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

ESE532, Fall 2017 | Final | Thursday, December 21

- Exam ends at 5:00PM; 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.
- 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

Problem 1			Problem 2			Problem 3		Problem 4					Total
a	b	c	a	b	c	a	b	a	b	c	d	e	Total
10	10	5	10	5	10	15	10	5	5	5	10	5	100

Average 65, Std. Dev. 16

1. Consider the following set of real-time tasks for a vision-based control system:

Task	Frequency	Operations per Invocation
Plan	once per second	1.28×10^{11}
Control	100 times per second	10^6
Mapping	50 times per second	4×10^7
Collide	100 times per second	10^7

Consider the following computational elements:

Compute Element	Performance	Area
High Performance Processor (HPP)	1 op per cycle at 2 GHz	3×10^4
VLIW	8 ops per cycles at 500 MHz	3×10^3
6-LUT (with interconnect)	200 MHz	1

Consider the following spatial datapaths for each of the tasks:

Task	6-LUTs	Effective operations per cycle
Plan	1,300	40
Control	32,000	500
Mapping	33,000	1024
Collide	3,200	100

Assume:

- perfectly divide work among processors (if need more than one),
- perfectly schedule VLIW
- perfectly divide task among instances of spatial datapaths shown above
- may share processor or VLIW among multiple tasks

(a) Complete table with the number of homogeneous resources required to meet real-time task goals and the associated area: (May share processors, VLIW, but must pay for an integer number of each.)

Task	HPP	VLIW	6-LUTs
Plan	$\frac{1.28\times10^{11}}{2\times10^{9}}=64$	$\frac{1.28\times10^{11}}{8\times5\times10^{8}}=32$	$\left(\frac{1.28\times10^{11}}{40\times2\times10^{8}}\right)1300=20{,}800$
Control	$\frac{100\times10^{6}}{2\times10^{9}}=\frac{1}{20}$	$\frac{100\times10^{6}}{8\times5\times10^{8}}=\frac{1}{40}$	$\left\lceil\frac{100\times10^{6}}{500\times2\times10^{8}}\right\rceil 32000=32{,}000$
Mapping	$\frac{50\times4\times10^{7}}{2\times10^{9}}=1$	$\frac{50\times4\times10^{7}}{8\times5\times10^{8}}=\frac{1}{2}$	$\left\lceil\frac{50\times4\times10^{7}}{1024\times2\times10^{8}}\right\rceil 33000=33{,}000$
Collide	$\frac{100\times10^{7}}{2\times10^{9}}=\frac{1}{2}$	$\frac{100\times10^{7}}{8\times5\times10^{8}}=\frac{1}{4}$	$\left\lceil\frac{100\times10^{7}}{100\times2\times10^{8}}\right\rceil 3200=3{,}200$
Total (integer)	66	33	89,000
Total Area	2.0×10^{6}	9.9×10^{4}	8.9×10^{4}

(b) Heterogeneous solution that meets real-time goals and minimizes area. For each task identify the number and type of compute units.
(May share processors, VLIW, but must pay for an integer number of each. Show number of datapaths for 6-LUT case, and calculate total area of those datapaths for Area column, when appropriate.)

Task	Type	Number	Area
Plan	6-LUT	16×1300	20,800
Control	VLIW		
Mapping	VLIW	1	3,000
Collide	VLIW		
		Total Area	23,800

(c) Show schedule for any HPP or VLIW processors that run multiple tasks at the coarsest time-slice for the task set.

Time Slot				Time Slot			
10 ms				10 ms			
5 ms	2.5 ms	0.25 ms	2.25 ms	5 ms	2.5 ms	0.25 ms	2.25 ms
map	collide	control	(unused)	map	collide	control	(unused)

Since these are real-time tasks, they must run at the specified frequency. We must run control and collide once every 10 ms (100 times per second). This means we must schedule all the tasks on the same processor on a 10 ms time slice to guarantee each control and collide task gets run. Since the map task only needs to run once ever 20 ms, we must split its $\frac{4\times10^7}{8\times5\times10^8\text{Hz}} = 10\,\text{ms}$ runtime into two 5 ms slices so that it gets its 10 ms of runtime every 20 ms while allowing control and collide to run once ever 10 ms.

2. Parallelism in Plan Task

   Working with the following algorithm for Plan and assuming this takes the 1.28×10^{11} operations stated in Problem 1:

```
#define DIM 1024
typedef struct cell {
   uint32_t cost;
   uint32_t path;
   uint16_t edge_cost[4];
   } cell_struct;
cell_struct from[DIM][DIM];
cell_struct to[DIM][DIM];
int xoff, yoff;

for (int iter=0;iter<3*DIM;iter++) {
  for (int y=0;y<DIM;y++)
    for (int x=0;x<DIM;x++) {
       uint32_t min_cost=from[y][x].cost;
       uint32_t min_path=from[y][x].path;
       for (int j=0;j<4;j++) {
         if (j==0)      {xoff=0; yoff=-1;}
         else if (j==1) {xoff=-1; yoff=0;}
         else if (j==2) {xoff=1; yoff=0;}
         else if (j==3) {xoff=0; yoff=1;}
         uint32_t ncost=from[y+yoff][x+xoff].cost;
         uint32_t ecost=from[y][x].edge_cost[j];
         uint32_t icost=ncost+ecost
         if (icost < min_cost){
            min_path=ENCODE(yoff,xoff);  // count 3 ops (depth 1 op)
            min_cost=icost;
            } // icost < min_cost
         } // for j
       to[y][x].cost=min_cost;
       to[y][x].path=min_path;
      } // for x
  cell_struct **tmp=from;
  from=to;
  to=tmp;
  } // for iter
```

(a) Describe how you can divide the computational task among processors or hardware datapaths to achieve various levels of parallelism Write code snippets or edits as necessary to make your division clear.

There are no data dependencies within the x and y loops, so computations are data parallel for each output [y][x] location. We can divide the computation by x-y regions onto the processors.

The simplest solution, which is adequate for the parallelism needed for Problem 1 (16–64), is to divide along y values, assigning each processor or datapath k to a range of y values from $\left(\frac{DIM}{P}\right)k$ to $\left(\frac{DIM}{P}\right)(k+1)-1$.

Dividing over both x and y would reduce the number of `from[y][x]` cells that are needed by multiple processors.

(b) Building on your answer above, how do you provide the specific parallelism needed in Problem 1? (Number is the value you calculated in Problem 1)

Compute Element	Number	How
HPP	64	Assign y=16k to $16(k+1)-1$ to each processor k
VLIW	32	Assign y=32k to $32(k+1)-1$ to each processor k
6-LUT datapaths	16	Assign y=64k to $64(k+1)-1$ to each datapath k

(c) What is the Latency Bound (critical path) in operations for the entire task assuming the data needed lives in registers (i.e., no latency coming or going from memory)?

No x, y dependence, so all the j loops can be run concurrently. There is a dependence between outer (iter) iterations, so we must run the $3\times$DIM j-loops in series. Each j-iteration performs an addition (icost), comparison, and a selection (2 serial operations). The ENCODE can happen in parallel with the comparison (or, strictly speaking, can be performed once at the beginning of the computation outside the iter loop). The 4 icost additions have no dependence and can be performed in parallel. Serialized by the comparison/selection, the four iterations of the j-loop have a latency of $4 \times 2 + 1$ (also see solution to Problem 3a). So, the entire latency is: $3 \times 1024 \times (4 \times 2 + 1) = 27,648$.

The only reason the j-loop is serialized is that we are trying to guarantee the path selected is exactly the same as the sequential case when there are equal cost options from multiple directions. If we don't care which path is seleted, just that it is a minimum cost path, then we can perform the j-loop selection as a reduction rather than as a sequential operation. Now we can compute the ncost additions all at once, then 3 stages of reduce ($\lceil \log_2(5) \rceil$, since we must select among 5 things – the original and the 4 neighbors), each of which requires a comparison and a mux selection. So, the latency is: $3 \times 1024 \times (1 + 3 \times 2) = 21,504$.

Either solution was acceptable. In practice, if there's no reason to prefer a particular path, the lower latency (more parallel) one is preferred. If you are working with customers, you might ask their constraints. For an exam or assignment, it is always best to state the choice you made. Similarly, for customers, it is good to document decisions like this.

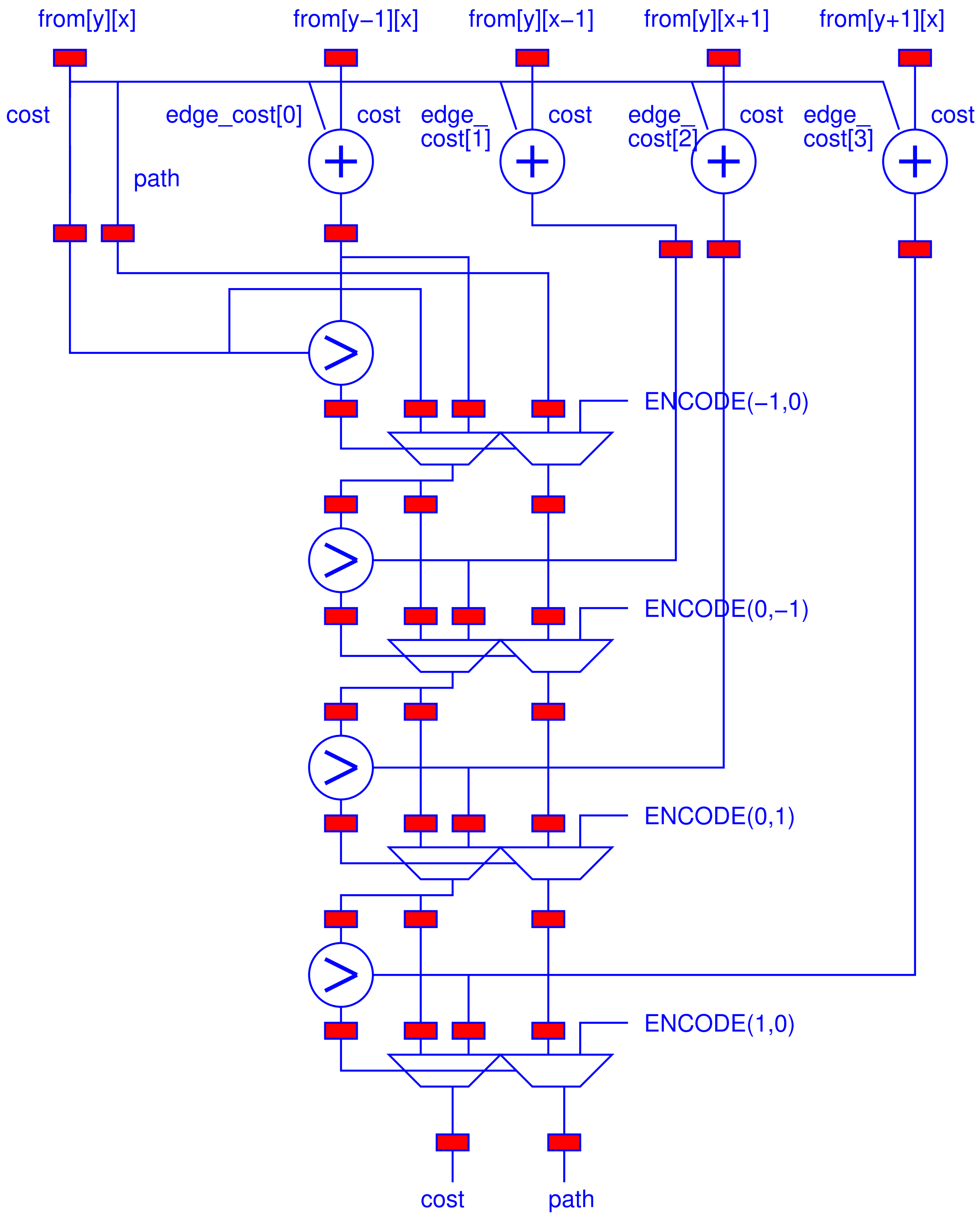
3. Hardware datapath for Plan

   Working with the same algorithm for Plan:

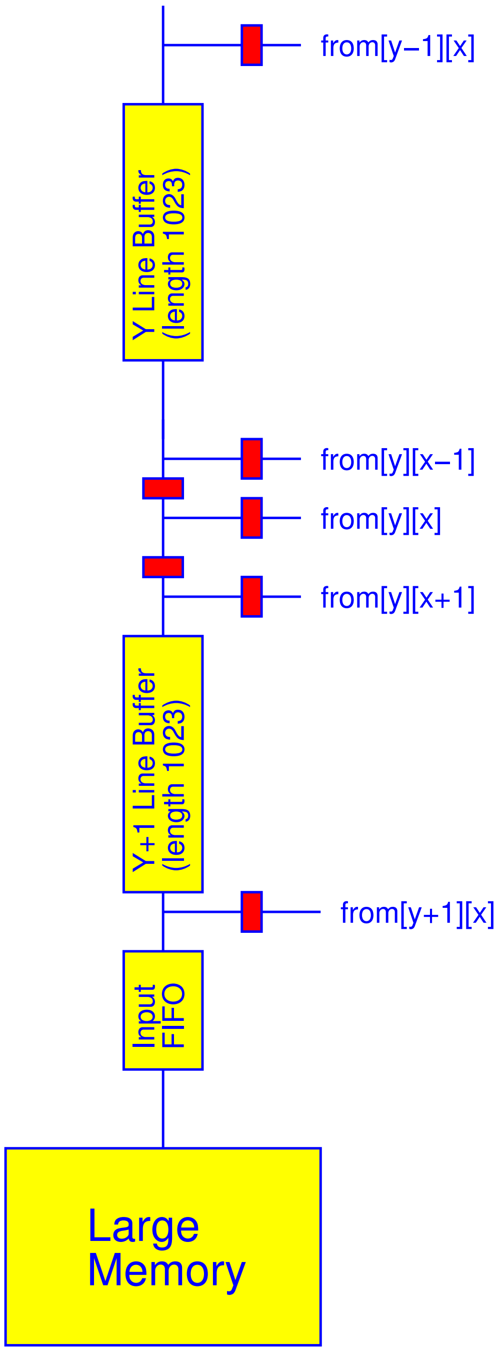
```
#define DIM 1024
typedef struct cell {
   uint32_t cost;
   uint32_t path;
   uint16_t edge_cost[4];
   } cell_struct;
cell_struct from[DIM][DIM];
cell_struct to[DIM][DIM];
int xoff, yoff;

for (int iter=0;iter<3*DIM;iter++) {
  for (int y=0;y<DIM;y++)
    for (int x=0;x<DIM;x++) {
       uint32_t min_cost=from[y][x].cost;
       uint32_t min_path=from[y][x].path;
       for (int j=0;j<4;j++) {
         if (j==0)      {xoff=0; yoff=-1;}
         else if (j==1) {xoff=-1; yoff=0;}
         else if (j==2) {xoff=1; yoff=0;}
         else if (j==3) {xoff=0; yoff=1;}
         uint32_t ncost=from[y+yoff][x+xoff].cost;
         uint32_t ecost=from[y][x].edge_cost[j];
         uint32_t icost=ncost+ecost
         if (icost < min_cost){
            min_path=ENCODE(yoff,xoff);  // count 3 ops (depth 1 op)
            min_cost=icost;
            } // icost < min_cost
         } // for j
       to[y][x].cost=min_cost;
       to[y][x].path=min_path;
      } // for x
  cell_struct **tmp=from;
  from=to;
  to=tmp;
  } // for iter
```

(a) Unroll and pipeline the inner (j) loop.
Draw the resulting pipelined datapath.



(b) Show how you manage data storage and movement of `from` data in order to deliver `from` data to the datapath above. Assume data is streamed from the large memory. Draw registers and memories and show inputs to the Part a datapath.



(This page intentionally left mostly blank for pagination.
You may use for extra answer or work space.)

4. Data Managment

   Continuing to work with the algorithm for Plan:

```
#define DIM 1024
typedef struct cell {
   uint32_t cost;
   uint32_t path;
   uint16_t edge_cost[4];
   } cell_struct;
cell_struct from[DIM][DIM];
cell_struct to[DIM][DIM];
int xoff, yoff;

for (int iter=0;iter<3*DIM;iter++) {
  for (int y=0;y<DIM;y++)
    for (int x=0;x<DIM;x++) {
       uint32_t min_cost=from[y][x].cost;
       uint32_t min_path=from[y][x].path;
       for (int j=0;j<4;j++) {
         if (j==0)      {xoff=0; yoff=-1;}
         else if (j==1) {xoff=-1; yoff=0;}
         else if (j==2) {xoff=1; yoff=0;}
         else if (j==3) {xoff=0; yoff=1;}
         uint32_t ncost=from[y+yoff][x+xoff].cost;
         uint32_t ecost=from[y][x].edge_cost[j];
         uint32_t icost=ncost+ecost
         if (icost < min_cost){
            min_path=ENCODE(yoff,xoff);  // count 3 ops (depth 1 op)
            min_cost=icost;
            } // icost < min_cost
         } // for j
       to[y][x].cost=min_cost;
       to[y][x].path=min_path;
      } // for x
  cell_struct **tmp=from;
  from=to;
  to=tmp;
  } // for iter
```

- Processor has a local memory that holds 64KB with single cycle access for 16b, 32b, or 64b data.
- uint32_t is stored in 4 Bytes; uint16_t is stored in 2 Bytes.
- `from` and `to` live in the large memory.
- Large memory has a read latency of 50 ns (100 cycles on HPP) for a single 16b, 32b, or 64b word or 100 ns (200 cycles) for a 2048b transfer into the local memory
  - For the processor, assume you have a routine `void read2048(int *local_buf,int *large_mem_buf)` that will initiate a 2048b read from the large memory into the processor local memory at the specified addresses.
- The processor can issue a write to the large memory in a single cycle and continue without waiting.
  - We should be concerned with when a read may need something previously written to memory and the impact of writes on memory bandwidth. To keep this problem simple, you may ignore these effects.
- Assume scalar (non-array) variables can live in registers.
- You may ignore loop and conditional overheads in processor runtime estimates; additions and comparisons are single cycle operations.

(a) What time (in ns) is required for the task as written when running on the HPP and all references to each component of `from` go to the large memory?

Read from **`from`** 10 times inside x loop (cost, path, and 4 reads to each of cost and edge_cost inside the j loop): $3 \times 1024^3 \times 10 \times 50\,\text{ns}$.
Perform 3 adds, 1 compare, and 1 ENCODE (3 ops) inside j-loop contributing 7×4 operations inside x loop; perform 2 writes to **`to`** for a total of 30 cycles inside x-loop: $3 \times 1024^3 \times 30 \times 0.5\,\text{ns}$.
Total: $3 \times 1024^3 \times 515\,\text{ns}=1.7 \times 10^{12}\,\text{ns}$.

(b) What fraction of time is spent performing read operations from the large memory for `from`?

$\frac{500}{515} = 97\%$

(c) What is the Amdahl's Law speedup if you only accelerate these memory reads to `from`?

$\frac{515}{15}=34\times$

(d) How do you need to modify the design to minimize the time spent reading `from` from the large memory?

Describe how the code needs to be modified. Show revised code as necessary for clarity.

Keep the the y-1, y, and y+1 lines in local memory. With 16B per cell_struct, this is 1024 $\times$ 16B per line or 48KB for all 3 lines, which fits in the 64KB memory. On entering the y loop, we will need to read in a new line. We can do this efficiently using read2048 to read 16 cell_struct entries at a time (16 entries $\times$ 16B/entry=256B $\times$ 8b/B = 2048b). Inside the x loop, all references are to the local memory. To get things setup, we need a prologue to read the first two lines before the y loop.

```
// all 3 in local memory
cell_struct prev[DIM];
cell_struct current[DIM];
cell_struct next[DIM];

for (int iter=0;iter<3*DIM;iter++) {
  for (int x=0;x<DIM;x++)
     current[x]=OFF_CELL; // cell with maxval for .cost so never chosen
  for (int x=0;x<DIM;x+=(256/sizeof(cell_struct)))
     read2048(&next[x],&from[0][x]);
  for (int y=0;y<DIM;y++) {
    cell_struct *tmp=prev;
    prev=current;
    current=next;
    next=tmp;
    for (int x=0;x<DIM;x+=(256/sizeof(cell_struct)))
      read2048(&next[x],&from[y+1][x]);
    for (int x=0;x<DIM;x++) {
       uint32_t min_cost=current[x].cost;
       uint32_t min_path=current[x].path;
       for (int j=0;j<4;j++) {
         cell_struct *nline;
         if (j==0)      {xoff=0; yoff=-1; nline=prev;}
         else if (j==1) {xoff=-1; yoff=0; nline=current;}
         else if (j==2) {xoff=1;  yoff=0; nline=current;}
         else if (j==3) {xoff=0;  yoff=1; nline=next}
         uint32_t ncost=nline[x+xoff].cost;
         uint32_t ecost=current[x].edge_cost[j];
         uint32_t icost=ncost+ecost
         if (icost < min_cost){
            min_path=ENCODE(yoff,xoff);  // count 3 ops (depth 1 op)
            min_cost=icost;
            } // icost < min_cost
         } // for j
       to[y][x].cost=min_cost;
       to[y][x].path=min_path;
      } // for x
    } // for y
  cell_struct **tmp=from;
  from=to;
  to=tmp;
  } // for iter
```

(e) What is the time (in ns) required on a single HPP processor after this modification?

Each read2048 is able to bring in 256B. Since each cell_struct is 16B, this means we get 16 cell_structs per 100 ns read.

Reading `from`: $3 \times 1024^2 \times \frac{1024 \times 16}{256} \times 100\,\text{ns}$.

Local Reads: $3 \times 1024^3 \times 10 \times 0.5\,\text{ns}$.

Operations and writes: $3 \times 1024^3 \times 30 \times 0.5\,\text{ns}$.

Total: $3 \times 1024^3 \times 26.25\,\text{ns} = 8.5 \times 10^{10}\,\text{ns}$.

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.