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

ESE5320, Fall 2022 — Final — Friday, December 16

- Exam ends at 2:00PM; begin as instructed (target 12:00PM)
  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	7	8	Total
10	5	5	10	10	15	10	5	10	10	10	100

Average: 73, Std. Dev. 11

Consider the following code to align (rotate, scale, translate) an image to best match a reference image and identify the largest connected region that differs from the reference (likely an object) on a real-time video stream:

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

#define WIDTH 1000
#define HEIGHT 1000
#define COLORS 3
#define MASK 3

#define VPARAMS 5
#define VP_X 0
#define VP_Y 1
#define VP_XS 2
#define VP_YS 3
#define VP_ROT 4

#define XOFF 2
#define YOFF 2
#define ROT 2
#define XSCALE 2
#define XSFACT 2
#define YSCALE 2
#define YSFACT 2

#define REGION_PARAMS 4
#define XMIN 0
#define XMAX 1
#define YMIN 2
#define YMAX 3

#define NODIFF 0
#define DIFFERENT 1
#define THRESHOLD 10

uint16_t reference[HEIGHT][WIDTH][COLORS];
int16_t sintable[360]; // -1 to 1 -- scaled by 2^14
int16_t costable[360];

// treat as single cycle operations
uint16_t max(uint16_t a, uint16_t b) {if (a<b) return(b); else return(a);}
uint16_t min(uint16_t a, uint16_t b) {if (a<b) return(a); else return(b);}

void get_image(uint16_t image[HEIGHT][WIDTH][COLORS]) {
  static uint16_t image_in[HEIGHT][WIDTH][COLORS];
  for (int iy=0;iy<HEIGHT;iy++)
    for (int ix=0;ix<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image[iy][ix][c]=image_in[iy][ix][c];
}

void copy_viewpoint(int16_t orig[VPARAMS], int16_t copy[VPARAMS]) {
  for(int i=0;i<VPARAMS;i++) copy[i]=orig[i];
}
```

```
void compute_viewpoint(uint16_t image[HEIGHT][WIDTH][COLORS],
                       uint16_t reference[HEIGHT][WIDTH][COLORS],
                       int16_t  old[VPARAMS], int16_t current[VPARAMS]) {
  uint64_t best_score=1<<62; // large integer
  for (int rot=old[VP_ROT]-ROT;rot<=old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<=old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<=old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<=old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<=old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;iy<HEIGHT;iy++) // loop F
                for (int ix=0;ix<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=(((ix*cr+iy*sr)*xs)>>(14+8))+x; // 14 to scale sr, cr
                    uint16_t ty=(((ix*sr+iy*cr)*ys)>>(14+8))+y; // +8 for xscale, ysc
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}
void compute_diff(uint16_t raw[HEIGHT][WIDTH][COLORS],
                  uint16_t reference[HEIGHT][WIDTH][COLORS],
                  int16_t current[VPARAMS],
                  uint16_t difference[HEIGHT][WIDTH]) {
  int16_t rot=current[VP_ROT];
  int16_t x=current[VP_X];
  int16_t y=current[VP_Y];
  int16_t xs=current[VP_XS];
  int16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;iy<HEIGHT;iy++) // loop I
    for (int ix=0;ix<WIDTH;ix++) // loop J
      difference[iy][ix]=NODIFF; // assume this runs like streaming data copy
  for (int iy=0;iy<HEIGHT;iy++) // loop K
    for (int ix=0;ix<WIDTH;ix++) // loop L
        {
          uint16_t tx=(((ix*cr+iy*sr)*xs)>>(14+8))+x; // 14 to scale sr, cr
          uint16_t ty=(((ix*sr+iy*cr)*ys)>>(14+8))+y; // +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
            {
              int diff=0;
              for (int c=0;c<COLORS;c++) // loop M
                diff+=abs(raw[ty][tx][c]-reference[ty][tx][c]);
              // should be: diff+=abs(raw[iy][ix][c]-reference[ty][tx][c]);
              if (diff>THRESHOLD)
                difference[iy][ix]=DIFFERENT;
            }

        }
}
```

```
void update(uint16_t label[HEIGHT][WIDTH][REGION_PARAMS],
            uint16_t difference[HEIGHT][WIDTH], int x, int y)
{
  if (difference[y][x]==DIFFERENT)
    for (int xoff=-1;xoff<2;xoff++) // loop S
      for (int yoff=-1;yoff<1;yoff++) // loop T
        if (difference[y+yoff][x+xoff]==DIFFERENT)
          {
            label[y][x][XMIN]=min(label[y][x][XMIN],label[y+yoff][x+xoff][XMIN]);
            label[y][x][YMIN]=min(label[y][x][YMIN],label[y+yoff][x+xoff][YMIN]);
            label[y][x][XMAX]=max(label[y][x][XMAX],label[y+yoff][x+xoff][XMAX]);
            label[y][x][YMAX]=max(label[y][x][YMAX],label[y+yoff][x+xoff][YMAX]);
          }
}
void largest_region(uint16_t difference[HEIGHT][WIDTH],
                    uint16_t  region[REGION_PARAMS]) {
  uint16_t label[HEIGHT][WIDTH][REGION_PARAMS];
  int best_area=0;
  for (int iy=0;iy<HEIGHT;iy++) // loop N
    for (int ix=0;ix<WIDTH;ix++) // loop O
      if (difference[iy][ix]==DIFFERENT)
          {
            label[iy][ix][XMIN]=ix;
            label[iy][ix][XMAX]=ix;
            label[iy][ix][YMIN]=iy;
            label[iy][ix][YMAX]=iy;
          }
  for (int iy=0;iy<HEIGHT;iy++) // loop P
    {
      for (int ix=0;ix<WIDTH;ix++) // loop Q
        update(label,difference,iy,ix);
      for (int ix=WIDTH;ix>-1;ix--) // loop R
        update(label,difference,iy,ix);

    }
  for (int iy=0;iy<HEIGHT;iy++) // loop U
    {
      for (int ix=0;ix<WIDTH;ix++) // loop V
        {
          int area=(label[iy][ix][XMAX]-label[iy][ix][XMIN])*
            (label[iy][ix][YMAX]-label[iy][ix][YMIN]);
          if (area>best_area)
            {
              best_area=area;
              region[XMIN]=label[iy][ix][XMIN];
              region[XMAX]=label[iy][ix][XMAX];
              region[YMIN]=label[iy][ix][YMIN];
              region[YMAX]=label[iy][ix][YMAX];
            }
        }
    }
  return;
}
```

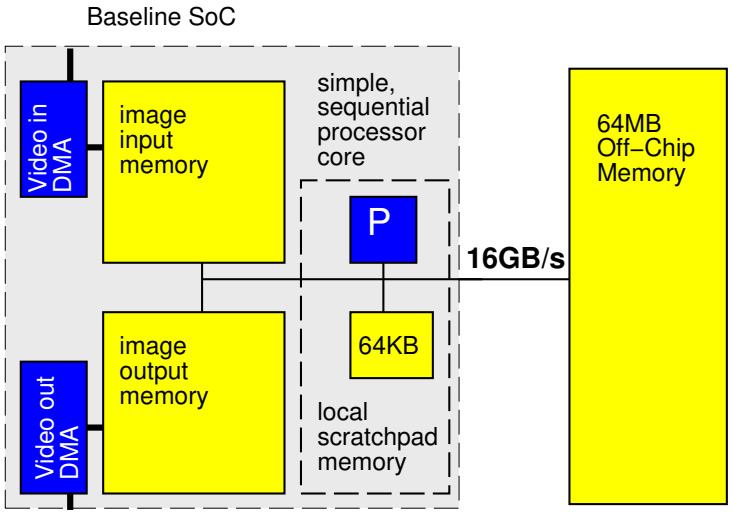
```
void send_region(uint16_t region[REGION_PARAMS], int16_t current[VPARAMS],
                 uint16_t image[HEIGHT][WIDTH][COLORS])
{
  static uint16_t image_out[HEIGHT][WIDTH][COLORS];
  int16_t rot=current[VP_ROT];
  int16_t x=current[VP_X];
  int16_t y=current[VP_Y];
  int16_t xs=current[VP_XS];
  int16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];

  int basey=region[YMIN];
  int basex=region[XMIN];
  for (int iy=basey;iy<=region[YMAX];iy++) // W
    for (int ix=basex;ix<=region[XMAX];ix++)
      {
        uint16_t tx=(((ix*cr+iy*sr)*xs)>>(14+8))+x; // 14 to scale sr, cr
        uint16_t ty=(((ix*sr+iy*cr)*ys)>>(14+8))+y; // +8 for xscale, yscale
        if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
            for (int c=0;c<COLORS;c++)
              image_out[iy-basey][ix-basex][c]=image[ty][tx][c];
      }
}

void extract_region() {
  uint16_t raw[HEIGHT][WIDTH][COLORS]; // uint16_t for 16b (2 byte) color per pixel
  uint16_t difference[HEIGHT][WIDTH];
  int16_t viewpoint[VPARAMS];
  int16_t old_viewpoint[VPARAMS];
  uint16_t region[REGION_PARAMS];
  get_image(raw);
  copy_viewpoint(viewpoint,old_viewpoint);
  compute_viewpoint(raw,reference,old_viewpoint,viewpoint);
  compute_diff(raw,reference,viewpoint,difference);
  largest_region(difference,region);
  send_region(region,viewpoint,raw);
}

int main() {
  while (1) { // loop Z
    extract_region();
  }
}
```

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, abs, shifts, max, min, and multplies as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with these compute operations and memory operations. (Some consequences: You may ignore loop and conditional overheads in processor runtime estimates; you may ignore computations in array indecies.)
- Baseline (simple, sequential) processor can execute one add, compare, shift, abs, max, min, or multiply per cycle and runs at 1 GHz.
- Data can be transfered between pairs of memory (including main memory) at 16 GB/s when streamed in chunks of at least 1000B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to image and 64 MB off-chip memories takes 10 cycles and can move 8B.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 16 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- Baseline processor is 1 mm^2 of silicon including its 64KB local scratchpad.
- By default, all arrays live in the 64 MB off-chip memory.
- `image_in` and `image_out` live in the respective image input and image output memories.
- Arrays for `sintable`, `costable`, viewpoints (`old_viewpoint`, `viewpoint`), and `region` live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, max, min, adds, 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. Consider abs and shift free in hardware.
- Data can be transfered to accelerator local memory at the same 16 GB/s when streamed in chunks of at least 1000B.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for compute operations and memory access for operations inside `extract_region`.

routine	
compute_viewpoint	
(compute)	$5^3 \times 3^2 \times 1000^2(6 \times 2 + 4 + 3 \times 3)$
(memory)	$5^3 \times 3^2 \times 1000^2 \times 3 \times 2 \times 10$
compute_diff	
(compute)	$1000^2 \times (6 \times 2 + 4 + 3 \times 3 + 1)$
(memory)	$1000^2 \times (3 \times 2 + 1) \times 10$
largest_region	
(compute)	$1000^2 \times 5 \times 10 + 1000^2 \times 2 \times 31 + 1000^2 \times 4$
(memory)	$1000^2 \times 5 + 1000^2 \times 2 \times 78 \times 10 + 1000^2 \times 12 \times 10$
send_region	
(compute)	$1000^2 \times (6 \times 2 + 4)$
(memory)	$1000^2 \times 3 \times 2 \times 10$

routine	Compute	Memory
compute_viewpoint	28×10^9	67.5×10^9
compute_diff	26×10^6	70×10^6
largest_region	71×10^6	1.73×10^9
send_region	16×10^6	60×10^6
extract_region	28×10^9	69×10^9

Not include the logical ands (&&) in calculation above. Ok if included in solutions.
Update is $6 \times 5 + 1 = 31$ compute operations and $6 \times (3 \times 4 + 1) = 78$ memory operations to large memories.

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

   (a) Considering both compute and memory cycles, what routine is the bottleneck? (circle one)

   (compute_viewpoint)
   compute_diff
   largest_region
   send_region

   (b) What is the Amdahl's Law speedup if you only accelerate the identified function?
   $\frac{97.623}{1.973} \approx 49$

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	Data (Reduce) Sequential Parallel	min reduce best_score
F	Data (Reduce) Sequential Parallel	sum reduce score
K	(Data Reduce Sequential Parallel)	difference computed pixel by pixel
N	(Data Reduce Sequential Parallel)	independent initial assignments
P	Data Reduce (Sequential) Parallel	row labels depend on labeling for previous row
Q	Data Reduce (Sequential) Parallel	depends on label of previous x position
U	Data (Reduce) Sequential Parallel	max reduce over best_area

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

NO	read difference (all parallel)	10
	compare (all parallel)	1
	assign (all parallel)	10
P	read differences (all parallel)	10
	initial label reads during difference	
	then communicate through registers	
	rows sequentialized ($\times 1000$)	
	left and right scan concurrent	
Q,R	scan across rows sequentialized ($\times 1000$)	
	XMAX, XMIN, YMAX, YMIN independent (parallel)	
	tight loop is max/min with previous entry in row (1)	
	tight loop $\times$ row scan $\times$ rows	10^6
	final label write	10
UV	read label (all parallel)	10
	subtracts	1
	multiply	1
	reduce best_area	$\log_2(10^6)$
	write region	1
Total		$10^6 + 74$

(This page intentionally left mostly blank for answers.)

How tight the core in QR/update can be is tricky. At the most conservative, read label and difference (10), reduce of 6 elements in 3 cycles, then update label (10), for 23. The reads and writes are avoidable if we keep in registers since the dataflow between operations is known. The values on previous row can be reduced ahead of the critical cycle from time (as can self and item after it on row that isn't changing).

Running left and right scans concurrently will require some care right at the middle where they are updating values in close proximity. That may make the middle labels take a few more cycles.

We don't think can treat x-scan as a reduce; only took off one point if did that.

5. How would you modify `compute_viewpoint` to minimize the memory resource bound by exploiting the scratchpad memory and streaming memory operations.

   - Annotate what arrays live in the local scratchpad.
     Create new array `image_row` to hold each row of the image.
   - Account for total memory usage in the local scratchpad (use provided table).
   - Describe your modifications to the code.
     - Use **for** loops that only copy data to denote the streaming operations

     Copy each row ($1000 \times 3 \times 2$B) into `image_row` in the body of F before starting G. All references to `image[iy][ix]` now go to `image_row[ix]`.
   - Estimate the new memory resource bound for your optimized `compute_viewpoint`.
     reference reads unchanged: $5^3 \times 3^2 \times 1000^2 \times 3 \times 1 \times 10 = 33.75 \times 10^9$
     image reads now local: $5^3 \times 3^2 \times 1000^2 \times 3 \times 1 \times 1 = 3.375 \times 10^9$
     Add time to stream in row: $5^3 \times 3^2 \times 1000 \times \frac{6000}{16} = 0.421 \times 10^9$

     $(33.75 + 3.375 + 0.421) \times 10^9 = 37.546 \approx 38 \times 10^9$

Variable	Size (Bytes)
`image_row[WIDTH]COLORS]`	6000B

(This page intentionally left mostly blank for answers.)

6. Considering a custom hardware accelerator implementation for loops A–H of `compute_viewpoint` 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?

   Start by assuming we unroll H; we need to understand how much unrolling of the rest of the loops is required. Since the loops are associative reduce, the inner loop can be pipelined to II=1. $\frac{5^3 \times 3^2 \times 1000^2}{A \times 10^9} \leq \frac{1}{30}$, giving us about 34. This suggests unrolling about a factor of 34 beyond H will be sufficient. It might be good to round up to something that divides 1000, but took either.

   Common Problem: Not accounting for pipelining.

Loop	Unroll Factor
A	1
B	1
C	1
D	1
E	1
F	1
G	34
H	3

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

Absolute Value	$3 \times 34 = 102$
Adders	$34(3 \times 2 + 2 \times 2) = 340$
Multipliers	$34 \times 3 \times 2 = 204$

(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)
old[]	1	none	1	16	10
current[]	1	none	1	16	10
sintable[]	1	none	1	16	360
costable[]	1	none	1	16	360
image[]	1	n/a			
image_line[]	1	cyclic 34 dim 1, x complete dim 2, c (and pack c)	1	48	30
reference[]	1	none	34	48	1,000,000

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
`compute_viewpoint` → `compute_diff`	N	10	`viewpoint`
`compute_diff` → `largest_region`	Y	2000	`difference[Y]` row
`largest_region` → `send_region`	N	8	`region`

Explain size choices for partial credit consideration.

Need to process entire search in `compute_viewpoint` before have a new viewpoint ($5 \times 2\text{B} = 10\text{B}$) to pass to `compute_diff`. `compute_diff` needs the viewpoint to process any image pixels.

`compute_diff` and `largest_region`process rows in order. So, `difference` rows can be passed as completed in `compute_diff`. Since `largest_region` processes rows both increasing X and decreasing X, it will need an entire row at a time for computation—cannot get away with processing individual pixels as they arrive since pixels will be coming increasing X (or, can process, but still need to buffer for use in decreasing X pass).

Need to process entire labeling in `largest_region` before have a new region ($4 \times 2\text{B} = 8\text{B}$) to pass to `send_region`.

8. Assuming you start with the accelerator from Problem 6, and building on your previous answers, what else do you need on an SoC to achieve real-time (30 frame/second) operation for `main`?

   - What processor(s) do you need to run the remaining code? (how many? any particular properties)?
   - If necessary, how is the remaining code divided among the processors?
   - What changes (if any) are needed to memory organization and data movement?

   Will need to perform stream prefetch conversion to difference (**`compute_diff`**) and difference and label (**`largest_region`**) to reduce memory time.

   Setup a coarse-grained dataflow pipeline with the accelerator and the downstream function/operations. We'll use one or more processors for each remaining function.

   `compute_diff` memory time saves about 9M cycles and can now be performed by 3 processors operating data parallel.

   `largest_region` memory time roughly reduces by a factor of 10 since all memories rows are referenced in order and can be streamed. Run xmin/ymin/xmax/ymax on separate processors. That would bring computation down below 20M cycles each, and memory down below 45M. It will take some additional care to avoid redundant reads and writes in update to get memory cycles below 13M to fit in the 33M cycle budget. Could also run left and right scan on separate processors. So, using 8 processors could work with less memory optimization.

   `largest_region` is tighter than it should be. Likely doable, but requires more tricks to make work in time budget.

   `send_region` memory time roughly reduces by a factor of 10, so **`send_region`** can be performed on a single processor.

   So, we are using a total of 3+4+1=8 processors (or 12 if use 8 processors for **`largest_region`**) beyond the **`compute_viewpoint`** accelerator.

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.