UG Exam Performance Feedback
First Year
2017/2018 Semester 1

COMP12111 Fundamentals of Computer Engineering

Paul Nutter
Christoferos Moutafis

Comments Please see the attached report.

COMP12111 Exam Feedback 2017/18

Q1 - CM

a) Hierarchy should not be confused with abstraction.
Abstraction is hiding the underlying detail in order to aid the designer. Abstraction makes it possible to design, build and test complex systems.
Hierarchy is the concept of breaking down the design into more manageable blocks, which are simple to implement and reuse.
An example, that we have encountered, is the 4-bit adder. It is composed of four 1-bit adders, which in turn are made from half adders. We design the simpler half-adder.

b) This was often incorrect. i) With 1 byte, we can represent 2^8 = 256 states.
Also, for ii), -129 -> overflow: input requires 9 bits; 8 bits were specified, outside the range -128 to +127.

c) This was the NAND gate. The NAND function is true when either OR BOTH of the inputs are zero. This was often confused. It is called a universal gate since combinations of NAND gates can be employed to accomplish any of the basic operations (they can thus produce an AND gate, an OR gate, and an inverter). The standard, non-inverting gates can't do that. This was largely missing from the answers.

d) Here, the equivalent schematic representation was only part of the question. A Boolean expression for the logical function implement was required. The equivalent logic gate here was the XOR gate.

e) Here, it is not enough to describe the DeMorgan's theorem in words. The actual formula of the theorem needs to be stated. Also, the "sum of products and products of sums" is not what is requested, De Morgan's in its simple pure form is requested. Stating the formula of the basic DeMorgan's theorem was what was expected here, for the first part of the question. The alternative expression derived was mostly correct by most. Not everyone however concluded that this represents the XOR function.

Q2 - PWN

The question consisted of 5 small questions, each worth 2 marks. On the whole I felt the questions were answered well, and I have no concerns about a general lack of understanding in an of the areas covered.

a) This question required you to discuss why NRZI encoding is used in USB, and to sketch the encoded waveform for the user data given. One mark was awarded for each. The discussion on NRZI should focus on the need to introduce regular transitions in the transmitted data to make recovery of transmitted data easier, with a transition for every 0 in the transmitted data. The waveform would look like:

1 0 1 0 1 1 1 1 0 0 1

where the signal level changes (a transition) each time a 0 is encountered in the data stream. Note: the signal would be inverted if we assume the signal level is initially 1. To get the full mark the data pattern in relation to the correct waveform must be shown. Issues were:

- Not discussing the purpose of NRZI – this was a very common issue.
- Not pointing out that NRZI is used to ensure regular transitions in the transmitted waveform.
- Discussing how USB works , i.e. D- and D+ signals etc – this was not asked.
- In correct waveforms assuming a pulse for each 0, rather than a single transition.

b) The aim of this question is to identify the most suitable clock frequency, from those available, for the critical delay given. The critical delay is 25ns, so you must ensure that the clock period is not shorter than this. There are four clock speeds available, 10MHz, 20MHz, 50MHz, and 100MHz, which correspond to clock periods of 100ns, 50ns, 20ns, and 10ns. Consequently, 20ns <25ns, so 20MHz is the most appropriate choice. A mark was awarded for the correct answer, and one for some reasoning/calculation to arrive at the answer. Issues were:

- Calculating a critical delay of 25ns gives a frequency of 40MHz (which is correct), but then saying this is close to 50MHz, so 50MHz is the correct choice. You need to remember the inverse relationship between time and frequency. 50MHz has a shorter period than 40MHz, and so is too fast for the critical delay identified.

c) This question was relatively easy, although not always answered correctly. You are required to discuss what is meant by non-volatile memory – retains data if the power is removed, give an example – ROM, and state why is some non-volatile essential in a computer system – to provide boot up information. Issues were:

- Not giving an example – make sure you read the question!
- Stating that it is essential to store user files – well yes, but then you need boot up information in order for your computer to understand what a HDD is and how it reads information from it – this is the bios!

d) For this question you are required to identify that Direct Memory Access (DMA) – and not any other variations – is used to transfer large blocks of data to/from memory – this was one mark. In additional, a brief discussion (not a long rambling

discussion) of its operation is required. I was looking for two things: that a DMA controller takes control of the system bus to enable data transfer, and that this is done independently of the CPU – ½ mark each. Issues were:

- Not identifying DMA as the approach – it isn't LD/ST instructions, or parallel transmission, or the use of buses or fetch/execute cycles, or any other random answer!
- Not identifying that it is performed independent of the CPU.

e) This was an easy question, although not everyone was able to identify by how much the PC should be incremented in the example given. The PC holds the address of the next instruction – 1 mark. In the example given the data is 16 bits, but each memory location is only 8 bits. Hence, a 16-bit value is store in two memory locations, so the PC must be incremented by 2. Issues were:

- Not being clear that the PC holds the address of the next instruction – it's not the current instruction for long.
- A random variety of PC increment values including: 1 (in MU0 maybe), 8 (maybe getting mixed up with memory width?), a byte (maybe getting mixed up with memory width?), 256 (not idea where this would have come from!) … quite a lot of wrong answers here.

Q3 - CM

This was intended to test basic knowledge about multiplexers, Finite State Machines (FSM) and Verilog implementation.

a) This question was mostly correct, but, often, included mistakes in the description of the mux and the number of select lines for the example. The mux is a switching device, that selects any (e.g. binary) information from one of its many input lines/buses and directs it to a single output line/buses. 3 select lines are line for an 8:1 mux.

b) There was considerable confusion in this question. The 4 inputs, 2 select lines, and output, needed to be clearly indicated, for the example. A multiplexer was clearly asked, not a demultiplexer.

c) The implementation here could be done either with Blocking assignments or with continuous assignment. Including a default case was often forgotten.

d) There was some confusion here. A modulo-12 counter FSM would have 12 unique states. You need 4 bits to describe the 12 unique states.

e) This was also intended to test/demonstrate the need for a clean design; there was mixed success here. In particular, transition arrows need to be clear.

- There was only a need for one loop back; the reset in the "0"/zero state.

- There was only a need to indicate the Reset=1 and the Reset=0 transitions.
- The number of states here is 12, not 13.
- There was no need to indicate a transition condition between the 11th and the 0th state.
- We need to read the question. The FSM Diagram was requested NOT the table of transitions.

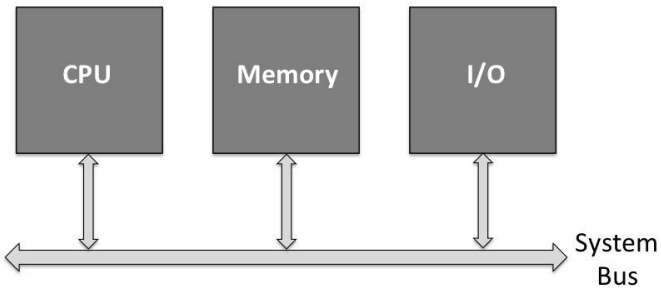
f) This question was mostly answered correctly (except when more states than needed were included).

g) There was large confusion in this question. The always block has “posedge clock” in its sensitivity list; nothing else is appropriate here. Non-blocking statements need to be used. This was very often incorrect. The reset functionality and default case was often missed or incorrectly implemented.

Q4 - PWN

This question was answered relatively well, particularly with the last part (part g) which involved writing Verilog code. It is clear that on the whole the cohort has a good grasp of the Verilog language.

a) This question required a diagram to explain the concept of the three box model (1 mark), accompanied by a brief discussion of the role of the system bus (1 mark). In the first case a diagram as follows was all that was required for the mark available:



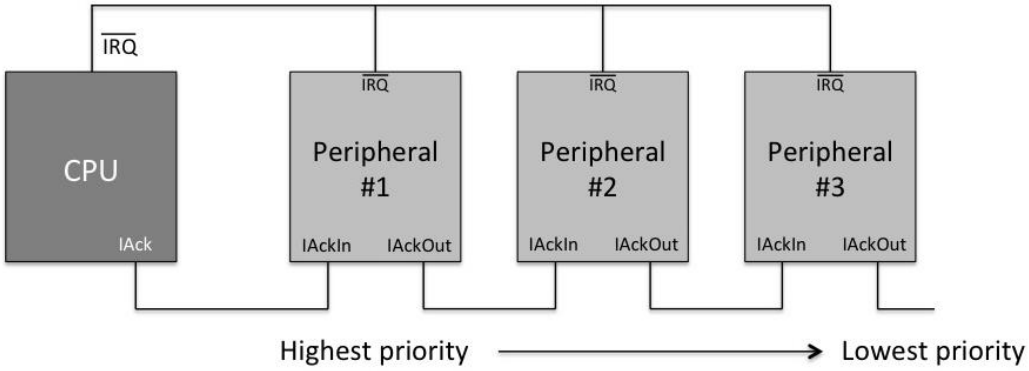
where the three boxes are identified along with the system bus connecting all three together. In the case of the 2nd part of the question, I simply required an answer to point out that the system bus contain data, address and control buses. Issues were:

- Not providing a diagram (read the question!).
- Not including the I/O block, “three box” model.
- Providing a diagram of the FSM (combinatorial block/register/ combinatorial block arrangement)
- The system bus not going to all three components.

b) This was an easy question. I was simply after an answer that identifies the role of the interface as forming the bridge between the CPU and I/O device to convert signals to an appropriate format etc. On the whole there were few issues with this question.

c) Relatively easy question for a couple of marks. You were simply required to discuss the advantage of interrupts over polling. The discussion should include a brief overview of the two approaches and the advantage that interrupts gives stated. On the whole, no issues, marks were lost for simply not providing enough detail.

d) This question produced some very interesting answers, somewhat random in nature. This is straight out of the notes and I was after a diagram like so:



1 mark for the blocks, 1 mark for the common interrupt, and 1 mark for the daisy chained acknowledge signal. The question doesn't ask for an accompanying discussion, so there's no need to provide one, just a diagram was required. Providing a number of boxes with lines between them and labelling one highest priority and another lowest priority was not enough to be awarded any marks.

e) Easy question. Almost everyone got this one right. In order of priority: External hard disk, scanner, printer. No diagrams were required, so I've no idea why some answers illustrated this using a diagram!

f) This question proved a difficult one to answer, with only a handful of students achieving full marks.
   i) The discussion should centre on the use of the mask register for enabling/disabling interrupts. Setting the appropriate bit in the mark resister to 1 will enable the corresponding interrupt to be enabled since the interrupt bit and corresponding mask register bit and ANDed before going to the priority encoder. The main issue here was not identifying the ANDing action. There were also a number of completely random, incorrect answers.
   ii) This required a brief discussion of the role of the priority encoder. I wasn't looking for an answer that simply said, it determines the priority of the interrupts – this is somewhat obvious from the name. Instead, I was after an answer that added weight to this, in particular, recognising that the priority encoder raises the IRQ line to the processor whenever and active interrupt goes high and in addition, provides two address bits to form an address where the interrupt routine can be found.
   iii) This was easy – the IRQ is raised whenever an active interrupt is raised.

g) This question was relatively well answered, which was very pleasing. The key here is to create a block of code that provides priority, which is achieved by (in the easiest sense) by using nested if … else statements. Example code may be:

```
begin
 if(int4)
  address = 2'b11;
 else if(int3)
  address = 2'b10;
 else if(int2)
  address = 2'b01;
 else if(int1)
  address = 2'b00;
 else
  address = 2'b00; // could use x, doesn't matter

 if(int4 || in3 || int2 || int1)
  irg = 1;                  // at least one int high
 else
  irg = 0;
end
```

Marks were assigned for correctly assigning values to irq and address, for specifying numbers correctly, i.e. 2'b11, for valid, working Verilog code, and for providing a correct working solution. Issues were:

- Not getting the priority right – starting with int1
- Having more complex Boolean expressions in the if statement, i.e. (int4 == 1 && int3 == 0 && int2 == 0 && int1 == 0) – this isn't necessary if you think how the if … else if … statement works.
- Not setting irq and address for ALL cases – remember, this is a combinatorial block of logic, there should be no dangling elses.
- Treating irq or address as inputs
- Copying the header – no need, you are only asked to provide the missing code – wasting effort and time by doing so.