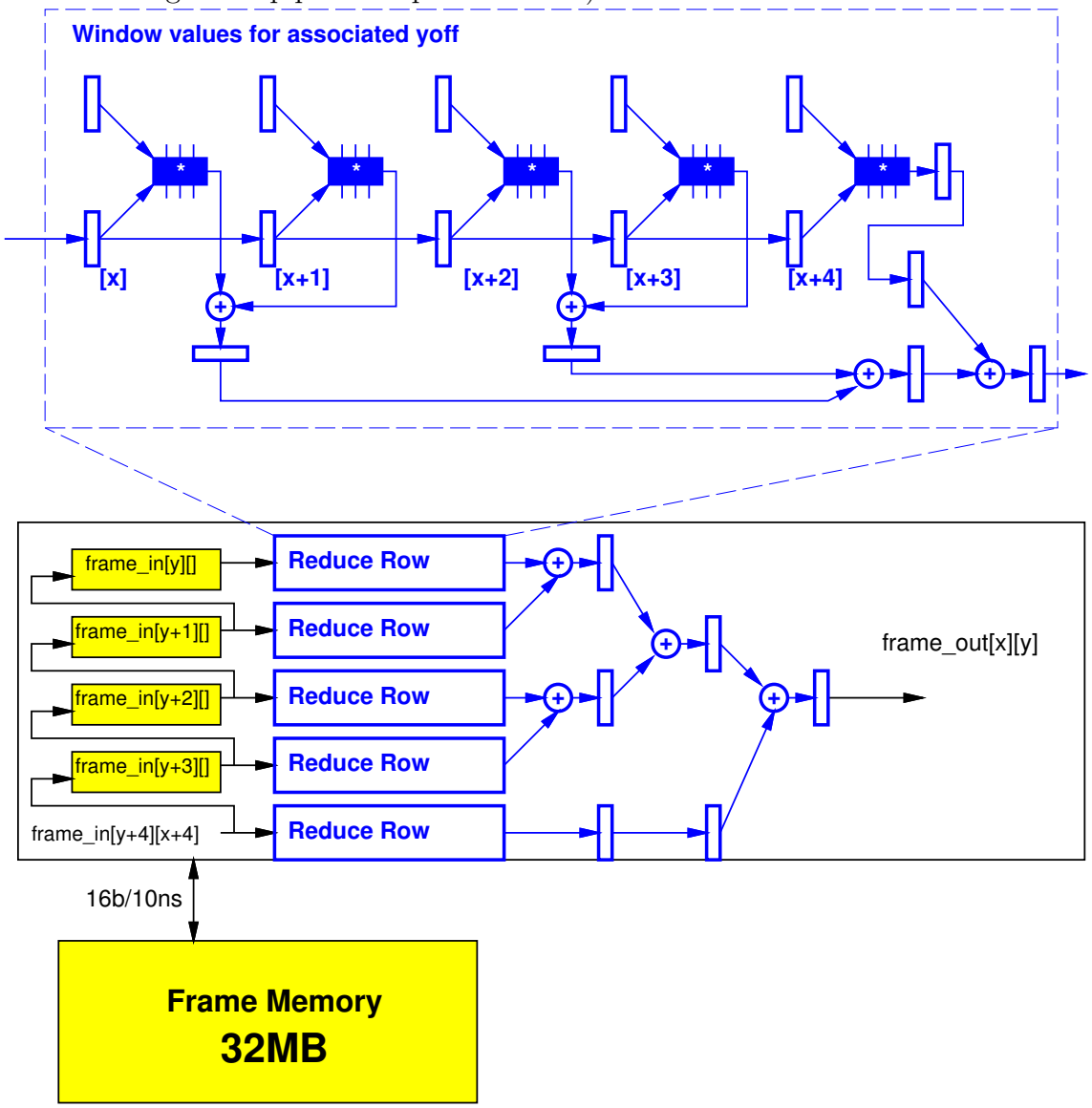
Code for Problem 2(e).i

Put `window[0][*]` in V0, `window[1][*]` in V1, ...`window[4][*]` in V4, and leave them there throughout the computation.
Put current frame pixels in V5 through V10.
Frame output value in `r3`, `frame_out` pointer in `r4`, pointers into `local_line` in `r5` through `r9`.

```
vld r5, v5
vld r6, v6
vld r7, v7
vld r8, v8
vld r9, v9
vmul 16, v0,v5,v5
vmul 16, v1,v6,v6
vmul 16, v2,v7,v7
vmul 16, v3,v8,v8
vmul 16, v4,v9,v9
vadd 16, v5,v6,v5
vadd 16, v5,v7,v5
vadd 16, v5,v8,v5
vadd 16, v5,v9,v5
vaddreduce 16, 5, v5, r3
st r3,r4 // store into frame_out
addi r4,2,r4 // advance frame_out pointer
addi r5,2,r5 // advance line pointers
addi r6,2,r6
addi r7,2,r7
addi r8,2,r8
addi r9,2,r9
```

(f) Design a hardware accelerator using building blocks with primitive 4 cycle pipelined multiply units, 1 cycle adder units, and registers operating on a 1 GHz clock.

i. Show a pipeline datapath for the inner two-loops with a pipeline II=1. Assume (as shown) there are memories configured to deliver pixels on lines y, y+1, y+2, y+3, y+4 into the pipeline. (Hierarchical schematics allowed. Make sure logic and pipeline depth are clear.)



ii. Throughput achieved in frames per second? $\boxed{50}$

iii. Latency through your pipeline (from arrival of the last pixel for a window to the output back to frame memory)? $\boxed{10\text{ns}}$

(g) Assuming we can widen the frame memory while maintaining the 10 ns cycle time, how wide does the memory need to be so that memory operations on the large memory is not the bottleneck using the hardware accelerator you designed in part (f)?

$\boxed{> 20 \times 16\text{b} = 320\text{b}}$

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.