Computer Structure Lab - Fall 2003

Dr. Guy Even, Amir Ingber and Marko Markov

Handout #6: A simplified DLX: design, testing, timing, and programming

Each question counts as one assignment. Submission deadline is given next to each question. We highly recommend that you divide the load between students in the same group, otherwise you will find the burden too high.

1. (*submit at the beginning of your* 11*th lab meeting.*) Design and test the simplified DLX.

   - Submit schematics of the project and of the new modules. Submit results of testing using test vectors of each module.
   - Submit VHDL file of the control. Describe how you tested the control. Submit results of testing the control using test vectors. (In order to reduce the amount submitted simulation, please submit test vectors of the following paths only: ALU, TESTI, LOAD, STORE, JALR, BTAKEN. However, you should prepare test vectors for all of the paths.)
   - Test your design to see if the correct RTL instruction is executed in each state. Write short DLX programs that test every instruction. Note that some instructions must be tested more than once (e.g. "branch taken" and "branch not taken"). Store these programs in the RAM of the I/O_SIMUL module. Build a project from your DLX design and the modified I/O_SIMUL module. Simulate this project. Submit: (1) Your programs. For every instruction, list the states that are traversed. (2) Waveforms of a simulation of your design. Add remarks to the waveforms that show the traversed states, the RTL instruction that were executed, and what they did.

   After completing these stages successfully, it is likely that your design can be run on the RESA and monitored to see if it functions properly.

2. (*submit at the beginning of your* 12*th lab meeting.*) Implement and test your design on the RESA.

   - Implement your design. You should easily meet the timing requirements (clock rate at least 5 MHz). Submit a report generated by the implementation software stating that your design is fast enough and that implementation was successful.
   - Re-use the test programs you used to test the RTL instructions on the RESA. Use the Logic Analyzer to verify that the RTL instructions are executed properly. Submit printouts of the monitoring with remarks explaining why you got the right results.
   - Read the RESA test program. Describe how it tests the design.

- Run the RESA test procedure on your design. Your design should pass the test! Get Marko's approval of your design.

3. (*submit at the beginning of your* 13*th lab meeting.*) Try to design the fastest design you can. You should at least meet the 10MHZ threshold. Implement your design and meet the timing requirements (clock rate at least 10 MHz). There are a few ways to try to decrease the feasible clock period.

   - The first method is to try to reduce delays due to routing in the FPGA. This can be done by "helping" the place & route tools. You will be told more on how this is done.
   - The second method is to identify the critical path and try to shorten its delay. This method is applicable provided that there are only a few critical paths.
   - The third technique can be used if you failed in shortening the delay of critical paths. Suppose the critical path includes the ALU. A way to solve this critical path is to allow two cycles for the ALU. This requires a change in the control so that each of the corresponding states are divided into two states. We do not recommend using this method (for example, the software will still report timing problems because it does not know that the signal is not sampled after one cycle). You should be able to solve the timing problem without it.

   After you meet the timing requirements, Implement and test your design on the RESA. Note that the timing analysis done by the software tools is often too pessimistic. Your design may be fast enough even if the software reports otherwise.

4. (*submit at the beginning of your* 14*th lab meeting.*)

   The goal in this assignment is to write a DLX assembly program that computes shortest path in a **directed** graph, using $Bellman - Ford$'s algorithm.

   The **input** should be as follows (starting from address $0x00800003$):

   (a) $n \in \{2, 1024\}$ - the number of nodes of the graph. The vertex set is $\{0, \ldots, n-1\}$.

   (b) $m \in \{1, 1024 \cdot 11\}$ - the number of edges of the graph.

   (c) A list of edges and edge weights. For each edge, three numbers are given: i,j,w, which denotes an edge $i \rightarrow j$ of weight w . $w \in \{-1024, 1024\}$

   The **output** should be as follows (starting from address $0x0080C000$):

   - if the graph contains a negative weight cycle, that is reachable from v_0, the output should be $0x80000000$ (minus infinity) in address $0x0080C000$.
   - otherwise: A list of values $d_1, d_2, \cdots, d_{n-1}$, where d_i is the length of the shortest path from vertex 0 to vertex i.

   Submit:

(a) A written explanation of your program. In particular, describe your internal format, how your program runs, the memory organization in your program, and the complexity of your algorithm.

(b) A printout of your annotated DLX assembly program.

(c) Test your program on four graphs:

- A path of 1024 nodes; All edges have unit weight.
- A graph of your choice, **without** a negative-weight cycle reachable from v_0.
- A graph of your choice, **with** a negative-weight cycle reachable from v_0.
- A graph of our choice - will be given to you by Marko.

Submit a printout of the first $0x30$ words in the memory starting from address $0x0080C000$ for each graph. Submit also a description of the graphs you chose.

for example, the first few places in the memory starting from $0x00800000$ as an input to (i):

```
0x00800000: 0x--------
            0x--------
            0x--------
            0x00000400      \* number of nodes = 1024
            0x000003FF      \* number of edges = 1023
            0x00000000      \* edge #1: 0->1 , w=1
            0x00000001
            0x00000001
            0x00000001      \* edge #2: 1->2 , w=-1
            0x00000002
            0xFFFFFFFF
            0x00000003      \* edge #3: 3->2 , w=1
            0x00000002
            0x00000001
            .
            .
            .
```

Remark: you cannot assume any kind of order of the input edges.