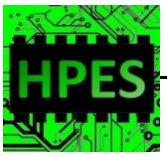

EEE4120F Quiz 4

Lecture 18 & 20 **DATE:** 4/5/2023

Name: _____ Student Number: _____ *Please fill in name!*

*This is a fairly short quiz, but it **is** for marks if you hand it in!*

NB: Please select only one answer option for each multiple choice question

CIRCLE/COLOR-IN ANSWERS FOR MULTIPLE CHIOCE QUESTIONS

TOTAL NUMBER OF QUESTIONS : FIVE (5) **TIME (mins):** **10**

#	Question - EACH QUESTION WORTH 1 MARK	Sec	W	%	X

Q1	This is an check for 'is the (thinking) power on' question…
	What will the following piece of Verilog code do? Select one option below.
	`// Code that may do something quite trivial`
	`module dothis ( input a, output reg b );`
	`always @(*) b <- a;`
	`endmodule`
		80	2	13%	

	[1] Send the value of b to a. [2] Send the inverse of value a to b. [3] Send the value of a to b. [4] Only send the value of a to b when there is a signal change. [5] The code is messed up, and probably won't compile.				
Q2	What is the difference between an conditional always and an unconditional always? Select only one option in your answer.	80	3	20%	
	[1] Both a conditional always and unconditional always has a sensitivity list, the different is that the one is followed by a * in round brackets, and the other has a list of sensititivities in the round brackets. [2] An unconditional always is not within another *always* statement which could block its operation. e.g.: always(*) a=b; *versus:* always(*) if (a) always a=b; [3] An unconditional always doesn't have a sensitivity list in brackets and activates or repeats as quick as it can; whereas a conditional always only actvates when certain sensitivities occur. [4] There is no such thing as a unconditional always in Verilog; the examiner is surely just making a joke, and not necessarily succeeding at that. [5] The unconditional always must be used only in simulation, for example to define a sequence of repeating steps that have delays between each step. But the conditional always is usable for both simulation and synthesis.				
	CODE TO VIEW FOR Q3-Q5 : See last page for Verilog program and equivelent assembly code. Yes, the Verilog is just using gates and boolean logic. This implements a 1-bit adder with carry. What we are going to work on is a speed performance comparison between the FPGA implementation (i.e. from the Verilog) and a equivelent program running on a CPU (which is technically within a microcontroller). But it's a really cheap basic microcontroller that - as the saying goes - is cheap as chips. Although more accurately cheaper than a fancy packet of chips (like 'Kettle Fried' sort).				

Q3	I'm not giving you a circuit diagram for the Verilog. But you don't necessarily need one. What you need to figure out is what is the speed at which this design would operate at. Note that you want to know that if one of the inputs, A, B or Cin, changes at what time after that would the it be safe to read an output (i.e. the last one to change). Assume all gates in this design operate at 10ns. You need to show your motivation and (if need) calculation in your answer.				
	_____ _____ _____ _____ (use back of last page if need more space)				
Q4	Now, what you no double anticipate: go ahead and work out what speed that assembly program is going to work at. Basically, assume that main() function repeates endlessly (you can ignore the JUMP command that the goto would translate into). We want to know how long does one iteration of the main() function take. The processor is clocked at 10MHz, each instruction takes just one clock cycle to complete (it is not a pipelined processor). Now use your answer to Q3 to calculate the speedup of the FPGA over the CPU.... or if you find that the CPU is faster indicate its speedup over the FPGA, be sure to indicate what you are considering the faster. Show your working and/or motivation for your answers.	210	5	33%	
	_____ _____ _____ _____ Speedup of _____ over _____ is _____				
Q5	Configuration architecture were discussed in lecture 20. When we're dealing with FPGAs, what exactly is meant by the concept of configuration architecture? Select the most correct option below: [1] An FPGA *is* simply a type of configuration architecture, in that it is an interconnected set of logic blocks and elements that gets configured. [2] An FPGA is the governing component of a configuration architecture, it is essentially the machanism for implementing a configuration architecture. [3] An FPGA is configured or programmed by the configuration architecture which is typical separate circuitry outside the actual FPGA. [4] An FPGA connects to external hardware, such as the host PC, via the configuration hardware. [5] An FPGA does not have to have associated configuration hardware, as term 'configuration hardware' refers to the user interface which is optional.	210	5	33%	
	TOTAL :	**580**	**15**	**100%**	
	Time : time est. in sec W : Weighting of question % : How much question counts		X : Office use		

CODE

VERILOG CODE FOR DIGITAL CIRCUIT:

```
// single-bit full adder with carry
module onebitadder ( A, B, Cin, Cout, Sum, CCheck );
  input  A, B, Cin;
  output reg Cout, Sum;
  output reg CCheck;    // should be the same as Cout if
                        // we got the logic right
  // named wires
  wire   xorab, andab;  // temp wires

  // perform the summing
  assign xorab = A ^ B;
  assign andab = A & B;
  assign Sum   = xorab ^ Cin;
  assign Cout  = (xorab & Cin) ^ andab;
endmodule
```

C CODE and equivelent ASSEMBLY code for doing comparison. Note that each assembly instruction corresponds to a machine (i.e. CPU) instruction. The CPU runs at 10MHz. It's a really basic PIC type processor.

```
/* Program to implement 1-bit adder with carry in and out */

/* fuction to read a word from a port address */
extern unsigned input ( unsigned& X, unsigned address );

/* fuction to write a word to a port address */
extern void output ( unsigned X, unsigned address );

/* main program to implement 1-bit adder with carry in & out */
int main ( ) {
  // declare variables to use
  bit A,B,C;        // end up being registers A-C on CPU
  unsigned D, X;
/*C code            || equivelent assembly code */
  input(A,0x70);    // IN X,0x70;  SWP X,A
  input(B,0x71);    // IN X,0x71;  SWP X,B
  input(C,0x72);    // IN X,0x72;  SWP X,C
  X   = A + B + C;  // ADD X,C
                    // ADD X,A
                    // ADD X,B

  // send out the sum value
  output(X,0x80);   // OUT X,0x80;  @ ie. send X to address 0x80
  // calculate Cout, i.e. 1-bit carry out, from D
  if (D&2)          // AND X, 2
     D = 1;         // SETZ D  -> set reg D if zero flag set
     else D = 0;    // CLRZ D  -> clear reg D if zero flag not set
  output(D,0x81);   // SWP X,D; OUT X,0x80;
  goto main;        // bit nasty code... just loops back to start
}
```