Final Exam problems

1. **Logic Synthesis.** Because the midterm and subsequent homework were done well – I cancel the logic synthesis part of the exam for your class.
2. **Basic combinatorial problems.** Graph coloring, binate and unate covering, maximum cliques, compatibility and incompatibility relations. All this material is of use in entire design automation not only in logic synthesis so it can be used as parts of the problems in the exam. But will be not tested by itself.
3. **Evolutionary programming to solve CAD and optimization problems;** Students should be able to explain and illustrate on examples what is Genetic Algorithm, Genetic Programming, realizations with arbitrary encodings and data structures, genetic engineering and Lamarckian ideas. You have to understand role of human in GA loop, role of various variants in practical problems, and especially methods to formulate fitness function, encoding. Because the project is going well – I cancel the evolutionary programming part of the exam for your class.
4. **Backtracking methods in CAD and optimization.** Because this is included in project and the project is going well – I cancel this from final exam for your class.
5. **Binary Decision Diagrams and extensions.** Students should be able to explain Shannon, and Davio expansions, and calculate them using Kmaps, expressions or truth tables. Next, use them to create decision trees and decision diagrams. Applications of decision diagrams in synthesis, simulation, verification and representation. Role of canonicity.
6. **Fast Transforms and Butterflies.** Students should be able to draw kernels of Fast Fourier Transform, Walsh Transform and Reed-Muller Transform. Illustrate transformation of a Kmap of Boolean Function to its Positive Polarity Reed-Muller Form using a butterfly based on RM Kernel. Describe pipelined, sequential non-pipelined and combinational realization of such a circuit.
7. **Retiming.** Given is a sequential circuit with D flip-flops (registers). Based on retiming rules move registers to improve timing or decrease the number of flip-flops. For pipelined design insert flip-flops between combinational blocks or partition combinational blocks first and next add flip-flops between partitions. Next move registers to optimize the circuit. For circuits with feedback consider replacing a delay with a sequence of delays and next do retiming. Sometimes consider also adding additional delay which makes the circuit not entirely equivalent but equivalent with accuracy of a delay. Consider cutting the feedback loop and redrawing circuits to be a DAG before adding delays in all paths and next moving them.
8. **Pipeline design and controller.** Students should be able to start from equations, Laplace, Z or other transform and draw the data flow-graph of pipelined DSP-like circuits. FIR and IIR Filters, image filters, equation solvers are good examples. Next the controller FSM for this pipeline should be designed and optimized. Timing table for a pipeline. Analysis of this table. Use of this table for verification of retiming, especially of circuits with feedback.
9. **Non-Pipelined Sequential Data Path with Control Unit (this is also called Controller or Glushkov Machine).** Student should be able to create a flowchart based on problem specification, next design by hand a data path based on the operations used in the flowchart and synthesize formally a Mealy or Moore controller from the flowchart. If minimization and assignment is too difficult, at least the students should know how to directly convert flowchart to graph of D-flip-flops, OR and branching

gates. Examples: Wave generators, Manchester circuits. Timing analysis and optimization for speed.

10. **Design of parallel controllers.** Data path for description on the level of register-transfer. Do the same as in point 7, but design a Petri-net like, parallel or non-deterministic controller. Role of nodes JOIN, FORK, DAND, DEXOR and normal oring nodes in such framework.
11. **Reachability analysis.** Convert non-deterministic FSM to a deterministic FSM using reachability analysis. Convert parallel graph to serial graph using reachability analysis. Role of analysis in formal verification.
12. **Regular expressions.** Definition of a regular language. Generating and accepting regular languages. Rabin-Scott automata. How to write regular expressions for simple events. Iteration, concatenation and union. Complex expressions. Examples of complex events. Keys and locks, sequence acceptors. The role of trap states. Conversion from regular expression to a regular graph. Adding and removing lambdas. Conversion to deterministic FSM. Examples of languages that are not regular. Counting and context-free languages.
13. **Iterative circuits.** The concept of iterative circuit. One dimensional and more dimensional iterative circuits. One-directional carry and two-directional carry signals. Relation between one-directional, one-dimensional interative circuits and Finite State Machines. Trade-off between speed and area in digital design. Trade-off between time and space realization of iteration – most important issue in digital design and how it relates to iterative circuits and FSMs. Examples of iterative circuits and FSMs. Comparator A>B, comparator A=B, circuit for finding the first one from left, circuit to filter ones between the first and the last ones in the sequence. Maximum circuits, sorters. Iterative versus trees. left ones, both ones, max A=B
14. **Turing Machine.** Why is it important? The concept and its realization. Example of Turing machine to do the following operations: copy n, copy n+1, 2n, n+m, n*m. Design of controllers with data path and control unit for these Turing machines. How to realize the tape? Control signals for the tape. Systematic and ad-hoc design of these types of controllers.
15. **Machines with stack.** As above, but you are controlling a stack instead of a complete tape. Examples of languages recognized by stack machines. Palindromes, language A^n B^n and similar. Deterministic and non-deterministic languages recognized by stack machines.
16. **Arithmetic**. Rational Arithmetic, complex arithmetic – data paths. Counter arithmetic. Controllers to add, multiply and divide using up-down counters. Design of controllers. Role of invariants in optimization. Waveform generators based on counters. GCD and LCM circuits. Euclid Algorithm.
17. **Compiler optimization for software and hardware.** Similarities. Role of transformations. Timing and area improvements. Use of distributivity, associativity, etc.
18. **Sequential Circuits.** Type D, JK, RS, T Flip-flops. Realization of machines with arbitrary flip-flops. State minimization of completely specified machines.
19. **Minimization of Incompletely Specified Finite State Machines.** Covering closure problem. Relation to binate covering. Maximum compatibles. Minimization based on maximum cliques. Algorithmic method of finding covering and closure. Graphic method of finding covering and closure. Ad-hoc methods of guessing and verifying good compatibles. Triangular table, its creation and solution.
20. **State Assignment of Synchronous Finite State Machines.** The problem of state assignment. Coding and codes. Partitions determined by inputs, states and outputs. Rule based state assignment. Hypercube-based state assignment. Many ways of creating the graph for embedding in hypercube. Numerical method of weighting the edges of the graph. How to draw a hypercube in n-dimension. Hypercube and a Kmap. Assignment method based on

column and output partitions. Multi-line method and basic theorem. The concept of partition pairs and partition lattices and how they are useful in state assignment. Operations on partitions – graphical and algebraic realizations. How to combine many state assignment methods to get a good solution quickly. State assignment for JK flip-flops and incompletely specified state machines.

21. **Microprogrammed Control Unit**. Structure of the simple microprogrammed unit without stack. Convert a flowchart to a binary ROM (RAM) table that programs output signals, next (jump) states and predicate selection addresses.
22. **Scheduling**. The scheduling problem. Scheduling without constraints. Scheduling under timing constraints. Scheduling under resource constraints. The Integer Linear Programming (ILP) approaches. Heuristic scheduling based on known methods such as graph coloring. ASAP and ALAP and their uses. Hu's algorithm, List scheduling. Force-directed scheduling. Scheduling with chaining.
23. **Resource Sharing**. Resource dominated circuits. Flat and hierarchical graphs. Functional and memory resources. Non-resource dominated circuits. Concurrent scheduling and binding. Allocation and binding. Optimum sharing problem. Compatibility graphs and conflict graphs. Algorithmic solutions to the optimum binding problem. Perfect graphs. Data-Flow graphs, their properties. ILP formulation of binding. Hierarchical sequencing graphs. Register binding problem. Multi-Port Memory binding. Bus sharing and binding. Scheduling and binding resource dominated circuits. Scheduling and binding approaches: scheduling before or after binding. Module selection problem and solutions. Examples of comprehensive data path design based on scheduling and binding algorithms.