Part II CST: SoC D/M: Quick exercises S3-S4 (examples sheet) Feb 2017 (rev a).

This sheet contains short exercises for quick revision. Please also look at past exam questions and/or try some of the longer exercises from the main sheet.

Unless otherwise stated, both here and in exam questions, you can answer using any mixture of RTL, SystemC or block or circuit diagrams. Syntax details are not important. Similarly, precise accuracy with C, C++ or any particular assembly language is not important.

SP3Q: Design Partition

SP3Q.1. Why would we typically put some circuitry on a small GaAs chip instead of our main SoC?

SP3Q.2. Why is it important for the designer (aka architect) to be able to quickly perform 'what-if' design partition experiments and how much accuracy do they need to deliver? (Perhaps defer this to the ESL slide pack).

SP3Q.3. What makes it a good idea to put CRC computation and error-correcting code computation into custom hardware?

SP3Q.4. How come my mobile phone can compress video from its camera while running on battery whereas when I do it on my laptop the fan comes on and the battery goes flat quickly?

SP3Q.5. When should I connect a CRC accelerator as a co-processor and when should it be out on the system bus somewhere?

SP3Q.6. When would I use shared memory and when would I use a FIFO to pass data between two custom cores on my SoC? If the receiving processor is notified by interrupt, what circuit might generate the interrupt in each case?

SP3Q.7. What is the difference in design style and fabrication process between a DRAM chip and a standard cell ASIC?

SP3Q.8. Why is logic synthesis to standard cell so much easier than full-custom design?

SP3Q.9. Why does an FPGA have more than one order of magnitude less logic density than an ASIC made from the same geometry silicon?

SP3Q.10. What functional blocks were 'hardened' in second generation FPGA's compared with the very early ones that only had flip-flops, LUTs and I/O blocks? (By hardended we mean implemented as a connectable, mask-programmed resource rather having to be implemented out of programmable logic). 1

SP3Q.11. Why have FPGAs increasingly hardened more and more IP blocks? Give three mainstream example block types.

SP3Q.12. State when or whether each of these would be a major cost in making a chip: engineering manpower, masks, silicon, yield-failures, IP-licensing, other... ?

SP4Q: RTL

RTLQ1. Using leaf cells of your own design, give an RTL structural netlist for an RS latch defined by:

```
module rslatch(input set, input clr, output q, output qb);
    wire qb;
    wire q  = !(set || qb);
    assign qb = !(clear || q);
endmodule
```

RTLQ2. Give a brief definition of RTL and Synthesisable RTL. Name two example languages. [4 Marks]

RTLQ3. Explain Verilog's blocking and non-blocking assignment statements. Show how to exchange the contents of two registers using non-blocking assignment. Show the same using blocking assignment. [6 Marks]

RTLQ4. Convert the following behavioural RTL into an unordered list of (pure) RTL non-blocking assignments: [5 Marks]

```
always @(posedge clk) begin
   foo = bar + 22;
   if (foo > 17) foo = 17;
   foo_final = foo;
   foo = 0;
   end
```

RTLQ5. Explain the terms 'structural hazard' and 'non-fully pipelined'. [4 Marks]

RTLQ6. Give a fragment of RTL that implements a counter that wraps after seven clock ticks. [3 Marks]

RTLQ7. Give a fragment of RTL that uses two multiply operators but where only one multiplier is needed in the generated hardware. Sketch the output circuit generated by a simplistic logic synthesiser and the optimised circuit that only uses one multiplier. [4 Marks]

RTLQ8. Give an RTL design for a component that accepts a five-bit input, a clock and a reset and gives a single-bit output that holds when the running sum of the five-bit input exceeds 511. [6 Marks]

RTLQ9. Show an example piece of synchronous RTL before and after inserting an additional pipeline stage. [4 Marks]

RTLQ10. Convert the following behavioural RTL into an unordered list of (pure) RTL non-blocking assignments: [8 Marks]

```
always begin
   @(posedge clk) foo = 2;
   @(posedge clk) foo = 3;
   @(posedge clk) foo = 4;
   end
```

RTLQ11. Give a fragment of RTL that uses non-blocking assignments (in Verilog the `<=` operator) to swap a pair of bytewide registers on a clock edge when an externally-provided enable signal holds. Sketch the resultant circuit (or gate-level, structural netlist) that would arise when your fragment is compiled to gates.

RTLQ12. The following fragment of RTL produces a repeating pattern on a one-bit output. Assuming it is accepted by some gate-generating synthesis tool, give the resultant circuit (or structural netlist) that it might sensibly generate.

```
     always begin
          @(posedge clk) a<= 1;
          @(posedge clk) a<= 0;
          @(posedge clk) a<= 0;
          end
```

SG4s – Simulation:

SG4s.1. Tabulate the events inserted in an event list when simulating one clock cycle of your net-level hardware design from question RTLQ11.. Assume each gate has a delay of 100 picosceonds.

SG4s.2. Modify your net-level hardware design from RTLQ11. by adding an inverter so that one of the bits is swapped in polarity when moving between one register to the other. What extra events would arise in the net-level simulation. What has happened to the maximum clock frequency of the design?

SG4h – Hazards:

SG4h.1. Consider a FIFO component that accepts 16-bit writes and provides 8-bit reads. Internally it is implemented with a 128-byte SRAM with organisation 128x8. A revised implementation uses an SRAM with organisation 64x16. Which implementation offers better throughput and which (if any) suffers from structural a hazard ?

SG4h.2. The writeback pipeline stage in a simple RISC processor was once described as 'a dummy stage whose only purpose is to avoid a structural hazard'. Is this a fair comment and why? How does data cache cycle time affect your answer?

SG4r – Folding, Retiming & Recoding:

SG4r.1. The critical path in a clock domain consists of three AND2 gates each with delay 35 ps and starts and ends at flip-flops with the following parameters: set-up time 4 ps, hold time 2ps, clock to Q time 15 ps. What is the maximum clock frequency ?

SG4r.2. Time and space can be exchanged by design refactoring. Compare the long multiplication algorithm in the notes with a naive repeated addition approach. Which has more time and which uses more space (if either)?

SG4r.3. Booth's radix-4 algorithm for long multiplication is embodied in the following ML function:

```
fun booth(x, y, c, carry) = // Inputs are x and y. Other args initially clear.
    if(x=0 andalso carry=0) then c else
let val x' = x div 4
    val y' = y * 4
    val n  = (x mod 4) + carry
    val (carry', c') = case (n) of
      (0) => (0, c)
     |(1) => (0, c+y)
     |(2) => (0, c+2*y)
     |(3) => (1, c-y)
     |(4) => (1, c)
    in booth(x', y', c', carry')
    end
```

How many adders does it require in its natural hardware implementation (no special fold/unfold applied)? How many clock cycles does it use ?

Long: Sketch the hardware datapath and list the inputs and outputs to the controlling FSM. (*Perhaps this bit should be on the long exercise sheet.*)