University of Pennsylvania
Department of Electrical and Systems Engineering
Electronic Design Automation

ESE535, Spring 2008 Assignment #7 Wednesday, April 22nd

Due: Wednesday, May 13th, noon (12pm).

Resources You are free to use any books, articles, notes, or papers as references. Provide citations in your writeup as appropriate.

Collaboration This is an individual assignment; no collaboration is allowed.

Writeup Writeup should be in an electronically readable format (HTML or PDF preferred—I do not want to decipher handwriting or hand-drawn figures). State any assumptions you need to make.

Points This assignment is graded out of 10 points. The possible points on all 4 problems is 12 points. This gives you a chance to earn extra credit to apply against earlier non-programming (1, 2, 3.2–3, 6B) assignments on which you did not receive a perfect score.

Problems

1. **2 [pts]** *Placement with minimum distance constraints*: Ionizing particle strikes (*e.g.* α-particles) may disrupt logic. One way to guard against errors is to replicate the logic and compare the results (*e.g.* duplicate and compare, triplicate and vote). As long as the ionizing particle upsets a **single** copy of the logic, it is possible to detect (and possibly correct) the error since one or more of the replicas will not be affected by the ionizing particle strike.

   As we shrink feature sizes, it becomes possible for a single particle strike to disrupt multiple logic bits in a localized region of the chip. If we placed the copies of a piece of logic too close together, they might both (all) be disrupted by a single particle strike, defeating the purpose of replication.

   As a result, when using replicated circuitry to mitigate against ionizing particle disruption, it is necessary to guarantee a minimum distance between replicas.

   Provide a 2D placement routine which minimizes squared wire length while guaranteeing a minimum specified Manhattan distance between gates. That is, in addition to the normal edges between graph nodes (E), your input will include an additional set of edges ($E_{min\_distance}$) which specify the minimum distance which must be guaranteed between node pairs. You may assume edges in $E_{min\_distance}$ are all 2-point nets (*i.e.* $e_{md} \in E_{min\_distance} = (src, sink, distance)$).

2. **[2pts]** *Verify FSM Communication*: Consider a pair of communicating Finite-State Machines (FSMs) each controlling an associated datapath. Let's call them A and B. There is a single channel in each direction between them (AtoB and BtoA). State machine A has the control signals SendAtoB and RcvBtoA. State machine B has the control signals SendBtoA and RcvAtoB. Both state machines have a single designated start state and share a common reset signal that returns them to the start state; the two machines operate from the same clock signal. In a circuit, some inputs are common to the two machines.

   For correct operation, machine B should signal RcvAtoB in the same cycle as machine A signals SendAtoB (and similarly machine A should signal RcvBtoA in the same cycle as machine B signals SendBtoA). If the machines do not activate these complementary signal pairs simultaneously data may be lost (*e.g.* if A sends and B does not receive) or one machine may receive garbage (*e.g.* if A receives but B does not send). Since the FSMs run independently after reset, it is possible that an incorrectly designed pair of FSMs might not always simultaneously present the complementary communication signals.

   Provide a verification routine which takes in two such FSMs, A and B, and a description of their common input signals and determines whether or not all communications operation are correctly paired.

3. [**4 pts**] *Common Operator Subgraphs*: In Callahan's compilation of C to reconfigurable logic, he generated a set of hyperblocks that he then mapped to spatial logic on an FPGA. Only one of the hyperblocks is ever active at a time. Consequently, it might be beneficial to share logic between hyperblocks. The most trivial case would be two hyperblocks that contained equivalent dataflow graphs (here equivalence would be the choice of operators and their connectivity; the input control flow and input variables might be different between the two hyperblocks; similar the output variables and control flow might be different as well). A more general case would be to share some subset of the dataflow graph (*e.g.* maybe one graph computes Z=(A+B+C*D)>>2 and another computes Q=(R+S+T*U)%7; both graphs share the subgraph i1+i2+i3*i4.

   Provide an algorithm to minimize the amount of unique datapath logic required by maximally sharing subgraphs among hyperblocks. That is, your goal is to minimize the number of operators which must be spatially implemented. Good solutions will also minimize the amount of multiplexing logic added to allow subgraph sharing. (A complete and practical solution will consider the impact of adding multiplexing and control logic to allow sharing along with the savings due to operator sharing; to make this problem simpler, we're allowing you to assume the operator area dominates control area for this problem.)

   (a) Search the literature and find one or more papers which address a problem similar to this one. (From the reading and discussion in this course, you should have gained experience reading the literature in this area.)

   (b) Provide proper citations for the paper(s) you found and will use to answer the following questions.

   (c) Describe how the problem solved in the paper(s) is similar to and different from this problem.

   (d) In your own words, summarize the algorithm. (I don't want a verbatim copy of their algorithm or their description; extract the essence. Where possible compare and contrast with techniques and ideas introduced and developed in the course.)

   (e) Describe how you would use or modify the algorithm to solve this problem.

   (f) Describe the algorithm runtime (asymptotic complexity) and optimality.

4. [**4 pts**] *Cascades*: Modern FPGAs have hardwired connections between adjacent LUTs which allow fast signal propagation without traversing slow programmable interconnect (*e.g.* carry chains). The hardwired connection means the input can only come from one position (*e.g.* the LUT to the immediate left). Adapting techniques from this course, describe an approach which exploits these cascades to reduce circuit delay, starting from an unoptimized RTL netlist and ending with a LUT-mapped and placed design.

   For concreteness assume:

   1. K=4-LUT logic blocks with the hardwired cascade being a 5th input
   2. You can choose not to use the 5th input (in which case the logic block is simply a 4-LUT)
   3. Island-style FPGA with 1 LUT per CLB/Island
   4. Fast cascades run horizontally left to right along the rows of the FPGA

   (a) State any further assumptions necessary.

   (b) Describe how you decompose this task into subproblems and any tradeoffs associated with this decomposition.

   (c) Describe the cost functions used in your flow.

   (d) Describe each of the algorithms used in your mapping flow.