UG Exam Performance Feedback
First Year
2018/2019 Semester 1

COMP12111	Fundamentals of Computer Engineering	Paul Nutter Christoferos Moutafis

Comments Please see the attached report.

COMP12111 Exam Feedback 2018/19

Section A – Questions across the material, set by CM and PWN

Multiple choice questions – these are automatically marked in Blackboard, so we can't provide feedback on students' misunderstandings. The correct answers and brief explanation are given below:

1. Correct answer C: 1024 by 16384. The most significant 10 bits are decoded to X (2^{10} = 1024) and the least significant 14 bits to Y (2^{14} = 16,384). Average 0.86/1.

2. Correct Answer: B. If the branch condition is satisfied then the result must be non-zero, so for the branch not to be satisfied then N = 1. Average 0.81/1.

3. Correct Answer: B. 01111110 is 126. 10110101 in decimal is -75, so the answer is 51, which is 00110011. Average 0.77/2.

4. Correct Answer: C. The instruction fetched goes into the instruction register. Average 0.94/1.

5. Correct Answer: C. A critical path of 50ns will limit the clock frequency to 20MHz. If the critical path is reduced by 20% then it becomes 40ns, which gives a new clock frequency of 25MHz. There was an error in the question in that there were two answers the same – this had no effect as the two the same were incorrect answers. Average 1.94/2.

6. Correct Answer: C. A 2's complement 8-bit number can represent values in the range -128 to 127. -129 is outside this range so an overflow condition has occurred. Average 1.91/2.

7. Correct Answer: B. Average 0.99/1.

8. Correct Answer: A. The average access time is given by

   $t_{access} \approx CHR.t_{cache}+(1-CHR).t_{RAM}$

   So in terms of CHR, this becomes

   $CHR \approx (t_{access}-t_{RAM})/(t_{cache}-t_{RAM})$

   $CHR \approx (15-25)/(5-25) = 0.5 = 50\%$

   As the answer requires some working out, then the full 2 marks are awarded for a correct answer. Average 1.81/2.

9. Correct Answer: D. We stay in state 1 when the conditions A or A'.B are not followed, i.e. A'B', which using DeMorgan's theorem gives (A+B)'. Average 1.59/2.

10. Correct Answer: C. A transition in the waveform occurs for every '0' transmitted. Average 0.78/1.

11. Correct Answer: A. Average 1.76/2.

12. Correct Answer: D. The interrupts from the peripherals are active-high, but the resulting interrupt to the processor must be active-low. The truth table will be

P1	P2	int
0	0	1
0	1	0
1	0	0
1	1	0

    Average 0.82/1.

13. Correct Answer: B. Average 1.83/2.

Section B

Questions 14-18 set by CM. Questions 19-24 set by PWN.

Q14.

This question aimed to test understanding of Verilog procedural blocks and it was generally answered well.

- Blocking statements are committed sequentially and non-blocking statements concurrently (1 mark).
- Blocking and non-blocking assignments are used within always blocks (1 mark).
- Non-blocking assignments are used in the design of sequential systems (1 mark), i.e. when there is timing dependency in the sensitivity list of the always block (1 mark).
- The data type reg is associated with variables assigned using blocking and non-blocking assignments (1 mark).

Issues:
The final question, what data type is associated with variables assigned using blocking and non-blocking assignments caused some confusion. The ***data type reg*** is associated with variables assigned using blocking and non-blocking assignments, not wire. Another thing to note is that Continuous assignments are defined outside of procedural blocks.

Q15.

This question aims to identify basic understanding of a multiplexer.

- The multiplexer is a switching device that selects one of a number of inputs and passes this to the single output (1 mark).
- The multiplexer is a combinatorial circuit (1 mark).
- An example Verilog assignment is assign a = (sel == 0) ? x:y; (1 mark for correct use of assign, 1 mark for a syntactically correct assignment).

Issues:
While the first question was answered mostly well, there was some confusion between what a combinatorial and sequential circuit is. The final question explicitly asked for a ***continuous assignment*** that provides the functional operation of a 2:1 multiplexer. This was often confused and there were attempts to provide answers using procedural blocks.

Q16.

A basic understanding of the D-type flip-flop was required here.

- The D-type flip-flop is an edge triggered device that holds 1 bit value, i.e. it is a memory element (1 mark).
- A discussion of the inputs should mention the single bit input, the clock and the use of the clock enable (although not explicit in all D-tpes) to control when the data is loaded (1 mark).

Issues:

- The role of clock enable input was often missed from the discussion.
- It is not n-bit, it is a 1 bit value memory element.
- D-type flip flop is a volatile memory element, not, a non-volatile memory element, it needs power to retain its state.

Q17.

This question required the modification of a Verilog module in order to incorporate an additional input signal that controls on which rising edges of the clock the flip-flop is updated.

- 1 mark for defining the additional input in the header.
- 1 mark for the checking of CE.
- 1 mark for correct operation.

Issues:

- Most people got it right.
- Don't forget the need for begin-end when you have more than one commands, e.g. in the if-else clause.

Q18.

This questions tested the Verilog implementation of a 16-bit register. It explicitely asked for clock enable input that is active-low and an active-low reset signal that acts asynchronously.

- correctly defined header – 1 mark.
- correct sensitivity list in the always block – 2 marks (1 for clock and 1 for reset).
- 1 mark for correct reset action (active low).
- 1 mark for correct clock enable action (active low).
- 1 mark for a fully functioning, syntactically correct design.

Issues:

- Active low was often missed both for the reset action and the clock enable action.
- Use of only one always block was needed.

Q19.

A brief discussion of the role of the PC, IR and control unit was required, i.e.

i. The PC is the program counter and this holds the address of the next instruction to be executed – 1 mark
ii. The IR is the instruction registers that hold the current instruction being executed – 1 mark.
iii. the role of the control unit is to control the operation of/configure control signals to the datapath – ½ mark – and contains the finite state machine – ½ mark.

The question was generally answered well, with an average mark of 2.3/3.

General issues:

- The control also sequences actions using the FSM – this needs to be made clear.
- Lacking detail in answers.
- IR contains the instruction, not the address!
- The IR holds the instruction being executed, which is the current instruction.
- The control does more than just interpreting flags. In general, I expected more detail as to what the control does – it does a lot more than just instruction decoding for example.
- IR does the decoding of the instruction … it does not. Neither does it control the ALU directly.
- The control block contains the clock – it uses a clock yes, but the clock is an external signal to the control.

Q20.

A brief explanation of what happens in the fetch and execute phases in MU0 are required:

Fetch:

- fetch the next instruction form the address specified in the PC and place in the IR
- increment the address in the PC

Execute:

- decode the instruction
- fetch the data from memory specified by the address in the instruction
- perform the operation
- write the result back (either to a register or memory)

Each point was worth ½ mark.

Generally answered well. Average mark 1.94/3.

Issues:

- Not recognizing that the PC is updated in fetch to point to the next instruction. This does not happen in the execute phase.
- A considerable misunderstanding of how much the PC is updated by. Each instruction in MU0 occupies a single memory address, so the PC is updated by 1. It is not ARM, so is not updated by 4 for each instruction (and other various values specified).
- The instruction decode happens in the execute phase and not the fetch phase.
- A large number of answers failed to either recognize where operands come from (or the fact that operands need to be fetched) or where the result is written to.
- A number of answers said that in execute the control sets up the datapath – this is true, but is not what is being asked.

Q21.

Key here is that each instruction will take two clock cycles to complete (fetch and execute). As a clock cycle is 1/20M = 50 ns, then each instruction will take 100 ns to complete – 1 mark for the correct time.

The average mark was quite low for this question – 0.2/1.

Issues:

- Failure to recognise that an instruction takes two clock cycles; lots of answers gave 50ns as the answer.
- Lots of general discussions about how long it takes for results to be produced etc … it doesn't matter, the critical path sets the minimum clock period. The instruction is executed after the 2nd clock pulse.

Q22.

The answer operation can be specified in 3 lines of code:

```
LDA &0A0
ADD &0A1
STA &0A2
```

Generally marks were awarded for the three instruction solution (I didn't penalise anyone for including a STP, even though it was unnecessary), correct MU0 syntax etc.

If the first instruction is at &010, then the PC will point to address &013 after the execution of the STA (or &014 if a STP was included).

On the whole answered well, with an average mark of 3/4.

Issues:

- If defining variable names for the address, using DEFW instead of EQU. This will not work as expected. Remember, in the MU0 instruction an address is specified where an operand can be found, if you use DEFW then the operand becomes the address 0A0, 0A1, or 0A2.
- Having multiple ADDs, or failing to load the accumulator. A number of answers simply ended up calculating 2x the content of an address (or 2x the address!) and storing that result.
- Misunderstandings about how much the PC is incremented resulted in all number of PC values after the instructions have been executed. In some cases, the address of 010 was treated as a binary value (the & specifies the value of hex – this was a deliberate trick!).

Q23.

A couple of general statements were required about delays, the origin of the delay (with respect to the carry propagation from bit to bit) and that the delay is data dependent. I awarded two points for two sensible reasons – 1 mark if only one point is made. I insisted that data dependency is recognised!

This was answered poorly, with an average mark of 0.8/2.

Issues:

- The questions asks about ripple carry adders in general – not the adder in MU0, so answers such as it doesn't produce a carry are not correct in the context of the question.
- I don't ask for a solution – so discussions about look-ahead carry adders received no marks.
- Not recognising that the delay is data dependent.
- Not being clear about where the delay comes from or that it's between bits.

Q24.

The idea is to understand the requirements of the preconditioner for the implementation of the four arithmetic functions of the ALU.

For the four operations the preconditioner is configured as follows:

- ADD, we are doing a simple addition, so output_X = input_X and output_Y = input_Y.
- SUB, since Cin is taken high, this is doing the +1 on the two's complement conversion of the input_Y, so output_X = input_X and output_Y = inverse(input_Y).
- inc, Y is not used so output_Y = 0 and output_X = input_X – the +1 comes from the Cin going high.
- passthrough, input_X is not used so output_X = 0 and output_Y = input_Y.

This needs coding into a Verilog always block. An example solution would be:

```
always @ (*)
  case(M)
    2’b00: begin
             output_X = input_X;
             output_Y = input_Y;
           end
    2’b01: begin
             output_X = input_X;
             output_Y = ~input_X;
           end
    2’b10: begin
             output_X = input_X;
             output_Y = 16’h0000;
           end
    2’b11: begin
             output_X = 16’h0000;
             output_Y = input_Y;
           end
    default: begin
               output_X = 16’hXXXX;
               output_Y = 16’hXXXX;
             end
  endcase
```

Generally marks are awarded as follows:

- use of always block – 1 mark (1/2 mark for missing @ or insufficient sensitivity list)
- use of case – 1 mark (well suited to case rather than nested if else statements) – drops to ½ if there is no default
- for each value of M the output conditions for output_X and output_Y are set correct – ½ mark for each output for each value of M
- correct Verilog syntax – 1 mark – use of begin and ends, correct use of =, endcase present (1 error than ½ mark, 2 or more lose the mark).

Some variation in the marking scheme for required for some answers due to the way the questioned were answered.

There was a mistake in the header code provided in the question, as cin should not be an input to the preconditioner (this is evident from the block diagram in Figure Q24.1) – I did not penalise anyone who introduced a dependency on cin – unless they used it to generate output_X and output_Y directly.

I felt this question was handled well overall. The average mark was 4.8/7.

General mistakes:

- Addition should not be performed in the module – all is required is the configuration of the outputs for the ALU.

- There were issues with selecting appropriate inversion symbols, so I accepted anything sensible.
- For a case statement you should ideally use a default for simulaton purposes – a large proportion of answers didn’t. In this case, the outputs should be set to X in a default, in some cases they were set to values (such as output_X = input_X), however, I did not penalize anyone who has a default, but failed to assign the outputs to X.
- On a minor note, the question asks for the missing code to be produced, NOT the whole module!
- Plenty of missing endcase statements.
- Not including an always statement, or using an invalid sensitivity list.
- There is no need to specify individual bits
- A number of answers provided if … else or assign statement. These are fine, but a case is better since the default can be used to test all conditions. In the case of continuous assignments (assign), the one thing you should be careful about is that the outputs are defined as reg in the header, so at compilation you will get an error!