UG Exam Performance Feedback
Second Year
2016/2017 Semester 1

COMP22111 Processor Microarchitecture

Paul Nutter
Jim Garside

Comments Please see the attached report.

COMP22111 Exam Feedback 2016/17

The average mark for the exam paper was a bit disappointing at 54%. On the whole students did well on two of the questions, then performed badly on the third. In addition, there were a number of answer sheets where it was clear the student has attended few (if any!) of the lectures (or the labs for that matter!).

Q1.

A popular question. The average mark was quite low though at 47%.

The main aim of the question was to test your understanding of the Verilog language and the creation/use of test benches. Part a) and b) are largely bookwork, but part c) tested your ability to write a Verilog testbench choosing sensible test date.

a)

i) On the whole answered fine. The main difference I was expecting to be recognised is that a task can change any number of output variables, whereas a function could only update a single variable. Wrong answers were usually stating them the wrong way round, or not being clear on the difference.

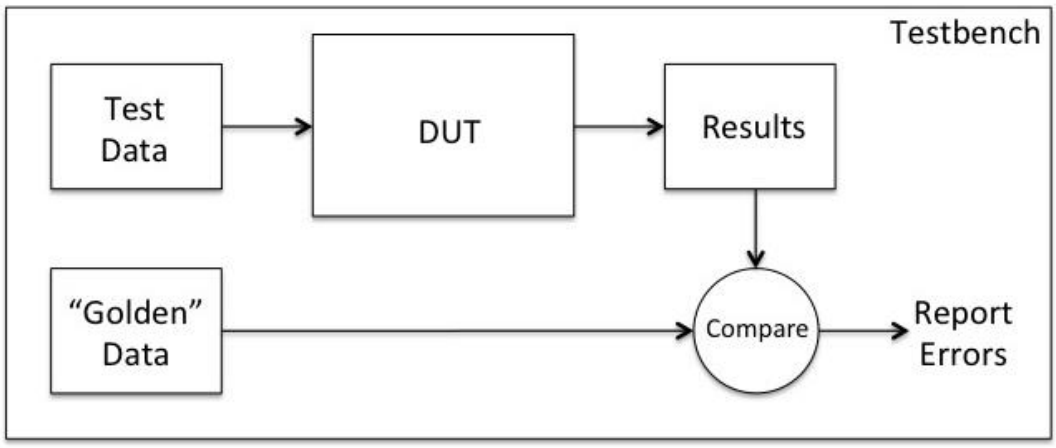
ii) In a number of cases the answers described in what situation you would use a task or a function, i.e. the different situations where you would use either of the two – this is not what was being asked. I was simply after a discussion that covered the use of tasks/functions where code is reused, or to make code more readable.

iii) The main issue with the answers for this question was clarity. Answers would say "at the end of the code" but then not mention that it must be within the body of the module. Very few answers identified that they must be separate from any always blocks.

iv) Mixed answers to this question, which surprised me. If `condition` is set to `X` then the function will not do anything – there is no option in the `case` list – it certainly won't set `Testbranch` to `X`. Almost everyone identified that to get a desirable response you should add a `default` case.

b)

On the whole answered well. I was after a block diagram that showed the overall view of a test bench, i.e.



The main mistake was not showing the compare branch that compares the output from the test with the expected ("Golden") data.

c)

I was very disappointed with the response to this question. The design of the shift register is relatively straightforward and the question hints that exhaustive testing is not necessary ("You may assume that the components within the design of the shift register have been tested exhaustively and operate to the required specification.")

Marks were awarded as follows:

Instantiation of the module under test – 2 marks
Creating a clock – 1 mark
Commenting – 2 marks
Sensible test data – 2 marks
Reset test – 1 mark
sel test – 1 mark
Use of $stop – 1 mark

I didn't mark code down for issues with syntax. There was no need to open files and write results to a file, so some students over-complicated their answers and failed to produce the basics, which was a sensible test strategy.

Taking each of these in turn:

Instantiation of the module under test

Largely fine. Although quite a few answered didn't bother.
Creating a clock

Hugely disappointing – you have come across examples in both COMP12111 and COMP22111. The number of examples where the clock was generated using delays in an initial block was astounding!

In some cases an always block for the clock

```
always #50 clock = ~clock;
```

was given, but clock was never initialised to a value, so this would result in clock constantly being X.

Commenting

Yes that old nugget! I expect test code to be commented. What are you doing and why? Why are you testing a particular test value, what would you expect to happen? There's no excuses for not commenting code properly!

Sensible test data

There's no need to test exhaustively. Choose some sensible test data. We are testing a shifter so 1010 and 0101 are good examples. Why? As are 0000 and 1111.

There's no need to create a for loop incrementing through all the available test values – this is too much.

Reset test

Always at the start! Initialise all variables, with reset = 0. Take reset high for a short while, then back to 0, then test everything else. The reset test does NOT go at the end, or for ever data value tested.

sel test

For each test vector cycle through sel_test to make sure it works as expected. Does it perform the correct operation – is data loaded and shifted appropriately?

Use of $stop

Necessary to pause the simulation … you'll be amazed how many answers didn't include it!

An example test is given on the next page.

```
// instantiate module under test

shift_register shift_4bit (.D(D_data),
                           .Q(Q_data),
                           .sel(op_sel),
                           .clk(clock),
                           .rst(reset));

// generate a clock, period of #100

initial clock = 0;

always #50 clock = ~clock;

// apply reset and test data

initial
begin
  reset = 0;
  sel = 0;
  D_data = 4'b1111;  // start with 1111 for reset test
  #100
  reset = 1;         // test reset action
  #100
  reset = 0;
  #75               // expected output 0000
  sel = 1;          // expected shifted value should 0000
  #50
  sel = 0;
  #50
  D_data = 4'b 1111;
  #50               // expected output 1111
  sel = 1;          // expected shifted value should 1111
  #50
  sel = 0;
  #50
  D_data = 4'b 0101;
  #50               // expected output 0101
  sel = 1;          // expected shifted value should 1010
  #50
  sel = 0;
  #50
  D_data = 4'b 1010;
  #50              // expected output 1010
  sel = 1;         // expected shifted value should 0101
  #50
  sel = 0;
  #50
```

```
  $stop;           // pause the simulation
end
```

Q2.

The most popular question, with a high average mark of 69%. Not surprising considering because of the closeness to the work you have undertaken in the lab.

The main aim of the question was to test your understanding of the Stump processor and its operation. Questions a) to c) are largely bookwork. Questions d) and e) test your knowledge and understanding (although you should have completed a similar exercise in the lab!).

a)

Easy question – although you'd be surprised how many answers did not answer the first part of the question. Why is it hard wired to zero? – simplest answer, to enable further operations to be implement that are not part of the ISA!

Some good examples given, including

MOV – ADD R3, R2, R0
CMP – SUBS R0, R3, R4 (remember the 'S'!)
NOP – ADD R0, R0, R0

b)

I was looking for the following to answer this question:

- data held in register is only operated on, not the contents of memory
- results written back to a register
- ld and st instructions provided to enable data to be read from/written to memory

On the whole answered well, although very few highlighted that the result is always written back to a register.

c)

On the whole this was answered well. Almost everyone who answered managed to discuss the differences between the source of the operands for Type 1 and Type 2 instructions. The only issue was that very few answers mentioned how (for example) the registers a referenced in the instruction, or that the literal value is supplied within the instruction.

d)

This question tested your understanding of some Stump code.

i) On the whole answered well. Most realised that the code was simply initialising variables.

ii) Most recognised that the code is performing a multiplication of two numbers. However, I was supervised how many choose the wrong branch condition, BGE being a popular one (this will loop one extra time when the counter is 0). A correct choice is BNE. I was a bit hard on answers that didn't clearly state that the result is written to a memory location!

iii) The offset from the current instruction is -2. However, you must remember that the PC has been updated (+1) so the correct offset is -3. A number of answers put 3 as the offset – wrong direction!

e)

I was really impressed with how many students got good marks for this question (although thinking about it, you will have done a similar exercise in the lab).

Common mistakes:

- Setting register enable signals to X
- Not getting the srcA and dest registers correct for Fetch (giving X in a lot of cases!)
- Getting memory signals wrong, or setting to X
- Not realising that most signals are X for the memory state

The marking scheme looked at the following:

- Correct status signals for register enables
- Correct signals for MUXs
- Stump register bank configured correctly
- ALU/shifter configured correctly
- Correct memory signals

In the cases where a signal could be don't care, X, I was looking for an X to be given since I wanted you to explicitly recognise the condition.

Q3.

Question 3 was quite a popular choice; the overall marks spread was quite wide although the average was disappointingly low. There were some systematic problems: for example part (a) revealed fairly widespread confusion over the difference between signed and unsigned -interpretation- of a value; answers to part (d) rarely mentioned the use of the 'carry' flag as an extension bit in shift and rotate operations - despite this being visible in the labs. as well as common in many processor ISAs. The intentionally more challenging aspects of the question - parts (e-g) on SIMD operation - were not all made explicit in the lectures but could be deduced from the information given. There seemed to be a reluctance to do this rather than ignorance of the topic.

Q4.

Question 4 was less popular but, in general, better answered. The spread of marks was less than for some other questions. It required more willingness to offer opinions on a design rather than repeat facts - something which is often less popular in exam. questions although justifiable choices, even if 'suboptimal' can earn significant marks. The most significant challenge was in part (e), looking at the possible codings for a 'BL' operation. Perhaps the majority of candidates perceived a difficulty with the range of offsets but no one made the (challenging) intuitive leap of -distributing- the target addresses more widely, i.e. by shifting the offset/address bits left to cover the whole address space at a coarser resolution. A number of candidates used bits which could have helped addressing for (arguably) less important purposes such as -specifying- a particular link register. (In such cases credit was still awarded according to the justification/explanation.)