Comments  Please see the attached.
COMP12111 Exam Feedback 2014/15

P W Nutter, & V Pavlidis

The average mark for the paper was 55%, with almost all students answering questions 1, 2 and 3. Taking into account that the lab average is around 75%, I expect the overall course unit average to be around 65%.

Q1 (PWN)

Observations: A number of students answered more than the 5 requested questions, in some cases all 8 questions were answered. This was a waste of time. Why? No extra marks can be awarded for answering all the questions, and more time could have been devoted to answering the other questions, particularly ones which you were less comfortable with, in order to maximise your overall mark.

a)

A common problem with answers to this question was getting abstraction mixed up with hierarchy. Remember, hierarchy is breaking a problem down into much simpler, and easier to implement, blocks that can be reused. So an example of the 4-bit adder being constructed from 1-bit adders relates to hierarchy and not abstraction.

Abstraction allows the designer to hide the detail in a design to make it easier to design complex systems. An example of abstraction that you are all familiar with is the symbol.

b)

On the whole this question was answered fine. However, there were a few common problems:

- The simulation stops at 200ns. Almost everyone who answered this question produced a set of waveforms that extended to 250ns.
- A number of answers started the clock with a value of 0, not 1 as it is initialised to in the Verilog code.
- A number of answers didn’t show a time axis – a waveform is only useful when you know the timescale!
- Not knowing how to represent a bus when drawing a waveform!

c)

On the whole this question was answered well. However, a large number of answers didn’t quote DeMorgan’s theorems (as equations), which is what the question asked you to do – this was 1 mark!
d) Not many answered this question. On the whole sensible diagrams were provided, however, the clock to the registers was not always shown, and in some cases the control was shown connected to the registers, and not the combinatorial logic in the datapath, which is what it is controlling. The discussion required a description of the structure of the datapath, and that it operates on data flowing through it, with the control determining what the datapath does via a finite state machine. 1 mark was awarded for the diagram, 1 mark for the discussion.

e) The question asked to produce a Boolean expression for the circuit shown, NOT a simplified expression. 1 mark was awarded for extracting an equation directly from the circuit diagram, and 1 mark was awarded for recognising that the circuit is an XOR gate – but it must be clear how this decision was arrived at, only \( \frac{1}{2} \) a mark was awarded otherwise.

f) This question was answered well overall. Some problems:

- not showing all the transitions (as you are asked to),
- not labelling all the transitions,
- labelling transitions with variables that are not given, and not drawing the right number of states.

1 mark was awarded for drawing a state transition diagram with 5 states, 1 mark was awarded for getting the transitions correct – stay in the state when stop is high, leave a state when stop is low.

g) Easy question, although there were some issues:

- giving answers that weren’t 5 bits (read the question!),
- not recognising that -16 is the smallest number that can be represented using 5-bits in two's-complement, so getting part iv) wrong by saying it cannot be represented.

\( \frac{1}{2} \) mark was awarded for each correct answer.

h) There was actually a deliberate error with this question in that not all combinations for the states of P and Q were given in the diagram. I wanted to see if you could suggest a sensible logic gate from the information given. The design is an XNOR gate (state P=1, Q=0 isn’t given). A number of answers lost marks
because they didn’t provide a proper truth table, or a truth table with multiple entries.
Q2 (VP)

A compulsory question marked out of 10. Answer 5 from 8. The majority of students attempted this question even if they attempted only few parts or in some cases more than five parts.

a)

Many tried this question as was discussed a lot in the class and many did get two marks. One mark was given for the correct order of the memory types and one for the correct mapping between technologies and level of memory hierarchy.

Problems:

- Some answers did not indicate the order and assumed that it was implied but this at the time can be an indication of a student not being sure about the answer. Half mark was taken off for that. Most of the students though clearly indicated the order by either numbering or drawing a CPU box and placing the memories in decreasing logical proximity. The best approach would be to indicate the processor and then the different memories, e.g., CPU.... etc.
- The concept of non-volatility was not understood from some students who claimed using DRAM as a non-volatile memory!
- Some answers also failed to map the memory types with the technologies and one mark was lost for that. Simply listing the characteristics of each technology was not the appropriate way to answer the question.

b)

Popular question. The answer was literally three sentences and several students got it right. However, there were incomplete answers as well. 1 mark was given for the arithmetic operations and a successful example, e.g., addition and 1 mark was given for logical operations and a proper example, e.g. a bitwise AND.

Problems:

- Giving instruction mnemonics of the MUO as an answer, this is not what we were looking for as each processor has its own unique mnemonics but the underlying operations within the ALU are fundamentally the same. No marks were given for just mnemonics.
- Providing instructions executed by the CPU as examples! The CPU not the ALU performs these instructions. Executing however these instructions may include ALU operations (e.g., addition) to determine for example the proper address. 1 mark was lost for giving load/store as an answer.

c)

Almost everybody attempted this question. All answers but the last were false. Several students left unanswered the parts that they were unsure of, which was a prudent technique to avoid marks being lost. No negative mark was given for
this question but no marks were given either if at least two out of the four parts were answered wrongly.

Problems:

- Several students have problems with reciprocals. A weakness in mathematics?
- Most remembered the role of interrupts for I/O communication.
- Some forgot the density relationship between SRAM and DRAM.
- Some also gave careless answers on the decoder question, which was unexpected as the answer is on a slide of the course.

d)

Many attempted this question but only about a fourth of the answers was correct. One mark was given for saying that the timer is a circuit that provides a timing reference and another mark was given for saying that this can be programmable providing a higher flexibility.

Problems:

- Many students said that timer and “clock” is the same thing! The timer is a circuit that produces a signal as a timing reference not a signal per se (such as the clock signal).
- Few pointed out that a timer could be programmable.

e)

About half of the students tried this question even though it was an easy one. One mark was given for stating the direct memory access (DMA) mechanism and another one for saying the CPU can perform another task while the memory transfer by the DMA is taking place.

Problems:

- Some students mentioned transferring some data to the cache but no mention how the data will be transferred to the cache was given.

f)

A very popular question and in general was answered correctly. One mark was given for saying that the control unit is essentially an FSM and another for describing the operation of the control unit in principle.

Problems:

- Often the FSM part was missed and one mark was lost for that.
g)

A very popular question, one mark was given for referring multiplication and division or serialization and deserialization as operations and another mark for correct examples of these operations.

Problems:

- Only shift operations were mentioned but not the purpose of these operations. e.g., to multiply/divide a number. No marks were given for these answers.
- Some answers stated only multiplication/division by two, while it's for any power of 2 in general. A mark was lost for this answer.

g)

About half attempted this question but most failed to answer this correctly. This question was referring to three specific methods described in three slides of the course. ½ mark was given for improving the technology (i.e. faster transistors), one mark was given for saying that speed can be gained by redesigning the critical path so that more instructions per second are executed, and ½ mark was given for redesigning the architecture of the processor.

Problems:

- Many answers were beyond the scope of the course but this is not what we were looking for.
- Many answers included different architectural approaches and ignored technology and design of the critical paths.
- Also several answers mentioned to increase the clock frequency!! But this cannot happen unless the critical path is shortened so that a higher frequency can be supported.
Q3 (PWN)

This question was directly related to activities undertaken in the lab.

a)

This question asked you to identify the faults with the 4:1 MUX Verilog example given, which selects from one of four 5-bit inputs.

There were quite a few errors:

- Input A is declared as a reg. This is an error because inputs are not assigned values, so they cannot be defined as reg. Inputs are always wires, which don’t need defining because ‘wire’ is the default type.
- Output Q should be declared a reg as it is assigned a value with a blocking assignment (remember the rules for assignments) – I only awarded ½ mark if the reason why Q should be declared as reg is not given.
- The sensitivity list for the always block should list ALL the input variables, not just A and B, so A, B, C, D, and Sel (or just use * as it is a combinatorial design).
- There is a missing ‘;’ after the 2’b11 case – all statements should end with ;.
- There is a missing ‘;’ after the default.
- The bus width for the default should be 5 bits, not 4, as the inputs and output are 5-bits, so 5’hX.
- There is a typo for Sel, it is defined ‘Sel’ but used as ‘sel’ – remember Verilog is case sensitive.
- There is a missing ‘endcase’ which is required for all case statements.

Each error is worth 1 mark, including an explanation, or stating the correct answer, up to a maximum of 6 marks for the question.

Some common problems/misconceptions:

- There were a number of cases where the problem was identified, but an explanation, or possible solution was not given – 1/2/ mark was awarded in this case.
- A number of answers only listed a small number of errors. Of course, this could be because they couldn’t spot them, however, 6 marks were available indicating that there is a good chance that there are 6 errors.
- Stating that X is not a valid value, or that outputs must be assigned a value 0 or 1, for the default case. This is not the case. X is don’t care. A default value is often assigned ‘X’ in order to trap any errors related to the variable in the case statement, such as it being undefined, during simulation.
- A default case is not needed as all the conditions for ‘Sel’ are covered. The default can be used to trap conditions where ‘Sel’ is undefined, see previous comment.
- A number of answers stated that ALL inputs should be declared as reg. As stated above, we are not assigning values to inputs, so they should be declared as ‘wire’, which is the default case so doesn’t have to be stated.
• “The case statement needs a begin and end” – a case statement doesn’t have a begin and end. You can have a begin and end for each case if more than one assignment is required.

• A number of answers identified that the sensitivity list is missing C and D, or alternatively should be just ‘Sel’. Remember, this is a combinatorial logic block, so any input variable used should be listed, or, you just use “*”.

• Stating that there is no indenting! Indenting is a style issue, the code will still work if corrected, but not indented.

• The inputs and outputs should be 1-bit values. Why? The device is a 4:1 MUX, it just happens that the inputs and outputs are 5-bits, so it’s a 5-bit 4:1 MUX.

• Sel needs to be a 3-bit variable – this I didn’t understand, but it was stated on a number of occasions. 4 inputs requires 2 control lines \((2^2 = 4)\), as given.

• Stating that the always block must have a begin and end. Indeed it should, if more than one assignment in the block. Here we don’t need a begin and end, as only one assignment is made.

• Stating that having a semi-colon after the header is incorrect. This is how you define a header in Verilog!

b)

This was a relatively straightforward question that required you to assign state codes to the four states, and then produce a state transition table. There is no excuse not to be able to do this, examples were covered in lectures, and there are interactive examples available on the course unit Moodle page. Consequently, I was expecting most to get this almost 100% correct – how wrong was I! I won’t give the answer here, as it is so easy. However, I will list some of the common problems/misconceptions:

• In a number of cases the state code for each state was not explicitly identified. How do I know what state ‘00’ is?

• There were a large number of examples where the table was not presented in the “usual” format. I was generous when marking these. However, there is no reason why they should be different. See the lecture notes for examples.

• In a number of cases the state code was not given in the table, just the state name, i.e. S0. The state transition table is to help with the translation of an FSM design to implementation. You must use the state codes, otherwise it is of no help.

• There were a large number of answers where the output Z was not listed in the table. Remember, the goal of the state transition table is to list the next state value, as a function of the current state and any input signals that control, the transitions, as well as list the state of any output signals as a function of the current state.

• Instead of listing the state of the input X, or the output Z, the values were listed as \(X, \overline{X}, Z, \) or \(\overline{Z}\).

• Some answers gave state codes that were 3-bits. Four states = 2 bit state code \((2^2=4)\).

• Giving a value of ‘X’ for the output. Outputs should always be assigned a value, i.e. 0 or 1 for valid states. The only time you will ever see an output
assigned to ‘X’ is in the Verilog implementation, where the default case of a case statement may be assigned a value ‘X’ in order to trap errors during simulation.

c)

This question asked you to take your design for the system given (part b) and translate to a Verilog implementation. Again, I felt this would be a relatively straightforward task due to your experience in the lab of writing Verilog modules.

The answer only required always blocks for the next state logic and output logic to be provided.

A possible solution is (probably the simplest):

```verilog
// next_state logic
// 6 marks awarded
always @ (*) // combinatorial
  case(current_state) // decode current_state then X
    2’b00: if (X==0) next_state = 2’b00;
         else      next_state = 2’b01;
    2’b01: if (X==0) next_state = 2’b10;
         else      next_state = 2’b01;
    2’b10: if (X==0) next_state = 2’b00;
         else      next_state = 2’b11;
    2’b11: if (X==0) next_state = 2’b10;
         else      next_state = 2’b01;
    default: next_state = 2’bxx; // trap errors
  endcase

// output logic
// 3 marks awarded
always @ (*) // combinatorial
  if(current_state == 2’b11) // Z only asserted in S3
    Z = 1;
  else
    Z = 0;
```

Some common problems/misconceptions:

- A large number of answers wrote out the complete module, copying the code given in Figure 6. Where does the question ask you to do so?
- Assigning values to “current_state” in the “next_state” always block. Remember the structure of the FSM we cover in Lectures. We have three always blocks, one for implementing the next state logic (a combinatorial logic block with a * in the always sensitivity list), one which assigns the calculated next state to current state at each rising edge of the clock (a
sequential logic block with posedge clock in the always sensitivity list) – this was given, and one that determines any output signals from the current state (a combinatorial logic block with a * in the always sensitivity list).

- A significantly high number of answers placed the clock in the sensitivity lists for the two always blocks required. There are no excuses for this, as you have seen a number of examples in lectures, labs, and on the Moodle page. These are combinatorial logic blocks, they are not controlled by the clock, so they should have a ‘*’ in the sensitivity list.

- A number of answers just put current_state in the sensitivity list. This is fine for the output block, as only current_state controls the output assignment. However, for the next_state logic, the input X also controls the next state assignment, so X should be included too (or just use ‘*’).

- Including reset in the sensitivity list/always block. The question clearly states that the reset operation has already been included. The reset operation is performed as part of the current state assignment (register block), which was given.

- When writing code, you should always comment it. Marks were awarded for providing comments for the two always blocks (well at least making some effort to comment your code). Not many answers included commenting.

- Using non-blocking assignments, <=. These are combinatorial blocks, they should use blocking, =, assignments.

- Missing ‘;’ at the end of assignments (Verilog – 101).

- Missing the else from an if … else. The blocks you are required to complete are combinatorial logic blocks, if you miss out the “else” then this may result in a latch being introduced to hold the value of a variable. Do not make the assumption that if the value doesn’t change, then you can ignore it, you must always give an assignment.

- Repeated if statements, i.e. if ... if ... if ..., instead of nested if ... else statements, i.e. if ... else if ... else if ... else.
Q4 (VP)

Very few answered this question and I thank the brave ones for that! Marked out of 20.

Although few students tried this question, I believe it was not difficult but it required to have digested the taught material throughout the term. Let's look at each part.

a)
As the size of a single memory chip is given the number of bits of the address bus used for chip select should first be determined. This number to the power of two yields the number of chips of the system and one mark is awarded. Another mark is awarded for giving the address space in hex format without the fault and a third mark is given for the resulting address space when the fault is considered.

b)
One mark was given for describing the four basic signals of the USB protocol. One mark was given for providing the binary sequence for the string “UoM” and another mark was given for drawing the corresponding waveform. No bit stuffing was necessary for transmitting this string and either a zero or one could have been assumed for the idle state of the channel.

c)
The most difficult part of this exercise but a similar example was given in class and was provided as a hand-out as well. One mark was awarded for the full instructions using the given mnemonics and three marks for filling out correctly the table with the contents of the registers.

d)
This part was based on Amdhal's rule. One mark was given for pointing out that the system is not balanced, another mark for increasing the I/O bandwidth to yield a more balanced system and another mark for determining the right frequency which was simply the inverse of the given delay of the interconnection.

e)
This question could be answered by using the material in the handouts. One mark was given for the schematic diagram of the bidirectional transceiver consisting of two tri-state buffers. A second mark was awarded for describing the operation of the drawn circuit and a third mark for describing some common uses of tristate buffers. The last mark was given for pointing out the illegal states for the control signals of the tri-state buffers.
None attempted this part to great disappointment as the answer consists of few gates. It only required a smart use of the three input gates which were used in a similar manner in Exercise 1 of the labs. Two marks were given for the schematic of the circuit and one mark for determining the delay along the circuit, which was a simple addition!