Logic Synthesis Term Projects

Professors Adnan Aziz (U. Texas, Austin)
and R. Brayton (U. Calif., Berkeley)

March 19, 1996

Introduction

The purpose of these projects is:

1. it is 40% of your grade (but this should be the least important item)
2. working on this project should be good training on how to go about approaching an original research problem.
3. the project should yield results
   - routines that can be incorporated into SIS or VIS,
   - experimental findings on the efficacy of proposed algorithms, and
   - new theory techniques for synthesis.

Projects should be done individually, with close cooperation with the mentor.

Timeline

Project selection: You should make a decision as to the projects you are interested in by **Thursday Morning, Mar. 21** Send me e-mail about your selections. Give me your top 3 choices. I will make my selections about which project you get by when you made the choice, your ability to carry out the project, and the choices the others in the class make.

Progress monitoring: You should make contact immediately with the project mentor(s) and establish some regular basis for meeting and reviewing your progress. I do not want a student going off on a tangent only for us to learn at the end of the semester that this happened. On the other hand we do not want to stifle creativity by monitoring things too closely. Further, I don't want rush jobs, where everything is left to the end of the semester. You need time to think about these projects.

Final reports: A 10 to 20 page document describing the project should be turned in by midnight Friday May 17. It should take on the format of a conference paper.

Evaluation

Your grade on the project will be based on a number of factors, particularly the originality and quality of the work. Other considerations include clarity (both the written report and the presentation), and attention to detail.

Suggested Projects

Below, are sketches of topics that have been suggested by various people, in particular Adnan Aziz who is teaching a parallel class at the University of Texas, Austin.

1. **Implementing the Lehman-Watanabe Technology Mapping Method in SIS**

   This is a new technique reported at ICCAD'95 and solves a long standing problem on technology mapping. Papers can be obtained from R. Brayton

   Mentor: H. Savoj, R. Brayton

2. **Domino logic synthesis and technology mapping**

   Domino logic circuits are used in high performance designs (by Intel, IBM, DEC, etc.) because their low switching threshold and reduced transistor count lead to fast and area efficient logic implementations.

   Domino logic is itself monotonic and therefore unate. This allows for highly efficient algorithms for complementation, reduction, and tautology checking and is therefore an ideal logic synthesis target. It is also robust path delay-fault testable(RPDFT). Unfortunately, straight domino logic cannot implement inverting functions. Dual rail signals (where an input and it's complement are represented using separate monotone signals) are necessary to compute an inverting function, and this leads to a less area efficient solution (equivalent to CMOS). A goal in synthesizing domino logic efficiently is to reduce the amount of dual rail logic.

   A goal for technology mapping is to obtain an area-efficient logic implementation mapped onto domino gates. Domino logic gates impose certain restrictions:

   - only non-inverting gates are available
   - function inversions require input complements to be available
   - wide ORs are very fast and area-efficient
   - dual rail XORs are very fast and area-efficient
   - latches can incorporate logic functionality
   - highly complex (AND/OR) gate functions are available

   Within these constraints one would like to efficiently map logic onto non-inverting domino stages. As a final idea, sets of pure CMOS gates (with an even number of inversions) could be used to implement monotone sections of logic as a form of area recovery along non-speed-critical paths.

Mentor: D. Kirkpatrick, R. McGeer, R. Brayton

3. **State Assignment for Domino logic**

   The goal here is to minimize the amount of dual rail logic needed for computing the next state and output functions. A known result is an encoding technique which eliminates the need for dual rail logic in the next state functions of dynamic PLAs [1]. This project would extend this idea to minimizing the amount of dual rail logic given a fixed input/output encoding by computing an optimal state encoding for FSMs.

   [1] M. Mukaidono, Y. Nagata, and C. Afuso. "An Optimum Half-Hot Code Assignment Algorithm for Input Encoding and Its Application to Finite State Machines", In IEICE Transactions on Information and Systems, pp 1231-8, Oct. 1995.

   Mentor: D. Kirkpatrick

4. **Manual and Automatic Abstraction in VIS**

   We want a method implemented in VIS where a module can be abstracted and used to replace a given one, for the purposes of running verification checks. Currently, VIS has only one method for abstraction. The user can input a file giving a number of signals to cut. These then become non-deterministic inputs and it is easy to see that this leads to a conservative abstraction with more behavior. Thus for many properties if they are proved with this abstraction, they also hold for the original design. In the AT&T verification tool COSPAN, they use the notion of a set of signals to cut and a set of signals not to cut. Then an iteration is used to try some additional cuts based on results of verification and error traces. Their experience is that this iterative mode is useful when aided manually by the cut/no-cut variables provided by the designer. Previous work using error traces has been done to take a too conservative abstraction and make it more concrete, but usually these modifications need user guidance.

   This project should implement some new abstraction capability in VIS perhaps working on a user interface where the user helps guide the abstraction being done. Some abstraction techniques are:

   (a) implement the cut/no-cut idea with some user friendly input for specifying signal selections. This is to be extended to some method for iterating the cut points.

   (b) group selected states or signals together. These are called state or language homomorphisms. A method for homomorphism checking needs to be implemented here. Homomorphism specification needs to be designed.

   Mentor: F. Balarin, S. Tasiran, R. Ranjan, G. Swamy

5. **Major Design Effort in Hierarchical synthesis and verification using VIS**

The goal of this project is to use VIS for a major design effort. As an example, one could develop a model for the DLX microprocessor in Verilog and verify it in VIS. The design should be developed in a top down fashion, starting from a simple abstract model.

The model should be validated using both the simulation and formal verification capabilities of VIS. Verification should be an integral part of the design flow: you should verify the design at each stage.

Synthesis should be applied to components at each stage, in order to get estimates on the performance and area of the entire circuit. The tradeoffs between the time taken to synthesize large portions of the design, and the quality of the result should be noted. Selective collapsing can be used to explore trade-offs.

For a large design, both formal verification and the more sophisticated synthesis algorithms (e.g. *full_simplify*) take prohibitively long. You will need to use the hierarchical synthesis as well as abstraction mechanisms in VIS to cope with complex designs.

The design should be sufficiently complex for this to be a non-trivial exercise. For example, if you are developing a model for a microprocessor, you should incorporate features such as pipelining, bypass logic, and/or out of order execution. Possibilities other than microprocessors include multi-cache protocols and bus protocols.

The source code for the design should be in Verilog and sufficiently documented. In the long run it is hoped that these designs will become standard benchmarks for both verification and hierarchical synthesis.

Mentor: A. Isles, R. Hojati, G. Swamy

6. **Synthesis of externally synchronous, internally asynchronous circuits**

Recent research (by Varshavsky, Marakhovsky and Chu; L. Lavagno can send the papers if this project is chosen) has shown that it is possible to take any two-phase or four-phase SYNCHRONOUS circuit, and build a clock control network that will guarantee

- the same externally observable behavior,
- local (ASYNCHRONOUS) synchronization and timing constraints.

Their synthesis procedure is somewhat unclear and given "by example". It basically amounts to building simple handshaking circuitry, for which the original two-phase or four-phase discipline ensures the absence of deadlocks. The authors are well known, and hence the algorithm can probably be formalized, implemented (in SIS) and tested (by simulation).

The project requires a little reading, a little programming, some simulation and (if possible) a formalization of an empirical discussion. There are some chances of publication IF the generalization is interesting and well-done. Possibly this can also have a lot of practical relevance in years to come. There is also sufficient interest in this to lead to multi-year research support (if done well).

Mentor: L. Lavagno

7. **Sequential Redundancy Identification and Removal**

   The project is to identify a subset of sequential redundancies in circuits without assuming all latches have reset lines. Iyer et al [DAC96] have a procedure for propagating contradictions in sequential circuits. If having two different values on a net requires a contradiction on another net, the first net is a sequential redundancy. The goal of this project is to initially implement these ideas and do further work in removing these redundancies. There is some work to be done in combining the effects of removing multiple redundancies. Also, it might make sense to use the information propagated from these contradictions to simplify nodes (instead of replacing them by constants).

   If some new ideas are discovered, the project can lead to a conference paper.

   Mentor: V. Singhal, S. Qadeer

8. **Partitioning for Synthesis**

   Many sequential logic synthesis algorithms are not very robust in the size of the circuit. Examples are safe replacement algorithms, and FSM re-encoding. This goal of this project is to automatically decompose sequential circuits so that it is possible to apply these algorithms to optimize random pieces of a larger design. A heuristic due to D. Long et. al. can be a starting point to obtain these sequential partitions.

   This project involves more experimentation. It requires an early implementation and testing many heuristics.

   Mentor: V. Singhal, T. Villa, E. Goldberg

9. **Equivalence reductions for a given CTL formula**

   In 1994, Aziz, Shiple, Singhal and Sangiovanni presented a state equivalence that is defined with respect to a given CTL formula. Since it does not attempt to preserve all CTL formulas, like bisimulation does, it computes coarser equivalences. This equivalence is used to manage the size of the transition relations encountered when model checking a system of interacting FSMs. Specifically, the equivalence is used to reduce the size of each component FSM, so that their composition will be smaller. This method can be applied, whether an explicit representation is used for the FSMs, or BDDs are used. Also, in some cases this approach can detect if a formula passes or fails, without having to compose all the component machines. The method is exact and fully automatic, and handles full CTL. This project is to implement this method in VIS, and assess its usefulness on a series of benchmark examples.

   Mentor: T. Shiple

10. **Synthesis for High-Speed simulation**

    There has been recent work in using BDDs to dramatically speed up simulation of sequential designs at the level of gates and latches [Ashar and Malik'95,McGeer and Saldanha'95] (this is referred to as *cycle simulation*).

The underlying idea is as follows: consider a combinational net-list on 100 inputs, 1 output and 10^5 gates. Simulating one input vector (using the most direct method, viz propagating it through each gate) will take of the order of 10^5 operations. However if you were to build a BDD for the output in terms of the inputs, you could simulate an input vector in 100 operations by simply following the appropriate branches in the BDD.

There are problems with this approach. First of all, the BDD for the output may be too large. This can be overcome by building BDDs for intermediate nodes in the circuit until a threshold value is reached. In this way new variables are introduced and the number of lookups is increased (although it expected that it will still be far less than the number of gates in the original network). Another problem is that the design may have a large number of outputs. In this case we can build a single BDD (the characteristic function) for several outputs at the same time. and evaluate multiple outputs in the same pass.

This problem can be thought of as a problem in synthesis in the following sense — usually in synthesis we try to minimize the area/time/etc of the implementation. However, here we are trying to develop an implementation which can be simulated efficiently [Murgai95, Newton'93].

A primitive simulator exists in VIS; it incorporates some of these ideas. The goal of this project is to use the existing simulator front end, and build a new, faster, back-end based on the ideas mentioned above. In order to achieve maximum performance it may be necessary to make some restrictions, e.g. only handle designs where all signals are binary-valued. Also note that fast simulation is VIS has several good applications (e.g. project 13.

Mentor: S. Khatri, R. McGeer, A. Saldanha

11. **Converting combinational logic into sequential logic**

This project was attempted in the past with little success and it is a little far out but interesting. But now maybe we know more. It is easy to take a sequential design and "unroll" it. If the number of iterations is known then the design becomes purely combinational. It is also easy to "pipeline" a combinational design to add latches between stages. However, the idea of this project is to convert to a real sequential design (non-pipelined). I am sure that this is not going to work very well for all types of logic, so part of the project would be to come up with some measure or test on what logic this could be applied to. The main idea would be to "reuse" logic as the combinational design is folded up. The new state added would be used to differentiate one stage from the next and to control what the logic is computing. The challenge would be to introduce as little state as possible.

It is possible that the idea of incremental re-synthesis can be used here. Consider that we have carved up the combinational logic into say 3 pipeline stages. Each has its own combinational logic. But now we want to reuse one of the stages and modify it dynamically, controlled by the state to reuse the logic and compute the other stages. Thus for each other stage, we have a new "spec" for the selected logic and want to modify it incrementally to meet this spec. The modification should use as little new logic as possible and need few new states to control it. This idea has some correlation to a recent (Berkeley) PhD thesis by N. Bhat who introduced a new FPGA architecture which used pipelining and logic reuse to get good area/speed implementations.

Indeed, this may have some application to FPGA synthesis where there is an excess of latches.

Mentor: A. Aziz, R. Brayton, Y. Kukimoto

12. **Determinizing Boolean Relations**

We have seen examples of Boolean functions $f : B^n \rightarrow B^m$ and Boolean relations which are any subsets of B^{n+m}. Usually we are interested only in Boolean relations R which are complete, i.e. for every x in B^n there is at least one y in B^m such that $(x, y) \in R$. A Boolean relation can be represented by its characteristic function or in table format (analogously to the ESPRESSO PLA format) as shown below:

```
.table x1 x2 -> y1 y2
0 0 1 0
0 0 0 1
1 - 0 0
- 1 0 0
```

If there is only one output, then a complete Boolean relation is an incompletely specified function.

Boolean relations arise in various contexts (refer to [DeMicheli Book] for examples). It is often useful (in verification contexts) to "determinize" relations, to get functions which are (in a sense that can be made precise) equivalent to the relation. This can be done by adding a (multi-valued) external input, e.g. the relation above can be made into a function by adding an input `x0` as shown below:

```
.table x0 x1 x2 -> y1 y2
0 0 0 1 0
1 0 0 0 1
- 1 - 0 0
- - 1 0 0
```

Now both `y1` and `y2` are completely specified functions.

We have shown (in unpublished work) that finding the least number of inputs that needs to be added to a relation in order to determinize it is NP-complete. We have also devised an algorithm for determinization that proceeds by building an undirected graph, whose vertices correspond to the rows of the table specifying the relation. Edges are inserted between vertices whose corresponding input parts of the rows intersect but have different outputs. It is relatively easy to see that the least number of colors needed to "color" this graph is the least number of values a new multi-valued input must take in order to determinize this relation.

The goal of this project is to implement a determinization routine based on heuristics for graph coloring. This approach may also yield a procedure for synthesizing Boolean relations, and it would be of interest to compare the quality of results obtained by this method to those say of `GYOCRO` [Watanabe'91].

Mentors: A. Aziz, R. Ranjan, and R. Brayton

13. **Hybrid Approach to Sequential ATPG**

BDD based approaches to ATPG proceed by taking the faulty circuit and the good circuit, and building BDDs for the outputs for each of the networks. The BDDs in terms of primary inputs and present states for corresponding outputs are compared; if for some output the BDDs differ, an input on which they differ is a test for the fault provided the present state input can be "justified" and output differences can be "propagated" to a primary output. (In fact these approaches do a lot of "pruning" of the circuit, and try and identify common points in the circuits, but this is the essential idea.)

The major limitation of this approach is that the BDDs can grow very large. The following strategy for sequential ATPG is proposed: build a BDD for all states where the primary outputs differ (a state is a pair of states, one from the good circuit and one from the faulty circuit). Call this set B_0. Define B_1 to be all states which can get to B_0 on application of one input vector. In this way inductively define B_k; there are routine methods for building the BDD for B_{k+1} from the BDD for B_k and the BDDs for the circuit. At some stage, suppose the BDD for B_k grows too large. Call this set B^*. Now start random simulation from the initial state. If a state in B^* is reached, the fault is not redundant, and a test for it is given.

Checking to see if a state in B^* is reached is easy using BDDs. Since B^* will typically contain many more states than B_0, you have a better chance of reaching it. Thus this approach should be substantially faster than simply simulating randomly. The goal of this project is to implement these ideas and check this conjecture. Note the relation with projects 7 and 10. Note also that one can use under-approximations to the B_k. Hence there may be some use of ideas of [Shiple, Hachtel] on under-approximations.

Mentor: A. Aziz, A. Saldanha

14. **Experimenting with SAT packages**

The equivalence of Boolean networks can be performed efficiently by reducing the check to the test of the satisfiability of a POS Boolean formula; the networks are different if there is an assignment of values to the variables which makes the formula true. Various researchers have reported that applying heuristic SAT procedures to such a formula leads to procedures for equivalence checking that are superior to other methods [Larrabee,Stephan].

There have been advances in SAT routines recently (they have been applied to a completely different context) [references?]. The goal of this project is to compare these new procedures with those described by [Stephan,Larrabee], and to develop routines to write out the functionality of combinational VIS networks as POS formulas.

Mentor: A. Aziz, A. Saldanha, V. Singhal

15. **Sequential synthesis using S1S (not SIS)**

    In recent work [Aziz-ICCAD95], we have developed a theoretical framework was developed for synthesis of FSM networks. In particular, a procedure was given for deriving a single (non-deterministic) finite state machine that captures (in a very precise sense) all the flexibility available for synthesis at a node in the FSM network. This procedure relies on manipulating finite state machines essentially as automata: the essential operations are forming the "product" of two automata, and determinizing automata, using the "subset" construction.

    The worst case complexity of our procedure is high; exponential to be precise, because of the need to determinize. However this is a worst case bound; in practice it seems that the subset construction rarely leads to an exponential blowup. Coupled with state minimization techniques [Aziz-ICCD], the procedure could be quite reasonable. The goal of this project is to implement this procedure, and its extensions to synthesis in the presence of fairness and non-determinism. (There are some existing automata toolkits that may prove to be useful [Tasiran-Hojati'95]).

    Mentor: A. Aziz, S. Tasiran, T. Villa

16. **Decision procedures for the Linear Theory of the Reals**

    The "linear theory of the reals" is a logical formalism for analyzing the properties of the real numbers with the operators + and <. There is a natural notion of "definability" in this logic. For example, the inequalities

    $$x_1 + 2x_2 - 3x_3 \quad \leq 4$$

    correspond to a particular half-space in $\Re^3$. Using logical ANDs and NOTs, we can build "richer" sets. Another logical operator is the "existential operator" $\exists$. Given a set A in 3 dimensions corresponding to variables x_1, x_2, x_3, $\exists x_3 A$ is the projection of A down to the variables x_1, x_2. There is a theorem stating that all such operations yields sets which can be represented as a union of regions defined by sets of inequalities.

    These sets, and their corresponding manipulations, naturally arise in the analysis of timed circuits [Aziz-Tau'95, Henzinger-survey]. The goal of this project is to build a package which can efficiently represent and operate on such sets. A straightforward implementation would be a good starting point; however there are many issues related to efficient representation which remain open, and need to be resolved. Ultimately, we want an efficient implementation of a decision procedure for this theory.

    Mentor: A. Aziz, T. Henzinger

17. **Multiple Constraint Performance Optimization**

    In some designs, there may be more than one set of arrival times and required times for a circuit. If we reduce these constraints into one, we may over-constrain the circuit. How should we change the optimization at technology independent and mapping stage to handle these constraints?

For more detail contact H. Savoj directly, e-mail: hamid@cds8830.Cadence.COM

Mentor: H. Savoj

18. **Improve Node Limit in BDD package**

    Most BDD packages have a node limit that lets a particular BDD operation quit if the size of BDD's grows too large. There are cases under which the node limit is not reached but a BDD operation hangs up. Find out why this is happening and find a solution. It may be the the use of the new BFS BDD package `CAL` can be of use here.

    Mentor: H. Savoj, R. Ranjan

19. **Gate Sizing**

    Gate sizing has become an important part of logic synthesis. Once the first pass of synthesis is finished, a floorplanning is done. Then the wire delays are recomputed based on the floorplanning results. The delays are passed to the synthesizer and gate resizing is performed based on the more accurate delays. Investigate current gate sizing algorithms, find out the shortcomings of the greedy algorithms and come up with new algorithms if possible.

    Mentor: H. Savoj

20. **ESPRESSO and SIS** This project is to update ESPRESSO and/or SIS and experiment with some recent ideas:

    (a) **Controlled Complement** Many algorithms in SIS need to find the complement of a function

    - resub -a,
    - Satisfiability don't cares,
    - Observability don't cares,
    - simplify,
    - full_simplify.

    Come up with a new complementation procedure that is controlled and quits if the size of complement is greater than a limit. Switch all the algorithms in SIS to use this new function.

    (b) Implement in `ESPRESSO` the version of LASTGASP due to E. Goldberg, and perform thorough experiments for comparison on benchmark circuits. This also requires developing an efficient implementation.

    (c) Experiment with unate complementation ala Mathony to see if this leads to a more efficient algorithm.

    Mentor: - H. Savoj

21. **Verilog Compiler**

    Modify the Verilog compiler to take inputs which are fully compatible with the Synopsys synthesis subset of Verilog. Currently, out Verilog compiler for VIS is not compatible with the synthesis subset defined by Synopsys and used by many companies. In particular, we use enumerated types but do not give their encodings. Our method for specifying enumerated types is also different. We need an option in the Verilog compiler `vl2mv` (say -Synopsys) which allows us to compile any such Verilog file.

    Mentor: - ST Cheng, T. Shiple

22. **Partial Reached State Computation**

    Computing the set of reached states in a *finite state machine* (FSM) is a key operation in both verification and synthesis. The reached state set allows designers to check a design for errors and other unwanted behavior. It is a precise measure of the exact set of states a design can take under all possible inputs.

    Given a directed graph, and a set of initial nodes in the graph, reachability computes the set of nodes on some path from the initial nodes. A FSM can be represented by a directed graph, which is also called a *state transition graph*. Computing the reachable states (nodes) of this graph can be done by a breadth first search traversal of the state transition graph beginning at the initial states.

    Unfortunately this computation explodes when the number of states in the FSM becomes very large, as is the case in real designs. This is often called the *state explosion problem*. To attempt to overcome this problem, BDD's are used to represent the required quantities. e.g. the transition relation (TR), which implicitly represents the FSM's state transition graph, and the initial and reachable sets of states.

    In order to further combat the state explosion problem, we propose to perform modifications on the transition relation before computing the reached state set. This will have the effect of simplifying the computation while introducing some error. By limiting the types of modifications to the TR, we can assure that the value returned by the reachability computation is either an upper or a lower bound. By performing intelligent modifications we hope to achieve adequate bounds on the reached state set without the computation exploding. Also, given a sequential machine described by a net-list of gates and latches, in some cases the logic is too big to extract the state transition graph. Another idea is to approximate the reached state set as an intersection of approximate reached state sets each computed by restricting to a few latches. The set of latches for each computation can be selected dynamically, and can be made dependent on already computed sets.

    The project would be to develop methods for BDD modification and for selecting subsets of latches, and then to provide an efficient implementation of these ideas. Some code for these ideas has already been written and can possibly be used to build on, although it needs to be made more efficient.

Mentors: Gitanjali Swamy, Sriram Krishnan, H. Savoj

23. **New method for using don't cares for reducing BDD size**

    An extensive study [Shiple-Hojati-Brayton-Sangiovanni,DAC'94] compared many possible ways for using don't cares to reduce BDD size. Recently, Burch and McMillan have come up with another proposal. It has the nice property that the resulting BDD is guaranteed to be no bigger than the original. The project would be to implement this in the most efficient way and compare with the best technique known so far.

    Mentor: J. Burch, T. Shiple

Miscellaneous

Projects of these types will naturally build upon existing work. You are encouraged to build upon existing code/results, and in your report you may adapt from others papers and manuscripts. However, you must explicitly make it clear that you have done so; failure to report this will be considered **plagiarism**. In the past, this has been dealt with severely.

We hope that many of these projects will lead to code that will continue to be useful after the course is over and find life in `ESPRESSO, SIS or VIS`. Some may lead to conference papers, masters degrees, and future research funding. If code is a significant part of your efforts, you are encouraged to document very carefully; in particular, you may want to use the `ext` system to do this. Not only will you have better code, but also you will learn a lot. Projects implemented in VIS have the advantage of exposing you to some modern software engineering practice pioneered in VIS (which is currently being adopted by several companies based on VIS demonstrations).