EEE4120F Quiz 4 based on:

Lecture 18 and 20

DATE: 4/5/2023

ANSWERS!!!

Please fill in name

This is a short quiz, but it ***is*** *for marks!*

NB: Please select only one answer option for each question

CIRCLE/COLOR-IN ANSWERS FOR MULTIPLE CHIOCE QUESTIONS

TOTAL NUMBER OF QUESTIONS : FIVE (5)

TIME (mins):

#	Question - EACH QUESTION WORTH 1 MARK	Sec	W	%
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.	80	2	13%
	`// Code that may  do something quite trivial`			
	`module dothis ( input a, output reg b );`			
	`  always @(*)  b <- a;`			
	`endmodule`			
	[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.			
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.			

	These first two need to complete first, while it is completing the Sum and Cout will still be working on the previous values.
	  assign xorab = A ^ B;
	  assign andab = A & B;
	So, that will take **max 10ns**, they will run at the same time. Yes, **I** did use blocking (=) assignments, but it is not in an always@ block, if it was in an always@ then the first xorab would be run and latch execution of the next one, so they would take max 20ns if they were in an always@.
	
	Now for these two:
	  assign Sum = xorab ^ Cin;
	  assign Cout = (xorab & Cin) ^ andab;
	These's no dependencies between these, so we choose the link with the longest delay, which would be calculating Cout. Which would be 2x10ns = **20ns**
	In total it's just 10ns + 20ns = **30ns** between a input change and stable outputs.
	So, this little circuit is going to run pretty fast!			

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%
	It is a totally sequential CPU function. There are 15 instructions that execute one after the other. The clock runs at 10MHz, so a clock period (the maximum time it takes for an instruction to complete) is 100ns. So one iteration of the main function will take 1500ns, i.e. 1.5us. Not fast for simple logic. So the speedup of the FPGA over the CPU would be Tp1/Tp2 = 1500/30 = 50 Speedup of __FPGA___ over __CPU___ is ___50______ (and hopefully, like your lecturer, you are rather happy at the easy divide 😊)			
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:	210	5	33%

	[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.			
	TOTAL :	**580**	**15**	**100%**
	Time : time est. in sec W : Weighting of question % : How much question counts	X : Office use		

!

####
X