Hwk #5 – Verilog and Pipelining
Due October 15th, 2009 Beginning of Class
100 points

Design a 16-bit pipelined ripple-carry adder in Verilog. The requirements are listed below.

1. The calculation latency is 4 cycles. This means the output should be ready 4 cycles after a pair of operands and an operation specification are presented at the inputs of the adder. For example, if the inputs arrive at the 1st positive edge of CLK, the corresponding result is available at the 5th positive CLK edge.

2. The adder is pipelined. It accepts a new input each cycle and produces a result each cycle when the pipeline is full. Each input consists of two operands and the associated operation specification, or a RESET.

3. The adder should be built using ripple carry adder stages. A 1-bit adder is provided as a starting point for building the ripple carry stages. We assume the delay of each 1-bit adder is 1/4 cycle, so you'll use 4 1-bit adders in each pipeline stage. As you can see, the idea is that if we're going to use this adder every cycle, we can speed it up using pipelining so that the simple ripple carry circuit is fast enough for our purposes and no fancy carry lookahead is needed.

4. The interface should behave as follows:

INPUT signals: CLK, RESET, operand0[15:0], operand1[15:0], operation[1:0]

OUTPUT signals: result[15:0], carryout, overflow, ready

if RESET is enabled, the ready signal should be zero for all 4 pipeline slots, else:

operation:

00: the adder is disabled (pipeline bubble)

01: result = operand0 + operand1, the ready signal goes high 4 cycles later, at the same time, the result is on the bus, the overflow goes high if overflow happened. All these output should last only one cycle, and then go back zero if there is no other result

10: result = operand0 – operand1, the ready signal goes high 4 cycles later, at the same time, the result is on the bus, the overflow goes high if overflow happened. All these output should last only one cycle, and then go back zero if there is no other result

11: reserved, the adder is disabled.

A picture of the desired circuit is given below. Note that in the figure, no internal control or clock lines are drawn. There may or may not be a centralized control logic block in your design, that part of the diagram is there to remind you that you'll need to worry about control. In this vein, remember that you're pipelining the operations, so you'll need to remember somewhere which operation (if any) is being done in which adders at a given clock cycle.

You are provided with the verilog template file for a 16-bit adder – `rcAdder.v`. A simple test bench is also provided – `tb_rcAdder.v`. You will probably want to run the tools on these files to make sure you can use the tools and that you know how the circuits work as your first step on this problem. Note that before you try to run the tools, you must set up your Linux environment properly. This process is documented on the Verilog

tools page in the handouts. If you have trouble with this part (e.g. you cannot run vcs after you think you've got things properly set up), ask Dong for help.