EEE4120F Class Test 2021

Date: 31 May 2021

Time: 18:00-21:00 *Duration:* 2h 30min

Venue: Beattie LT,B115

Procedures:
please read this page while waiting for the test to start!

This test is planned to be venue-based. Accordingly, the requisite procedures must be followed at all times, in line with the UCT tests and exams COVID-19 protocols for venue-based examinations. All students and invigilators are required to wear a mask while on campus and in the testing venue.

NOTE: NO MASK = NO ACCESS TO THE TEST VENUE AND YOU WILL BE MARKED AS ABSENT.

We are going to be strict with the time duration due to health and safety reasons. Any latecomers will not be provided additional time once the time is up.

Documents involved: You should have the following papers on your desk: This question paper, the answer sheet, and an exam booklets. Do not turn over the question paper or answer sheets before you are told that the test has started. The UCT examination answer book is provided just for your rough work, make sure to put your answers in the provided answer sheet.

The confusion log: A "confusion log" page (the last page in the answer sheet) is where you can make note of difficulties or confusion experienced in the test in terms of the clarity of question(s) posed, as there will be limited opportunity for asking questing during the examination. However, do not waste time adding entries to the "confusion log" in the hope of the marking being gentler, such an approach will likely backfire. If there is a question that confuses you, keep your log entry brief but focused on the issue, and if relevant include a short indication of assumption(s) made.

If finished early: If completed early, and before the last 20 minutes of the test, please signal to the invigilator that you are done and will be leaving. Then proceed to leave, taking your test answer paper and depositing it in the answer return box at the venue exit.

When finished: Check that all your answer papers have your student number on them and also put your name and test name on the cover of your exam answer book (if you used it). Then wait for the exit procedure to be started, in which rows will depart single file maintaining social distance. Leave, taking your test answer paper and depositing it in the answer return box at the venue exit.

Instructions and Test Structure

- Make sure that your student number is on your answer sheet and confusion log.
- There are **3** parts to this test, each divided into sub-questions. Answer all questions.
  Marks breakdown: Part 1: 50 marks Part 2: 30 marks Part 3: 20 marks
- Verilog cheat sheet on last page!
- Total time: 2.5 hours Total marks: 100

Part 1: Multiple Choice [50 marks]

Use Part 1 of the Answer Sheet for your answers. Make sure to circle just one answer option, A – E, unless specifically told that more than one option can be selected. If using pencil, please go over your pencil answers in pen before the test ends for a more permanent selection, any answer in pen will be prioritized over an answer in pencil.

Questions 1.1 – 1.3 doubles as a 2nd attempt for the Comprehension Test. Answer all these questions regardless of your ELO2 status (if you have already passed the Comprehension Test your ELO2 status will not be affected).

Q 1.1: **[5]**

In considering Raven Rover subproblem 1.1., it was mentioned to convert the OCTAVE flipimage.m code to C, to do performance testing. Considering that the application needs to be high performance, to reduce delays in gathering radar images data, select a good reason below explaining why the OCTAVE was to be changed to C instead of just loading OCTAVE and running the m-file directly on Raven's Mini Data Centre. (select only one option)

(a) OCTAVE has too limited a range of functionality, therefore it is far better to use standard C and a small set of efficient libraries, such as system.h and stdio.h.
(b) OCTAVE code will execute far more efficiently than standard C code, therefore measuring the performance in OCTAVE would greatly overestimate the speed.
(c) This was asked for because running interpreted OCTAVE code has a performance cost, as well as the added burden of loading and keeping OCTAVE in memory.
(d) The development work of coding this routine in C would be much quicker and easier for others to understand, hence the desire to have it in C, especially for a large team.
(e) Converting the code to C would be extraneous because doing the performance testing of this routine in OCTAVE or in C will give very similar results.

Q 1.2: **[5]**

When considering communications between the control centre on Earth and Curiosity on Mars (using its directional antenna), which statement below is accurate about the time it takes for a radio message to go from one to the other… (select only one option)

(a) Curiosity sends its data fast, getting up to 10Mbps for sending data to Earth, but the return is much slower.
(b) The lag in communications from Curiosity, on Mars, to the Earth-based control centre can vary greatly, being affected by the orbital characteristics of both Earth and Mars which causes the distances between the two to changing by millions of kilometres.
(c) Although the signals sent from either side are dispatched at the same speed, they can get delayed due to the sun, an asteroid or some other obstruction in the way causing these microwave signal to go more slowly through this stuff and thus taking longer.
(d) The problem is entirely on the Earth side because it is not always the case that one of the Earth-side receiver stations is pointing towards Mars when it transmits.
(e) All the above are entirely false: the radio messages always take the same time to go from the one to the other.

Q 1.3: **[5]**

In subproblem 1.2(c) you are asked "What is the best-case power consumption for transmitting an image from Rover to Curiosity?" i.e. via WiFi, and for an uncompressed image. What was the most important aspect you needed to calculate prior to determining the power consumed for this? Select one options below indicating what was needed to get the right answer… (select only one option)

(a) You need to figure out how many blocks were being transmitted as this would influence the amount of data Raven needs to send via SPI to the WiFi module which would consume more power than the actual 30mW WiFi signal transmission.
(b) You needed to figure out how long the transmitter was on for (say T seconds) and multiply this by the Watts drawn per second when transmitting (i.e. 30mW), i.e. getting an answer of (T x 30)/1000 W.
(c) You would need to determine how many images are currently in the Raven's memory, as the more images the more power would be consume for sending any one of them.
(d) By finding how long the transmitter will be on for (say T seconds), and how much power the transmitter draws (say 30mW) you can calculate the power draw per hour.
(e) It all depends on the specification of the transmitter. If it draws 30mW, as was specified, then there you go: that is the power used. 30mW, no more and no less.

***

Questions 1.4 – 1.6 doubles as a 2nd attempt for the Validation Test. Answer all these questions regardless of your ELO2 status (if you have already passed the Validation Test your ELO2 status will not be affected).

Q 1.4: **[5]**

Which one of the cases below is most accurately a description of a SISD as per Flynn's taxonomy of computer architecture … (select only one option)

(a) The computer can access one data item and apply it to only one instruction at a time.
(b) The computer can access one data item and only apply it to the next instruction.
(c) The computer can access multiple data items but can only apply the same instruction to each data item.
(d) The processor can access one data item and can apply that item do multiple instructions at once.
(e) The computer can access multiple data items and can apply possible different operations to each item accessed.

Q 1.5: **[5]**

In Lecture 9, the eight main steps in designing parallel systems were discussed. Consider that you were assigned to develop a parallel solution to run on the Rover Raven's Mini Data Centre (MDC). What is the first step (as per the list) you would start with? (select only one option)

(a) Load balancing of the solution.
(b) Performance analysis of the solution.
(c) Understanding the problem that needs the solution.
(d) Rearrange the granularity of the solution.
(e) Prepare a golden measure (in OCTAVE or MATLAB) for the solution.

Q 1.6: **[5]**

Consider that you're the first engineer to develop an application for the Rover Raven's Mini Data Centre, and have chosen to develop an application framework. What is a major motivation for choosing a framework approach? (select only one option)

(a) It could save you time in getting a first 'hello world' type application running.
(b) It could save time for you and future developers using the architecture.
(c) It could trial the versatility of the architecture.
(d) It could be used for optimizing the hardware design.
(e) It could be used to compare the performance of OCTAVE to C.

***

Questions 1.7 – 1.10 are quick checks of your understanding of some syllabus topics.

Q 1.7: **[5]**

Select the right definition of MPI in regards to parallel computing … (select only one option)

(a) MPI = Message Programming Interface.
(b) MPI = Message Passing Interface.
(c) MPI = Multiple Programming Interface.
(d) MPI = Multiple Processor Interface.
(e) MPI = Multi-processor Programming Interface.

Q 1.8: **[5]**

What is distributed memory? … (select only one option that is most accurate)

(a) In distributed memory, a machine has memory accessible by multiple processors.
(b) In distributed memory, processor nodes can directly access each other's memory.
(c) Distribute memory, is the same as port-mapped memory, you need to use a special instruction to access the shared memory.
(d) Distributed memory is where processor nodes have their own separate memory.
(e) Distributed memory is another name for 'inter-process communication' (IPC).

Q 1.9: **[5]**

What is the initial block in Verilog? … (select only one option that is most accurate)

(a) There is no such block or keywork in Verilog.
(b) It specifies how long a Verilog module takes to start.
(c) It is used in simulation to set up conditions and implement test benches.
(d) It is used to specify the initial values of an array (e.g. `initial x[2:0] = {0,1,0};`).
(e) It is a comment for the author's initials (e.g. `initial BS - Bob Saget wrote this`).

Q 1.10: **[5]**

A state machine may be designed as 'clock-triggered', also called a 'hot-running state machine'. In this case, if the clock is not running but an input changes what would happen in the state machine? (select only one option that is most accurate)

(a) No action will be taken as nothing is happening on the clock line.
(b) The input will be handled, possibly changing state of the state machine.
(c) It would only be a problem when running on hardware, irrelevant to simulation.
(d) The state machine will be reset.
(e) A transition to the next state will happen.

Part 2: FPGA Performance [30 marks]

Read over the following design scenario and then utilize the Question 2.1 page of the Answer Sheet to indicate how the FPGA concerned will be set up for answering Question 2.1 and then proceed with answering the other two questions in this part.

Question 2.1 [15 marks]

Have a look at the 2nd page of the answer sheet. You will see an FPGA internal structure that has logic elements (LEs) comprising various gates (AND, OR, XOR gates) and LUTs (i.e., look up tables). There are 15 LEs that are gates and 2 LEs that are LUTs. Remember that the LUTs are essentially memory devices where the inputs (in this case named little *a* and *b*) is an address and the 1-bit data, the *f* column, is a memory value that is output for a particular a, b inputs. As you can see each LUT has only 4 bits of memory (e.g., you could use LUT1 to implement an AND gate if you set LUT1.*f* = {0,0,0,1}). I have simplified it saying that to program this FPGA you can just draw lines from inputs to any LE, and from any LE to on to any other LE (but an LE cannot send it's own output back to one of its input, that would causes a violation)[1].

We want to implement (place & route) this Verilog module on this FPGA (the simulation is on https://www.edaplayground.com/x/pQiA, but that's just for review if you want after the test):

```
// EEE4120F Class Test 2021 Verilog placing and routing
// decider module.

module decider (A, B, C, D, X, Y);
  input A,B,C,D;
  output reg X,Y;  // won't bother with any resetting to space LEs.

  always@ (*) // i.e. just assume below is done all the time
    begin
      if ((A & B) != C)
        X = (B & C);
      else
        X = 0;

      if (D == (C & B))
        Y = ~D;
      else
        Y = 1;
    end // always@
endmodule
```

Yes, this may or may not appear a challenge to place and route this on the FPGA. But do give it a go, just hook up the inputs, LEs, and outputs to implement the circuit … and try to do so efficiently. Note you do not need to optimize these Boolean functions to save gates / LUTs! (that may take too long). To make things a little easier and save you time, I have put together a testbench and run the simulation on the next page if that is useful (you obviously don't need to implement the test bench as well, that is just for information).

[1] Yes, I know this is a simplification – usually you cannot connect one LE to any other LE as one needs a means to do such connections in a programmable way, as opposed to just physically writing things up like they did back in the day… but do not worry, Question 2.2 addresses this concern.

```
// EEE4120F Class Test 2021 Verilog placing and routing
// decider testbench code

module decider_testbench();
  integer i;
  reg  [3:0] counter;
  wire X, Y;

  // instantiate the decider module
  decider decider1(counter[0],counter[1],counter[2],counter[3],X,Y);

initial
  begin
    // toggle bits to ensure a safe reset
    counter = 4'b0;
    #10
    $display("ABCD -> XY");
    for (i=0; i<16; i=i+1)
      begin
        counter = i;
        #10
        $display("%b -> %b%b",counter,X,Y);
      end
  end
endmodule
```

OUTPUT OF RUNNING SIMULATION:

```
ABCD -> XY
0000 -> 01
0001 -> 01
0010 -> 01
0011 -> 01
0100 -> 01
0101 -> 01
0110 -> 11
0111 -> 01
1000 -> 01
1001 -> 01
1010 -> 01
1011 -> 01
1100 -> 01
1101 -> 01
1110 -> 10
1111 -> 00
```

Question 2.2 [8 marks]

We're going to do a performance comparison to a (somewhat slow) microcontroller. But first you need to calculate the speed of your design. You can assume all the inputs will change together, but you need to calculate how long it will take from when the inputs change to when the output is stable.

The type of LEs in the FPGA have the following response rates from the time an input changes to the time its output is stable:

LE Type	Response rate
AND, OR	5 ns
XOR	7 ns
LUT1 and LUT2	9 ns

Firstly determine the propagation delay of your answer for Question 2.1.

The CPU (i.e. the microcontroller) does it in a loop of 20 instructions, needing to do read and writes as well. It takes 200ns to do one loop (including jumping back to the start of the loop).

Indicate if the FPGA is providing a speedup or a slowdown over the CPU (select the option in the answer sheet). And indicate what the speedup is.

Question 2.3 [7 marks]

A problem is that this design is not really programmable as show, it's hard-wired. Explain how you could represent this design as a more re-programmable system. Already you can see there is some start towards this with the LUT, where data can be put in the LUT to implement various binary operations that can go from two inputs to one input. But doing just one big LUT is not an adequate answer, you still want to have various discrete LEs that get connected up, but how can that be done? Provide a short explanation of how this can be done.

Part 3: Verilog [20 marks]

Question 3

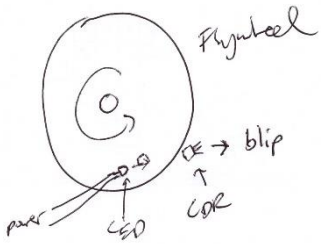

Illustration of flywheel setup

Welcome to the last question! There is only one part to this question. In this question we want to do a time measurement for how fast a flywheel is spinning – we just want to work out the period in clock ticks. The flywheel has a hole in it through which light shines whenever it passes the LED (shown on right). An input called *blip* gives a positive edge when the hole is detected (i.e. the LDR senses the light). The C code below shows how the period is measure, you are to convert this to Verilog. A starting point is given in the Answer Sheet.

```
// Flywheel period measure, prints out the period in loops (can assume num clks)
// runs on simple microcontroller that uses memory mapped I/O
#include <iostream>

using namespace std;

// memory mapped inputs and outputs:
unsigned char* read_blip    = BLIP_IN;    // a byte but assume it is a bit
unsigned     * write_period = PERIOD_OUT; // 32-bit output for measured period

int main() {
    // some needed variables / registers local to the module:
    unsigned      count    = 0;   // 32-bit counter
    unsigned      period   = 0;   // up to 32-bit period measure
    unsigned char old_blip = 0;   // value of blip in previous clock
    unsigned char new_blip;       // value of blip in this clock (can use blip)
    // cout only here for debugging, can leave out of answer:
    cout << "blip = " << (int)BLIP_IN << endl;

    while (1) {  // whenever a clk positive edge occurs do this: (i.e. @always)
        // increase counter
        count++;
        // read input:
        int new_blip = *read_blip;
        // check if blip has changed from off to on (i.e. rather check edge)
        if (new_blip && !old_blip) {
            period = count;
            count  = 0;
            // cout only here for debugging, can leave out of answer:
            cout << "period = " << period << "\n";
        }
        old_blip = new_blip;  // save old value of blip
    }
    return 0;
}
```

Sample run of program with debugging cout code left in:

```
blip = 0
blip = 1
period = 100
blip = 0
blip = 1
period = 100
blip = 0
blip = 1
period = 100
blip = 0
```

In this instance it measures 100 loops (assume clocks) for the period that the wheel is spinning at. If it is 1 millisecond per loop, the wheel is turning at 10 revs per second, or 600 RPM.

END OF TEST

CHEET SHEET

Appendix B: Verilog Cheat sheet

Numbers and constants

Example: 4-bit constant 10 in binary, hex and in decimal: 4'b1010 == 4'ha -- 4'd10
(numbers are unsigned by default)
Concatenation of bits using {}
4'b1011 == {2'b10 , 2'b11}
Constants are declared using parameter:
parameter myparam = 51

Operators

Arithmetic: and (+), subtract (-), multiply (*), divide (/) and modulus (%) all provided.
Shift: left (<<), shift right (>>)
Relational ops: equal (==), not-equal (!=), less-than (<), less-than or equal (<=), greater-than (>), greater-than or equal (>=).
Bitwise ops: and (&), or (|), xor (^), not (~)
Logical operators: and (&&) or (||) not (!) note that these work as in C, e.g. (2 && 1) == 1
Bit reduction operators: [n] n=bit to extract
Conditional operator: ? to multiplex result
Example: (a==1)? funcif1 : funcif0
The above is equivalent to:

```
((a==1) && funcif1)
|| ((a!=1) && funcif0)
```

Registers and wires

Declaring a 4 bit wire with index starting at 0:

```
wire [3:0] w;
```

Declaring an 8 bit register:

```
reg [7:0] r;
```

Declaring a 32 element memory 8 bits wide:

```
reg [7:0] mem [0:31]
```

Bit extract example:
r[5:2] returns 4 bits between pos 2 to 5 inclusive

Assignment

Assignment to wires uses the assign primitive outside an always block, e.g.:

```
assign mywire = a & b
```

Registers are assigned to inside an always block which specifies where the clock comes from, e.g.:

```
always@(posedge myclock)
   cnt = cnt + 1;
```

Blocking vs. unblocking assignment <= vs. =

The **<=** assignment operator is non-blocking (i.e. if use in an always@(posedge) it will be performed on every positive edge. If you have many non-blocking assignments they will all updated in parallel. The <= operator must be used inside an always block – you can't use it in an assign statement.
The blocking assignment operator **=** can be used in either an assign block or an always block. But it causes assignments to be performed in sequential order. This tends to result in slower circuits, so avoid using it (especially for synthesized circuits) unless you have to.

Case and if statements

Case and if statements are used inside an always block to conditionally update state. e.g.:

```
always @(posedge clock)
  if (add1 && add2) r <= r+3;
  else if (add2) r <= r+2;
  else if(add1) r <= r+1;
```

Note that we don't need to specify what happens when add1 and add2 are both false since the default behavior is that r will not be updated.
Equivalent function using a case statement:

```
always @(posedge clock)
  case({add2,add1})
  2'b11  : r <= r+3;
  2'b10  : r <= r+2;
  2'b01  : r <= r+1;
  default: r <= r;
endcase
```

Module declarations

Modules pass inputs, outputs as wires by default.

```
module ModName (
  output reg [3:0] result,  // register output
  input [1:0] bitsin,  input clk, inout bidirectnl  );
 … code …
endmodule
```

Verilog Simulation / ISIM commands

```
$display ("a string to display");
$monitor ("like printf. Vals: %d %b", decv,bitv);
#100   // wait 100ns or simulation moments
$finish  // end simulation
```