CS226 Practice Problem Set 1 (Spring 2016)

Date posted: Jan 18, 2016 **Expected Solving Time: 1 hour**

- *If you need to make any assumptions, state them clearly.*
- ***These are ungraded practice questions. You are strongly encouraged to solve these on your own to ensure you understand the material being taught in class.***
- ***Mutual discussion is allowed, but copying is not. Please read the guidelines on the course webpage if you don't understand the distinction between the two.***

1. Consider the datapath and controller circuits for integer division, as studied in class. We saw that in order to control the sequencing of operations in the integer division algorithm, it suffices to control when values are loaded in different registers. We then looked at two different ways of achieving this:

   - Controlling the clock inputs of registers, by AND-ing the global clock signal with a controller generated register-specific "clock enable" signal.
   - Feeding the free running clock to each register, and controlling the data input of each register in each clock cycle, using multiplexors.

   For really high-speed clocks, messing with the clock inputs of registers is undesirable, but for low-speed clocks, this is a reasonable option.

   For purposes of this question, assume we are using a low-speed clock; so the first option above is acceptable. Note that even with this option, we may need to have multiplexors at the inputs of registers, because we may want to load a register from different sources at different times. For example, register "a" in the integer division datapath needs to be loaded from two different sources: user-provided input, and the output of the subtractor.

   Fig. 1 shows a schematic of the integer division datapath assuming clocks of registers are enabled (or "gated") by the controller. However, we do need multiplexors at the inputs of some registers. Note that we no longer need 3-to-1 multiplexors (that switch one of three inputs to the output, depending on the condition). Instead, 2-to-1 multiplexors suffice in this case.

   (a) Design a truth table for the controller in this problem, adding as few (hidden) state variables, $sv_1, sv_2, \ldots$, as you can, to implement the desired sequencing. Your sequencing must now enable the clocks of appropriate registers, and also provide the right inputs to these registers in case there are multiplexors at the inputs of the registers.

   You may make the following assumptions to solve this problem:

   - The time taken by any datapath block to compute its outputs, once its inputs have been provided, is negligible compared to the clock period.
   - a, b, t, q and r are 128 bits wide.
   - After the Load signal goes high (to load new values of a and b in the respective registers), it goes low only after two clock cycles, and subsequently it does not go high again for at least 1000 clock cycles. In other words, loads are infrequent compared to the delay involved in computing the quotient and remainder from a given pair of integers.
   - The user provided (external) values of a and b do not change when the Load signal is high.

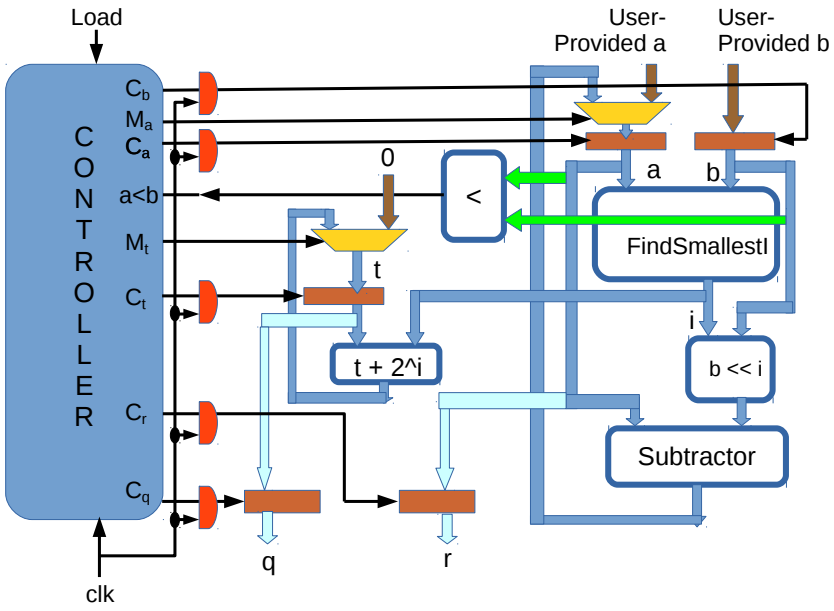

Figure 1: Integer division revisited

To help you get going, a template for the desired truth table is provided below. The input and output signals in this table refer to those shown in Fig. 1.

Controller Inputs				Controller Outputs								
$a < b$	Load	sv_1	$sv_2 \ldots$	C_a	M_a	C_b	C_t	M_t	C_q	C_r	Next sv_1	Next $sv_2 \ldots$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(b) Now assume that the FindSmallestI block takes upto D time units to compute its result. In other words, after fresh values of a and b are provided to the inputs of FindSmallestI, it takes upto D time units for the FindSmallestI block to produce the correct i on its output, such that $b.2^i \leq a < b.2^{i+1}$. Using the value of i from the output of the FindSmallestI block before this delay has elapsed may therefore result in erroneous results being computed. Suppose further that $D \leq 4\times$ (clock period). The delays of all other datapath blocks are assumed to be negligible compared to the clock period.

Re-design the controller truth table of sub-question (a) using the same template, but taking into account the above delay of FindSmallestI.

(c) The Load input of the integer divider is fed from another circuit, which has started malfunctioning. Specifically, after Load goes high, it is guaranteed to stay high for at least two cycles, during which the user-provided (external) values of a and b don't change. However, Load may again become high shortly afterwards (due to malfunctioning of the circuit driving Load), i.e. it is no longer guaranteed not to become high for 1000 clock cycles. Assume further that the user-provided (external) values of a and b can change when Load is low, but they do not change when Load is high..

Re-design the controller truth table of sub-question (b) such that the integer divider circuit ignores Load going high and changes happening in the user-provided values of a and b until the computation of the current quotient and remainder is completed.