SoC D/M Exercise Sheet, 2014/2015

This sheet contains exercises of various lengths. Many exercises are nominally allocated marks at Tripos examination level (i.e. with 20 marks making a full exam question).

There is some repetition of material between the exercises, so a suitable target is to solve approximately one third of them. Exercises marked with a ♡ form a recommended core. Some sections contain additional preference instructions.

Example answers are available to supervisors.

SP4 (RTL, Simulation, Hazards, Folding) Short Exercises

RTLQ1. Give a brief definition of RTL and Synthesisable RTL. Name two example languages. [4 Marks]

RTLQ2. 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]

RTLQ3. ♡ 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
```

RTLQ4. Explain the terms 'structural hazard' and 'non-fully pipelined'. [4 Marks]

RTLQ5. Give a fragment of RTL that implements a counter that wraps after seven clock ticks. [3 Marks]

RTLQ6. ♡ 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 Verilog compiler and the optimised circuit that only uses one multiplier. [4 Marks]

RTLQ7. 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]

RTLQ8. Show an example piece of synchronous RTL before and after inserting an additional pipeline stage. [4 Marks]

RTLQ9. Convert the following behavioural RTL into an unordered list of (pure) RTL non-blocking assignments: [8 Marks]

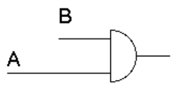
```
always begin
   @(posedge clk) foo = 2;
   @(posedge clk) foo = 3;
   @(posedge clk) foo = 4;
   end
```

0.1 SG1 (RTL, Simulation, Hazards, Folding) Longer Exercises

RTL1. Synthesisable RTL standards require that a variable is updated by at most one thread: give an example of a variable being updated in two `always` blocks and an equivalent combined always block or circuit that is valid/correct. [8 Marks]

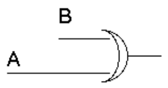
RTL2. Give a schematic (circuit) diagram for the design of RTLQ7. Use adders and/or ALU blocks rather than giving full circuits for an such components. [7 Marks]

RTL3. Complete the truth tables for the 2-input AND gate (using symmetry) and the other three functions shown in this grid where the inputs range over { 0, 1, X, Z }. (This was not lectured in 2012. X denotes don't know and Z denotes high-impedance.)



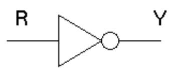
AND gate

		B			
		0	1	X	Z
A	0	0	0	0	0
	1	0	1	X	X
	X	0	X		
	Z	0			



OR gate

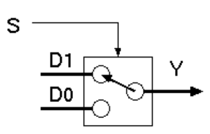
		B			
		0	1	X	Z
A	0				
	1				
	X				
	Z				



Inverter

R	0	1	X	Z
Y				

Two-input mux



S	D0	D1	Y
0			
1			
X			
Z			

RTL4. Draw the gate-level circuit of an RS-latch using a pair of cross-coupled NOR gates described by the following pair of continuous assignments. Describe the complete sequence of events that occur when the the latch is initially clear and is set with a positive pulse of 5 nanoseconds between times 10 and 15. Assume the gate delay is 200 picoseconds. Your answer will essentially be a list of net/time/value triples but you also need to say when these are added and removed from the event queue. [8 Marks].

```
assign q = !(clear || qb);
assign qb = !(set || q);
```

RTL5. ♡ Identify the structural hazards in the following fragment of Verilog. What holding registers are simplistically needed if the main array (called daza) is a single-ported RAM? What if it is dual ported ? Can all the hazards be removed be recoding the algorithm ?

```
module FIBON(clk, reset);
   input clk, reset;
   reg [15:0] daza [32767:0];
   integer pos;
   reg [3:0] state;
   always @(posedge clk) begin
      if (reset) begin
         state <= 0;
         pos <= 2;
         end
      else case (state)
             0: begin
                daza[0] <= 1;
                daza[1] = daza[0];
                state <= 1;
             end
             1: begin
                daza[pos] <= daza[pos-1]+daza[pos-2];
                if (pos == 32767) state <= 2; else pos <= pos + 1;
             end
           endcase // case (state)
      end
endmodule
```

RTL6. Summarise the main differences between synthesisable RTL and general multi-threaded software in terms of programming style and paradigms. [10 Marks].

RTL7. In Verilog-like RTL, write out the complete design, including sequencer, for the datapath and controlling sequencer for the Booth long multiplier following the style of the long multiplier given in the lecture notes. (later we will do it in SystemC RTL-style and SystemC TLM-style).

```
// Call this function with c=0 and carry=0 to multiply x by y.
fun booth(x, y, c, carry) =
    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
```

It should start as follows:

```
module LONGMULT8b8(clk, reset, C, Ready, A, B, Start);
   input clk, reset, Start;
   output Ready;
   input [7:0] A, B;
   output [15:0] C;
```

(Details of language syntax are unimportant) [15 Marks]

RTL8. Optional exercise: (no further examinable ground covered): Repeat the previous exercise for for long division (using either the shift-left till greater then shift right method or using Goldschmidt's method (repeatedly multiply n and d by $1 - 2d$)). [20 Marks]

RTL9. Modify (fold in space) the following RTL code so that it uses half-as-many ALUs and twice-as-many clock cycles to achieve the same functionality as the following component:

```
module TOFOLD(clk, reset, start, pp, qq, gg, yy, ready);
   input clk, reset, start;
   input [7:0] pp, qq, gg;
   output reg [7:0] yy;
   output reg ready;
   integer state;
   always @(posedge clk) begin
      if (reset) begin
         state <= 0;
         ready <= 0;
         end
      else case (state)
            0: if (start) state <= 1;
            1: begin
               yy <= (pp*gg +  qq*(255-gg)) / 256;
               ready <= 1;
               state <= 2;
            end

            2: if (!start) begin
               ready <= 0;
               state <= 0;
               end
          endcase // case (state)
      end
endmodule
```

[10 Marks]

END OF DOCUMENT.