Compiling Esterel into Better Circuits

Jia Zeng
Department of Computer Science
Columbia University, New York

Abstract
Producing efficient circuits from a high-level language such as Esterel remains a problem. Sparse state coding requires many more latches used than minimum and waste of reachable state space, while tight state encoding produces slow circuits due to the cost of encoding and decoding.

This paper presents an algorithm to generate small and fast circuitry for Esterel. There are three main parts of the algorithm: state assignment, hardware synthesis, and circuit optimization. The technique is based on Program Dependence Graph. It uses heuristic search in coding space, computes the cost and adjusts until finding a compromise point on latch/logic tradeoff.

The algorithm will be used to compile Esterel into small circuits that meet a timing constraint.

1 Introduction

Esterel is a high-level language designed for real-time systems. It supports high-level control constructs such as concurrent composition, preemption, and exceptions. This aspect makes Esterel a more challenging language to translate into circuitry, but also enable aggressive optimizations because the compiler is able to gain a better understanding of the program's behavior.

State assignment to Esterel is based on the Program Dependence Graph (PDG) of Esterel. Baxter and Bauer present it in [9]. It is based on the concept of control flow, and preserves all information of an original Esterel program. Esterel supports implicit state machines through explicit and implicit pause statements that delay for a cycle, such as the await statement. States sustain and transfer only between these statements. So we assign states for each of them. Figure 1 gives an example for PDG.

Heuristic search is used in the algorithm to find an efficient state coding. Three main kinds of coding are used in the searching space. First is Berry's [1] one-hot encoding. It produces fast circuits while gives much redundant state space. Second is Edwards's [2,4] group-hot-by-level encoding. It shares latches between those decision nodes whose parents are in the same level but not parallel. Third is some variant of the former two encodings. It shares latches between the decision nodes in some levels, but not for all levels where sharing is possible. In other levels, it still keeps one-hot encoding for the nodes.

We use heuristic search in the state encoding space until we find a resolution with the fewest latches under the requirement, or until the search space is exhausted. We start from one-hot encoding. If it can't meet the requirement given, we fail and return. Else, we'll try variant coding means to delete sharable latches. Every time after re-encoding, the cost of the circuit is re-evaluated. If it is higher than required, the new coding will be thrown away and the former code will be returned. Or we'll repeat the process until we exhaust the searching space. Thus we find the most efficient coding that meets the cost requirement.

The Esterel hardware synthesis is straightforward when the coding has been chosen.

There are two parts of circuit optimization: combinational optimization and sequential optimization. This paper is concentrate on the sequential optimization. And in fact, the sequential optimization has been done at the stage of state encoding before generate real circuits. SIS, the standard public-domain optimizer, will be used to for the combinational optimization after generating circuits.

2 Related Work

The classic state assignment is based on Finite State Machine. Hachtel and Somenzi [6] described synthesis of finite state machines. It uses minimization of incomplete specified machines to get reduced reachable states.

Villa and Sangiovanni-Vincentelli [7] present algorithms used in NOVA for optimal state assignment of FSMs. It is based on the state code adjacency concept but more efficient and flexible. NOVA represents constraint satisfaction as a graph-embedding problem. It uses heuristic search to resolve this problem. Its best strategy is "iohybrid_code", which produces results with quality comparable to the results of the maximum adjacent method. Its core algorithm is "ihybrid_code". The set of input constrains is partitioned into satisfied constrain set (SIC) and rejected constrain set (RIC) at the beginning. The algorithm first gives the coding with minimum length under the satisfied constrains. Then it increases the embedding cube to satisfy the RIC within the encoding space that is specified by the user. The iohybrid_code strategy takes similar steps as in ihybrid_code but also takes the output constrains into consider. Generally speaking, output constrains are in lower priority to input ones in this strategy.

Devadas, Ma, Newton, and Sangiovanni-Vincentelli

[8] present a method called MUSTANG that is one of the earliest multi-level state assignment methods. It used the state code adjacency concept to reach the aim of maximizing the size of number of common cubes. It build an attraction graph with weighted edges. An edge's weight is increased if it links to the common fanout and fanin states. MUSTANG was used to help MIS logic synthesis system reducing the number of product terms or literals needed to implement the next-state and output functions.

Berry first outlined the translation of Esterel into circuitry in 1992 [1], refined later to cover reincarnation. It generates a sub-circuit for each statement, and registers only for pause - the kernel unit-delay statement. So each leaf state is encoded by one-hot coding. In that case, encoding and decoding circuits are trivial. But it uses many latches and results reachable state space redundancy. Later, Sentovich, Toma and Berry [3,5] described the technique for reducing the number of latches. They rely on computing the reachable state set implicitly using BDDs, then re-synthesizing the circuit using this knowledge to remove sequential redundancies. The whole program is taken as one state machine.

In the compiler we are building, more than one state machine are assigned in different level. It is necessary especially for parallel branches. We can share latches between sequential branches but need to avoid parallel ones.

Edwards [2] proposed three key means to advance Esterel hardware synthesis. First is the CFG. It takes a totally new structural translation to Esterel. Calculating control dependence in the graph, it removes the redundant circuit. That is much more efficient than removing by analyzing the circuit.

Second is a better state encoding. That technique he proposed chooses states encoding at a high level, providing much greater flexibility and larger encoding space to choose from.

Third is to use the don't-care information in logic synthesis. It gives more flexibility to the implementation and helps to generate high-quality circuits.

7 Bibliographies

[1] Gerard Berry. Esterel on hardware. Philosophical Transactions of the Royal Society of London. Series A, 339:87-103, April 1992. Issue 1652, Mechanized Reasoning and Hardware Design.

[2] Stephen A. Edwards. High-level synthesis from the synchronous language Esterel. In Proceedings of the International Workshop of Logic and Synthesis (IWLS). New Orleans, Louisiana, June 2002.

[3] Gerard Berry. Efficient latch optimization using exclusive sets. In Proceedings of the 34th Design Automation Conference, pages 8-11, Anaheim, California, June 1997.

[4] Stephen A. Edwards. An Esterel compiler for large control-dominated systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21(2), February 2002

[5] Ellen M. Sentovich, Horia Toma, Gerard Berry. Latch optimization in circuits generated from high-level descriptions. ICCAD'96, November 1996.

[6] Gary D. Hachtel, Fabio Somenzi. Logic Synthesis and Verification Algorithms. Keluwer Academic Publishers. 1996.

[7] Tiziano Villa, Alberto Sangiovanni-Vincentelli. NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations. In The Proceedings of the 26th ACM/IEEE Design Automation Conference, pages 327-332, June 1989.

[8] S. Devadas, B. Ma, R. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: State Assignment of Finite State Machines Targeting Multi-level Logic Implementations. IEEE Transactions on -Computer-Aided Design, vol. 7, no. 12, December 1988.

[9] W. Baxter and H. R. Bauer, III. The program dependence graph and vectorization. In Proceedings of the Sixteenth Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Austin, TX, 1989.

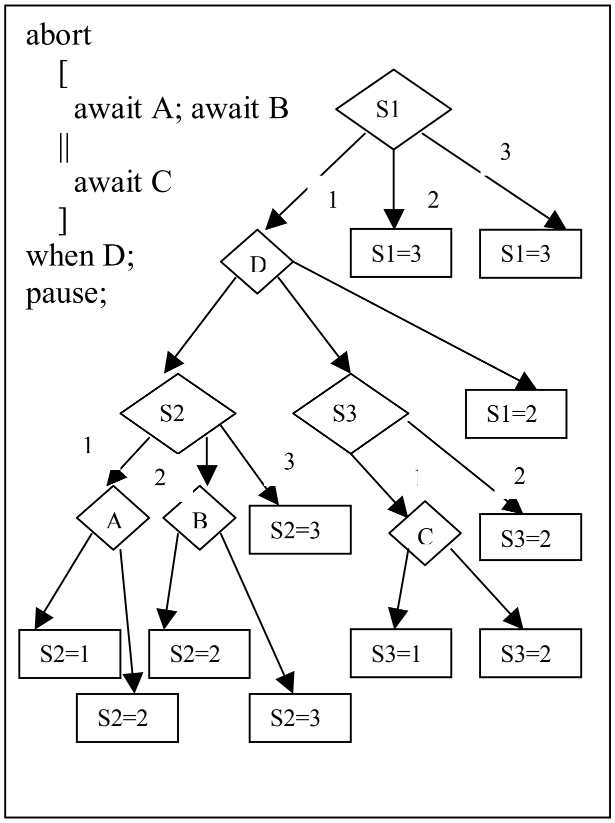

Figure 1 PDG for an example program