Two hours

UNIVERSITY OF MANCHESTER
DEPARTMENT OF COMPUTER SCIENCE

Implementing System-on-Chip Designs

Date: Tuesday 14th January 2020

Time: 09:45 - 11:45

Please answer all THREE Questions
Each question is worth 20 marks

This is a CLOSED book examination

The use of electronic calculators is NOT permitted

1. a) In SoC design, what is a "netlist"? (2 marks)

   b) Explain why pipelining functions will usually *increase* their *latency*. (2 marks)

   c) Give two distinct reasons why VLSI designers may be concerned with the power used by an ASIC. (2 marks)

   d) The 'C' in CMOS stands for "complementary"; what does this mean in terms of transistors in a CMOS gate and why is it beneficial in the logic operation? (2 marks)

   e) Source-level *test coverage* can be derived from simulation runs. Give one aspect of verification of an implementation that such a tool can help with and one aspect which would *not* be detected at this level. (2 marks)

   f) Here is a badly written fragment of Verilog:

```
always @ (posedge clk) ccc <= bbb;
always @ (posedge clk) bbb  = aaa;
```

   If `aaa` is 1, `bbb` is 2 and `ccc` is 3 at one moment, what is the value of `ccc` after the next rising edge of `clk`?
   Explain your reasoning. (2 marks)

   g) Figure 1 shows a CMOS 'complex' gate. Sketch a schematic showing an arrangement of MOS transistors which will implement this function.
   Ensure that the 'type' (i.e. pMOS or nMOS) of each transistor is clear. (2 marks)

Figure 1: CMOS complex gate: 'or2andi'

   h) You are designing a digital filter for a system-on-chip; this requires many successive multiply-accumulate operations on 16-bit operands at high speed. What considerations go into your choice of an appropriate microarchitecture for the multiplier? Suggest an appropriate choice of multiplier design. (2 marks)

   i) Why might part of a system-on-chip might be *power gated*? Illustrate, with a figure or otherwise, how this might be implemented. (2 marks)

   j) What is a 'scan chain' in an SoC? What can it contribute to a design and why is it a convenient model? (2 marks)

2. a) Describe, with truth tables or otherwise, the difference(s) between the Verilog comparison operations "==" and "===". (2 marks)

   b) What is the advantage of the "===" comparison for *verification*? (1 mark)

   c) When testing a known peripheral module on a bus a *handshake* sequence may be employed, as shown in the timing diagram (fig. 2).

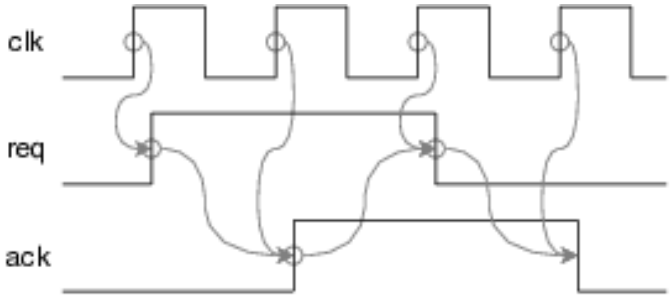

Figure 2: Timing diagram

Figures 3 & 4 show two ways verification code could drive the '`req`' signal for this module.

```
@ (posedge clk) req <= #1 1'b1;
@ (posedge clk);
@ (posedge clk) req <= #1 1'b0;
```

Figure 3: First Verilog example

```
while (ack !== 1'b0) @ (posedge clk);
req <= #1 1'b1;
while (ack === 1'b0) @ (posedge clk);
req <= #1 1'b0;
```

Figure 4: Second Verilog example

Would both of these code fragments generate the same input as shown in fig. 2? State any assumptions you make. (1 mark)

   d) Which of these verification code fragments (figs. 3 & 4) provides a more flexible test and why? (2 marks)

   e) Suggest a way to make your favoured example an even *better verification* test. (2 marks)

   f) In the Verilog examples, what (if anything) do the "`#1`" delays contribute to the test process?
   Give any reasons you can think of. (2 marks)

g) The handshake shown in figure 2 provides an interface between units with the *same clock signal*. What additional problem(s) might you expect if the `req` and `ack` signals were generated from units using *different* clock signals?
How might you alleviate or eliminate such problem(s)? (4 marks)

h) If the communicating units in the previous part of this question are also powered using *different supply voltages* what other problem(s) might be encountered?
How might you alleviate or eliminate such problem(s)? (4 marks)

i) Why are multiple supply voltage domains sometimes used on an SoC? (2 marks)

3. a) Figure 5 shows a buffered multiplexer which you have to implement. The labelled buffers are guaranteed large enough to accommodate any expected data, which arrive in *bursts* of several kilobytes interspersed with idle gaps. The two sources are *independent* so data may arrive from each simultaneously.

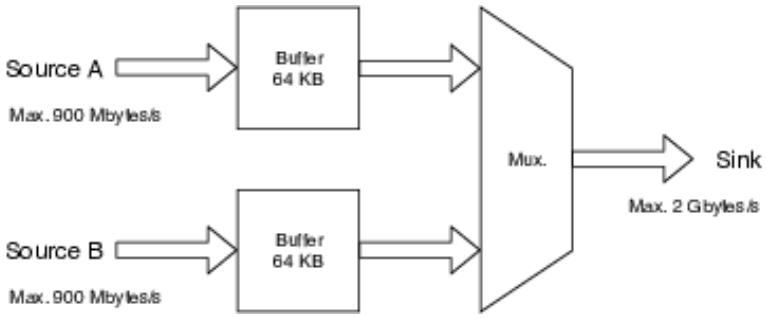

Figure 5: Buffer architecture

Why would the buffers be built from SRAM rather than D-type flip flops?

(2 marks)

b) It is possible to build *dual-port* SRAM – where two independent operations can be made *simultaneously* – but the area-per-bit is about greater than that of a single-port SRAM and accesses are slower.

You have to implement the buffers shown in figure 5. Your component library has available macros for:

- A 4 K single-port SRAM with a 32-bit word length and a minimum cycle time of 4 ns.
- A 2 K dual-port SRAM with a 32-bit word length and a minimum cycle time of 5 ns. This can perform two *simultaneous* access cycles.

Each of these macros occupies a very similar silicon area: 5000 μm^2.

To implement the required capacity for *each* buffer shown, what is the area occupied using:

i) single-port SRAM macros

ii) dual-port SRAM macros (2 marks)

You can assume that interconnection area is negligible compared to the macro size.

c) Ignoring capacity in this part of the question, for a single channel, what would be a buffer's peak *throughput* in Mbytes/s achievable using:

i) one single-port SRAM macro

ii) one dual-port SRAM macro (6 marks)

d) The required buffer bandwidth *throughput* is a 900 Mbytes/s (per channel). Suggest how you could provide this with the SRAM macros available. (4 marks)

e) It turns out that the area available will restrict the design to using only single-port SRAM macros if the required capacity is to be provided. How might this be done whilst still providing a throughput comparable with that achievable with dual-port SRAMs?
Highlight any characteristics of the problem which constrain your design and any you can exploit in creating your solution. (6 marks)

END OF EXAMINATION