MULTI-VALUED LOGIC NETWORK MINIMIZATION AND IT'S APPLICATIONS

by

Yunjian Jiang

Memorandum No. UCB/ERL M00/26

19 May 2000

COVER

MULTI-VALUED LOGIC NETWORK MINIMIZATION AND IT'S APPLICATIONS

by

Yunjian Jiang

Memorandum No. UCB/ERL M00/26

19 May 2000

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Multi-valued logic network minimization and it's applications

by Yunjian Jiang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee:

Professor Robert K. Brayton
Research Advisor

Date

* * * * * *

Professor Kurt Keutzer

Date

Multi-valued logic network minimization and it's applications

Abstract

Multi-valued logic network minimization and it's applications

by

Yunjian Jiang

Master of Science in Engineering-Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Robert K Brayton, Chair

We address the optimization problem of a pure multi-valued logic network. Each node in the network is a logic function with multi-valued inputs and a single multi-valued output. We start with the minimization problem of a multi-valued relation and generalize binary don't cares to partial cares for multi-valued logic. We then look at the observability of a multi-valued function node in a combinational logic network. We give algorithms to generate obserbability don't cares and partial cares for such a node. We propose an application of such a multi-valued network: software compilation for embedded applications. We study the interfacing issues between multi-valued logic networks and software codes, and the minimization problems accounting for the target software implementation.

To Emma

Contents

List of Figures

List of Tables

Acknowledgements

Acknowledgements

I would like to thank my research advisor Prof. Robert Brayton, for his constant support and guidance throughout this two years of study. It has always been exciting and enlightening when we have our weekly discussions. I am also grateful to Prof. Kurt Keutzer, whose professional insight has been encouraging and of great guidance. I also had the chance to work with Prof. Sangiovanni-Vincentelli. His broad knowledge and experience have provided keen suggestions. This work originates from Prof. Brayton's idea and started as a joint EECS219B class project. I would like to thank MVSIS group members Subarna Sinha and Minxi Gao for recent efforts in developing the MVSIS package. I would like to thank all graduate students in 219B class, Fernando De Bernardinis, Rupak Majumdar, Heloise Hwawen Hse, Niraj Shah and Scott Weber. Also I would like express my thanks to my EECS249 project partner Yujia Jin and mentor Freddy Mang.

Life is in your breath.
– Zhuang

Chapter 1

Introduction

Multi-valued logic can be used in hardware synthesis as a higher level representation before the circuit is encoded into binary. There are optimization opportunities at this stage that cannot be discovered in the binary domain. It can be used to solve the optimal encoding problem for the synthesis of finite state machines (FSM). It also has application in machine learning and fuzzy control. In this chapter, we review some of the research work in the multi-valued logic synthesis literature and their state-of-the-art applications. Then we describe our motivation in revisiting this area and the types of problems we are particularly interested in. At the end we will outline the materials covered in this report.

1.1 Multi-valued logic synthesis overview

Most research work in multi-valued logic synthesis has been targeted for the optimal state encoding problem for FSMs [VKBSV97]. For two-level implementation, Rudell *el at* have studied multi-valued PLA minimization problem as an extension of ESPRESSO [RSV88]. The problem addressed there is minimizing the sum-of-product representation of a set of logic functions with both multi-valued inputs and outputs. The theory has been well established and efficient software packages are available. Input variable pairing and encoding are also studied there. This has been applied to the optimal encoding problem with two-level implementations [VSV90] [VSV89].

For multi-level implementation, the synthesis problem formulated has been to minimize a multi-level logic network, with one multi-valued variable appearing at the primary input. The application is state assignment for FSM's with multi-level circuit implementa-

tion. Lavagno *el at* hava established the theory for algebraic decomposition and factorization for this type of applications [LMBSV90] and completed an implementation called MIS-MV.

Boolean relation minimization problem was introduced in [BS89b] and there has been extensive research work in this area [BS89d] [BS89c] [BS89a]. An exact solution was formulated as a binate covering problem. Because of the intrinsic complexity of the problem, researchers have proposed heuristic algorithms [WB93a] [SSB93] [GDN92]. Symbolic relations, as an extension of Boolean relations, have also been studied for the minimization of interacting FSMs [LS90] [WB91b]. Further, observability Boolean relations were studied for the minimization of a combinational logic network with multiple output nodes [Sav92]. Also, symbolic relations were applied to the minimization of a FSM in an interacting FSM network [WB93b] [WGB96]. However, the complexity has hindered the application of such well established theory in real industry synthesis flow.

In summary, driven by the application of state assignment for FSM's, the study of multi-valued logic synthesis has been limited to two-level PLA minimization and multi-level network minimization with single multi-valued variable. For a multi-level logic network with arbitrary multi-valued variables, there has been little theory and application in the literature. The reasons are: (a) The encoding problem from multi-value to binary is difficult, and often the optimization effects in the multi-value domain are obscured by the sub-optimal encoding process. (b) For pure hardware implementation, it is not clear how much gain one can achieve by applying optimization algorithms at multi-valued domain. (c) There is no good multi-valued logic minimization package available for multi-valued networks.

1.2 Our motivation

We address the optimization problem of a pure multi-valued logic network. Each node in the network is a logic function with multi-valued inputs and a single multi-valued output. We propose an application of such a multi-valued network, software compilation for embedded applications. We study the minimization problems associated with such a multi-valued network with the target implementation accounted for in the cost metrics.

First, we consider the minimization of a multi-valued relation. The notion of don't cares used in binary logic [SSL+92] is generalized to multi-valued logic. These contain two types of flexibility, incomplete specification and non-determinism. We define multi-valued don't cares and partial cares to capture this flexibility. Multi-valued functions combined

with multi-valued partial cares are similar to Boolean relations in binary logic, where a set of compatible functions are to be explored for the optimization. Hence, the algorithms developed for Boolean relation minimizations, [BS89d] [WB93b], can be applied in partial care minimization for a multi-valued function.

Second, we consider a multi-valued logic function node in a combinational network, and give the algorithms for generating a compatible set of don't cares for each node. Observability don't cares (ODC) for an MV-node is the set of minterms which, when applied in the network, block the output values of that node, i.e. the primary outputs do not depend on the values of that node. ODC's are useful in the minimization of the MV-function. Compatible ODC's (CODCs) are the don't cares which do not depend on how the don't cares at other MV-nodes in the network are used. The methods to compute ODC's and CODC's are extended from the binary case [Sav92] [SB90] [SBT91]. Observability partial cares (OPC) are the set of the minterms that block a subset of the output values, i.e. the primary output can not distinguish any pair of values in that subset. OPC's provide additional flexibility for the implementation of an MV-node. This is similar to observability Boolean relations for binary networks. However, in order to use the OPC's, a multi-valued relation minimizer needs to be applied. We give one method for generating OPC's.

There are different representations for multi-valued logic functions and relations [BK99]. The most commonly used ones are: sum-of-product forms (SOP) and multi-valued decision diagrams (MDD) for two-level logic and MV-networks for multi-level logic. Logic representations affect the performance of synthesis algorithms. Recent work by Perkowski [GP98] studies the effect of different representations and applies the proposed form to machine learning benchmarks. We use the notation and definitions from [Bra99] throughout this report.

Third, we propose using multi-valued logic synthesis to optimize control dominated software for embedded systems. Control variables in software programs are computed and tested just as variables in multi-valued logic. This can potentially explore logical relations between variables to restructure the control flow of a program. This is not usually considered by traditional compiler optimization. So far, use of logic optimization for software compilation has been limited to binary logic optimization, probably due to lack of an effective MV-optimization package. Part of this thesis explores the interfacing between software code and MV-networks.

1.3 Outline of this report

In Chapter 2 we discuss two-level multi-valued functions and relations. We propose a new way of examine a symbolic relation and define *partial cares* to capture the flexibilities in implementation. We propose the application of multi-valued relations in control synthesis. In particular, we look at the communication protocol converter synthesis problem as a control synthesis problem. We apply multi-valued relation minimization to the synthesis result and show the effectiveness. In Chapter 3 we discuss multi-level logic network minimization problem. We give an algorithm to generate observability don't cares for a node to provider functional flexibilities. We give a heuristic algorithm to encode intermediate multi-valued variables into binary codes. We introduce some applications for multi-level logic networks in Chapter 4. In particular, we study the problem of software compilation for embedded applications and some interfacing issues between software code and an MV-network. We give conclusions and an outline for future research work in Chapter 5.

Chapter 2

Partial care and multi-valued relation minimization

In this chapter we discuss the minimization problem of two level multi-valued relations. We first review some of the definitions and notations for multi-valued relations. Then we explore the flexibilities that can be potentially used in the minimization. Finally we conclude by describing one of the applications of multi-valued relation minimization.

2.1 Two level multi-valued functions

Consider a multi-valued function with multiple multi-valued inputs and a single multi-valued output. A multi-valued relation is, like a binary relation, a one to many mapping. Let $P_1 = \{0, 1, ..., |P_1| - 1\}$, $P_2 = \{0, 1, ..., |P_2| - 1\}$, ..., $P_n = \{0, 1, ..., |P_n| - 1\}$ be the input space, and $Q = \{0, 1, ..., |Q| - 1\}$ be the output space. The multi-valued relation $R : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q$ maps each minterm in the input space, $P_1 \times P_2 \times ... \times P_n$, to a set of values in Q, i.e. $m \in P_1 \times P_2 \times ... \times P_n, R(m) \subseteq 2^Q$. We assume that R is complete, i.e. $R(m) \neq \emptyset$, for all m. Associated with R is a set of multi-valued functions $\{f_i\}$ that are compatible with R, $f_i \prec R$. The multi-valued minimization problem is to find an optimal implementation of f that is compatible with R.

Example 2.1 *Multi-valued relation R_1 is defined in the following mapping table.*

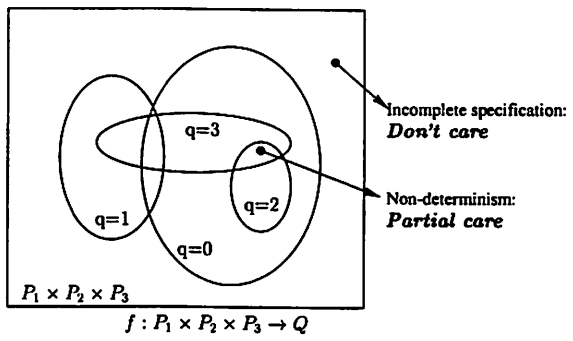

Figure 2.1: Multi-valued functions

R_1 :	$\{0,1\}\times$	$\{0,1,2\}\times$	$\{0,1,2\}$	$\rightarrow$	$\{0,1,2,3\}$
	0	1	–	$\rightarrow$	$\{0,1\}$
	1	–	1	$\rightarrow$	$\{0,2\}$
	–	0	1	$\rightarrow$	$\{0,3\}$
	0	–	2	$\rightarrow$	$\{1,3\}$

The mapping is shown graphically in Figure 2.1. As can be seen, two types of flexibility are revealed, namely incomplete specification and non-determinism. There are unspecified minterms in the truth table, e.g. $\{0\}\times\{0\}\times\{0\}$, which can take any value in Q for the mapping. This represents the traditional don't care minterms in binary logic domain. There are also minterms that can take a subset of the values in Q, e.g. $\{0\}\times\{1\}\times\{2\}\rightarrow\{0,1,3\}$. We call these *partial care* minterms. This situation is not present in binary logic, where each minterm can take either value 0 or value 1 if it is not a don't care.

Definition 2.1 (Don't Care) *A minterm* $m \in P_1 \times P_2 \times ... \times P_n$ *for multi-valued relation* $R : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q$ *is a don't care, iff* $R(m) = 2^Q$.

Definition 2.2 (Partial Care) *A minterm* $m \in P_1 \times P_2 \times ... \times P_n$ *for multi-valued relation* $R : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q$ *is a partial care, iff* $R(m) \subset 2^Q$.

Don't cares are a special case of partial cares for multi-valued relations. Partial cares can result from observability relations in a logic network, or special requirements given by the designer. Consider an MV-node n_i in a MV-network. There exists a set of minterms, which when applied at the primary input space, allow output values of n_i to be within a subset of the values $v_s \in |x_i|$ for n_i. This set of minterms, when mapped into

the local input space of n_i, can be used to produce any value in the subset v_s for node n_i. This provides additional flexibility in terms of the implementation of n_i, which does not affect the functionality of the network. In the hardware implementation, a function has to be deterministic and produce a single value for each input minterm. Therefore, the synthesis process needs to explore the flexibility given by partial cares and produce a deterministic function satisfying some optimality criteria. If the target application is software, however, the functionality of an MV-node need not be determinized for the purpose of output evaluation.

Definition 2.3 (Compatible) *A multi-valued function* $f : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q$ *is compatible with an multi-valued relation* $R : P_1 \times P_2 \times ... \times P_n \rightarrow 2^Q$ *if* $\forall m \in P_1 \times P_2 \times ... \times P_n$, $f(m) \in R(m)$.

Given an multi-valued relation R with the set of don't cares and partial cares, the minimization problem for hardware implementation consists of two steps: (1) find a multi-valued function f that is compatible with R; (2) find an optimal implementation for f, in terms of the number of product terms and/or the number of multi-valued literals. For instance, the example given in Figure 2.1 can be optimized into the following function f_1:

f_1 :	$\{0,1\}\times$	$\{0,1,2\}\times$	$\{0,1,2\}$	$\rightarrow$	$\{0,1,2,3\}$
	1	–	–	$\rightarrow$	0
	0	–	–	$\rightarrow$	1

2.2 Partial care minimization

The exact minimization problem can be mapped into a binate covering problem directly. Or, if the multi-valued output variable is encoded into binary variables, the multi-valued relation becomes a Boolean relation. Exact and heuristic algorithms for Boolean relation minimization [BS89d] [BS89a] [LS90] [WB91a] [WB91b] [WB93b] [GDN92], etc., can thus be used to minimize multi-valued relations. However, the objective function for this minimization may not be the one desired for MV-logic.

2.2.1 Branch and Bound

Here we propose a simple branch and bound algorithm for minimizing a multi-valued relation using partial cares.

```
Algorithm [Partial care minimization]:
input: characteristic function P × Q → B
output: P × Q → B with minimal number of terms
local CARE: care set minterms
local PART: partial care set minterms
local DONT: don't care set minterms

Foreach pair of product terms u and v
        If u[P] ≡ v[P] then
                u := u|v; remove v;
Foreach product term u
        If |u[Q]| ≡ 1 then push(CARE, u);
        If |u[Q]| > 1 then push(PART, u);
Foreach product term u
        v = u[P] × {all ONE's};
        push(DONT, v);
DONT = ¬DONT;

partial-branch-and-bound(CARE, PART, DONT);
End



partial-branch-and-bound (CARE, PART, DONT) :
Select next partial term u ∈ PART;
Foreach output value q ∈ u[Q];
        push(CARE, u[P] × {q});
        push(DONT, u[P] × {u[Q] \ q} );
        cost = ESPRESSO(P × Q → B);
        If (cost < lbound) then
                lbound = cost;
                partial-branch-and-bound (CARE, PART, DONT);
        else
                unbound(PART, u, q);
Return
```

Figure 2.2: Branch and bound-based partial care minimization algorithm

We use the characteristic function of the MV-relation as the media for minimization. The minterms are partitioned into three sets: *care set*, *partial care set* and *dont care set*. Care set minterms are those completely specified minterms; dont care set minterms are unspecified minterms ; partial care minterms are minterms that are specified to be mapped to multiple output values. The algorithm iterates through all partial care terms and branches on the output values in Q for each partial care term and bounds on the number of product terms as a result from ESPRESSO. This process is shown in Figure 2.2.

2.3 Application to protocol synthesis

Two-level multi-valued logic minimization can be applied to the state encoding problem for finite state machines, as described in the book [VKBSV97]. In this section we propose an application to protocol synthesis as an instance of a control synthesis problem.

2.3.1 Protocol converter synthesis as a control synthesis problem

The protocol converter synthesis problem is the automatic synthesis problem of a finite state converter between two incompatible communicating protocols. This problem emerges from the need for system composition of reusable IP blocks, which function using different communication protocols. We study the protocol conversion problem in the context of synchronous hardware. In particular, we view the problem as an instance of the classical synchronous controller synthesis problem:

Given: a module S and a specification module R.
Find: a controller C such that the composed system S||C refines R.

The general controller synthesis problem is known to be computationally intractable. We restrict our attention to a subclass of the problem: neither P nor S have private states. The problem is then reduced to transition invariant control synthesis with complete information. The complexity is linear in the size of the state space of P and the size of the description of S. Furthermore, by imposing these restrictions, symbolic methods can be employed.

Multi-valued logic synthesis is used to minimize the textual description of the generated controller. We distinguish three kinds of minimum controllers: minimum most

general controller, minimum deterministic controller, and the minimum controller containing a subset of behaviors. All three kinds of controllers can be obtained by applying two-level multi-valued logic synthesis algorithms. Partial cares are utilized for minimizing controllers of the latter two kinds. We apply the following simplifying assumptions to our scheme:

- At the functional level, the information being exchanged from protocols P and Q is a bit stream;
- P and Q are synchronous and share a common clock;
- Bits are sent by an abstract message producer and received by an abstract consumer;
- The composition of protocol P and Q with the converter C, $P \parallel C \parallel Q$, exhibits a **FIFO** buffer behavior at the top level;
- Invariant R, the **FIFO** buffer, is modeled without private variables;
- Protocols P and Q are modeled without private variables.

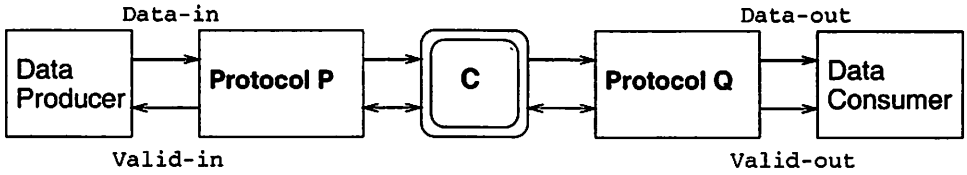

Figure 2.3: Protocol conversion problem

We view a protocol as a mechanism for transferring bit streams between an abstract message producer and an abstract message consumer. As shown in Figure 2.3, we require the producer to produce one bit of data in $data_{in}$ whenever $valid_{in}$ is asserted; the consumer absorbs one bit of data whenever $valid_{out}$ is asserted. We currently require that the communication behave like a **FIFO** buffer, which reads $data_{in}$ and controls $valid_{in}$, $valid_{out}$ and $data_{out}$. The protocol conversion problem can therefore be formulated as:

Given: specification for protocol P and protocol Q; specification for a size-k buffer K;
Find: a converter C such that the composed system P||C||Q refines K.

This can be reduced to a control synthesis problem by setting

$$\begin{aligned} \mathsf{S} &= \mathsf{P}||\mathsf{Q} \\ \mathsf{R} &= \mathsf{K} \end{aligned}$$

The resulting controller C from control synthesis is the converter C for P and Q in the protocol conversion problem. The algorithm used for transition invariant control synthesis can be divided into 3 steps:

1. Find the maximum controllable region and the corresponding controller C transitions;
2. Find the maximum controllable initial region;
3. Find the reachable region of the maximum controllable region and the corresponding controller C transition;

2.3.2 Protocol minimization

We use reactive modules [AH96] to model the communication system. Reactive modules are a high level modeling language that has been used in formal verification. A system is modeled as a set of modules. A module P has a finite set of typed variables, denoted by X_P. A state of P is a valuation for the variables in X_P. X_P is partitioned into two sets: the set of *controlled variables* $\mathrm{ctr}X_P$ containing the variables whose values are determined by the module P, and the set of *external variables* $\mathrm{ext}|X_P$ containing the variables whose values are determined by the environment of P.

A module is composed of atoms, each of which controls a subset of the controlled variables. Within each atom, the initial values and update values of the variables it controls are specified using initial and update guarded commands. A guarded command has the form $p \rightarrow \gamma$, where the guard p is a Boolean predicate over primed and unprimed variables, and γ is an assignment specifying the values assigned to all the controlled variables in the atom when the guard is true.

The control synthesis result is the assignment strategy for controlled variables at each controllable state. We minimize the size, namely the number of guarded commands, in the resulting reactive module. The reasons are: (1) the size of the reactive module directly relate to the implementation cost, either in hardware or software; (2) smaller controllers are more easily understood by protocol designers who may want to add further restrictions

Reactive Module	Multi-valued logic
Guarded commands	Sum of product terms
External variables	Primary inputs
Interface variables	Primary outputs
Atoms	Intermediate nodes
Non-determinism	Partial cares

Table 2.1: Reactive modules and multi-valued logic

or requirements to the specification. Performance issues like delay or buffer size are not addressed in the minimization. They are expressed in the specification module and the controller satisfies those requirements automatically by construction.

Propositional reactive modules naturally resemble multi-valued sequential logic networks, with state variables representing latched wires and atoms representing multi-valued intermediate nodes in logic networks. Each variable in a reactive module can be treated as a multi-valued variable in multi-valued logic. The following mapping is used in translating reactive modules into multi-valued logic.

The resulting controller is mapped to a two-level multi-valued logic circuit represented by characteristic functions. The controllable states, i.e. an evaluation of observable variables to the controller, are treated as the input part of the multi-valued function; the updating strategies, i.e. assignments to controlled variables, are treated as the output part of the multi-valued function. This is a multi-valued relation, i.e. for each minterm in the input space there exists a set of minterms in the output space. The reason is that for each controllable state there exists multiple updating strategies. For ease of minimization, the characteristic function is further encoded into a multi-valued relation with only one output variable that takes multiple values. The resulting relation is expressed in characteristic function form for minimization.

The main objective in the minimization is the textual size of the reactive modules. Three types of minimization approaches are used, which result in three 'smallest' controllers:

- *Smallest most general controller:* The characteristic function of the multi-valued relation is minimized directly with ESPRESSO [RSV88]. This approach keeps all non-deterministic behavior of the controller.

- *Smallest controller that contains a subset behaviors:* This is a Boolean relation min-

Examples	Number of variables
size-1 buffer	6
size-4 buffer	11
phase 2	5 || 5
phase 4	5 || 5
PCI	6 || 6

Table 2.2: Example protocols and FIFO buffer

Protocol conversion	Before	Espresso	Partial	Determinized
phase 2 Master || phase 4 slave	612	20	16	18
PCI Master || phase 4 slave	289	10	10	10
PCI Master || phase 2 slave	421	32	26	26

Table 2.3: Protocol synthesis results

imization problem where non-determinicity is preserved. We apply the partial care-based MV-relation minimization algorithm in Figure 2.2 for this task.

- *Determinized controller:* This is a Boolean relation minimization problem where a deterministic MV-function is to be found. Recent research of AURA by Luca Carloni *et al.* that uses negative thinking scheme can be applied in this framework. Here we applied a modified version of the algorithm in Figure 2.2.

2.3.3 Experiments

The control synthesis algorithm is implemented in Mocha[AHM+98]. Multi-valued logic synthesis algorithms are implemented in MVSIS[Be99]. Some of the characteristics of the protocol examples we have used are shown in Table 2.2. The characteristics for the converters we have generated are shown in Table 2.3. The numbers in the table are numbers of sum of product terms, which relate to the number of guarded commands.

Protocol Formalization: From the perfect knowledge assumption, the converter C generated in our process can observe all variables that P and Q are in contact with. However in the real world, converter C can not see $valid_{in}$, $data_{in}$, $valid_{out}$, and $data_{out}$. To resolve $data_{in}$, we require P to send out data in the same cycle that it receives one. Suppose $data_P$

is the data signal that P uses to send data to the converter C, then $data_P \equiv data_{in}$ must always be true. Then within the converter C we can replace any occurrence of $data_{in}$ with $data_P$. Similarly, we can resolve $data_{out}$ by requiring $data_{out} \equiv data_Q$, where $data_Q$ is the data signal that Q receives data from the converter. Generally no similar technique exists for $valid_{in}$ and $valid_{out}$ because otherwise P and Q would just be space-less FIFOs with unnecessary signals. Hence, to resolve this we need to relax the perfect knowledge assumption and use another controller algorithm. This may result in additional computational complexity.

FIFO Formalization In formalizing the FIFO, the main problem is separating the FIFO specification from the actual converter implementation. When deciding the controlled variables for the converter C, we incorporate all variables from the environment including variables from FIFO. Thus, the variables that the converter controls are dependent on how the FIFO is specified. Because of this, the sequential dependency order for the variables in the FIFO specification is crucial. If dependency is such that it is impossible to be achieved by P and C, then no converter will be generated. In the case between 2 phase to 4 phase protocols for a Moore controller, we used a FIFO module specification in which signal s has to be updated in the same cycle as signal $valid_{in}$. These two signal are controlled by P and converter C separately. As a result, it is impossible for the Moore converter C to update signal s in the same cycle as P updates its signal $valid_{in}$. Currently, we break down the FIFO into 4 different atoms and specify each atom as a Moore machine whenever possible.

Aside from safety constraints, we have attempted to experiment with liveness constraints. If we allow the buffer to stutter for an arbitrary number of cycles, the resulting most general converter may contain a behavior that does nothing. This does not satisfy liveness constraints. We introduce liveness by disallowing the buffer to stutter. Thus the converter is guaranteed to promote data transfer by following exactly the cycle behavior specified in the FIFO buffer. However, this additional liveness constraint will trim down both useless and some useful controller.

In the minimization process, it is promising to combine the minimization algorithms with reachability analysis. Whenever a decision is made and some of the transitions are pruned, some originally reachable states may become unreachable. This can result in more improvement in the minimization performance.

Chapter 3

Multi-valued logic network minimization

As pointed out in Chapter 1, multi-level multi-valued logic networks have various applications. In this chapter, we describe algorithms to minimize such networks, which explore the flexibility provided by logic observability in the network. We extend previous work of observability don't cares in the binary domain and give some experimental results for the proposed algorithms.

3.1 Preliminary

A multi-valued combinational logic network, or *MV-network*, is a network of nodes. Each node represents a multi-valued function, or *MV-function*, with a single multi-valued output and multi-valued inputs. There is an directed edge from node i to node j, if the function at node j explicitly depends on the output variable at node i. The multi-valued intermediate variables are called MV-variables.

The optimization problem of such a MV-network is to find an optimal partition and interconnection of MV-nodes, each implementing a MV-function. The optimizing criterion is a function of the total number of nodes and *size* of the MV-function contained in each node. This cost function will depend on the final target of implementation. Because of the large solution space, finding the optimal solution is more than NP-hard. We are aiming for a set of heuristic minimization methods, each focusing on a particular aspect of the implementation. The assembly of these methods form optimizing *scripts*, which can heuristically lead to a

good solution. Like in SIS [SSL+92], these methods include algebraic factorization and decomposition, collapsing, resubstitution, don't care based minimization, node grouping and splitting, variable pairing and encoding, *et al.* We have built up a software framework called MVSIS to carry out these optimization tasks.

3.2 Observability in multi-valued logic networks

Algebraic methods [Bra99] [GB00] can be used to derive an appropriate structure for the MV-network. Once the structure has been decided, the multi-valued function at each node can be optimized according to the maximal permissible behavior allowed for this node. The flexibility is given by satisfiability don't cares (SDC), observability don't cares (ODC) and observability partial cares (OPC). For the definition and application of SDC and ODC refer to [Sav92]. Observability Boolean relations have been studied for Boolean networks, e.g. [Sav92] [SB90] [SBT91] [DM90] [WGB96], but the computational intensity has prevented the methods from being applicable for large circuits. Multi-valued observability partial cares are a generalization to binary observability relations. This section addresses the usefulness of multi-valued observability don't cares and partial cares, and the methods to generate them.

3.2.1 Maximal set of observability don't cares

Let y_i be the output variable, and $\{x_1, ..., x_r\}$ be the input variables of node i. Let $y_i \in \{0, ..., n\}$, and $x_j \in \{0, ..., t_j\}$. The MODC for the input edge x_j, is the set of minterms in the primary input space, such that the output MV-function y_i is insensitive to all values of x_j. This set of minterms can be used as don't cares for the minimization of the MV-function x_j, since they have no effect on the output value of y_i. We first compute the set of don't care minterms $MODC^l$ in the local input space of y_i, under which the values of x_j are indistinguishable. This gives the maximal ODC for edge $x_j \rightarrow y_i$. $MODC^l$ can be defined as follows:

$$MODC^l(y_i, x_j) = \{m | f(m[x_j = 0]) = ... = f(m[x_j = t_j]), m \in P_1 \times ... \times P_r\} \quad (3.1)$$

The form $m[x_j = k], k \in \{0, \ldots, t_j\}$, means partially setting the value of x_j in minterm m to k. The value of y_i does not change, if we arbitrarily flip the value of x_j

within the range $\{0,\ldots,t_j\}$ and keep the other parts of minterm m fixed. This gives the condition that the function produced by x_j is totally blocked by the other inputs and can not be observed at y_i. $x_j \rightarrow y_i$ becomes a redundant wire for this set of minterms. $MODC^l$ can be computed using multi-valued cofactoring:

$$\begin{aligned} MODC^l(y_i,x_j) &= f^0_{x_j^0}\cdot f^0_{x_j^1}\cdots f^0_{x_j^{t_j}} + f^1_{x_j^0}\cdot f^1_{x_j^1}\cdots f^1_{x_j^{t_j}} + \cdots + f^n_{x_j^0}\cdot f^n_{x_j^1}\cdots f^n_{x_j^{t_j}} \\ &= \sum_{l=0}^{n}\prod_{k=0}^{t_j} f^l_{x_j^k} \end{aligned} \tag{3.2}$$

f^l is a binary function, $P_1 \times ... \times P_r \rightarrow B$, which defines the set of minterms in $P_1 \times ... \times P_r$ that produce output l for y_i. Function $f^l_{x_j^k}$ is the cofactor of binary function f^l with respect to literal x_j^k. It is independent of x_j and preserves the onset of f^l whenever $x_j = k$, i.e. $x_j \cdot f^l_{x_j^k} = x_j \cdot f^l$. Function $f^l_{x_j^k}\cdot f^l_{x_j^1}\cdots f^l_{x_j^t}$ defines the set of minterms in $P_1 \times ... \times P_r$ such that the output value for y_i is always l no matter what value x_j takes, i.e., the universal quantification over the values of x_j. Formula (3.2) represents the complement of the Boolean difference $(\partial f/\partial x_j)$ in the multi-valued sense.

Theorem 3.1 (MODC) *The binary function (3.2) computes the set of MODC minterms for input edge x_j, in the local inputs space of y_i as defined in (3.1).*

Proof. (Sufficiency) Let minterm m be in the function computed by (3.2). There exist l, such that $m \in f^l_{x_j^0}\cdot f^l_{x_j^1}\cdots f^l_{x_j^t}$. Hence, $m \in f^l_{x_j^k}$ and $f(m[x_j = k]) = l$ for all $k = 0,1,\ldots,t$. (Necessity) Let minterm m be in the set defined in (3.1). There exist l, such that $f(m[x_j = k]) = l$ for all $k = 0,1,\ldots,t$, i.e. arbitrarily toggle the value of x_j in m does not change the value of $f(m)$, which is 1. Therefore, $m \in f^l_{x_j^0}\cdot f^l_{x_j^1}\cdots f^l_{x_j^t}$.

However, this set of don't cares may not be valid if the functions of other nodes, $\{x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_r\}$ are changed, in which case CODC's have to be used instead.

3.2.2 Compatible set of observability don't cares

The validity of MODC's for a particular input edge requires other input edges to produce certain values. There may be cyclic dependencies in this relationship, thus causing incompatibility. Consider node i, with input edges: $x_1,\ldots,x_j,\ldots,x_l,\ldots,x_r$. Let $x_j \in \{0,\ldots,t_j\}$, and $x_l \in \{0,\ldots,t_l\}$. Let $MODC_j$ and $MODC_l$ be the maximal set of

observability don't cares for the input edges x_j and x_l respectively. Let $m_q \in P_1 \times ... \times P_r$, be a minterm in the local input space of y_i, such that $m_q \in MODC_j$ and $m_q \in MODC_l$. The primary input minterms that produces m_q will be used as don't cares for both x_j and x_l. The optimization of x_j as a result of m_q may destroy the validity of $MODC_l$; and the optimization of x_l as a result of m_q may also destroy the validity of $MODC_j$. Therefore the minterm in $MODC_j$, for the input edge x_j, is said to be compatible with $x_l, l \neq j$, if it is not a minterm in $MODC_l$, or if $MODC_j$ does not depend on the value of x_l, i.e.

$$MODC_j^{C_l} = \{m | (m \notin MODC_l) \vee (\forall x_l(m) \in MODC_j), m \in MODC_j\}$$

By "$\forall x_l$" we denote the computation of universal quantification over all values of x_l, the result of which is a multi-valued cube. The interpretation of this relation is that: of all the minterms, $m \in MODC(f, x_j)$, where f is insensitive to x_j, m is said to be compatible with another input edge x_l, if (1) either m is not a don't care for x_l, (2) or no matter what value is chosen for x_l, f is still insensitive to x_j under m.

From a global optimization point of view, there may exist an optimal way of distributing the set of observability don't care minterms to the input edges, but the complexity of finding this is too much. Therefore we take a greedy approach, where the input edges are implicitly ordered and the CODC for each input edge is computed by making the associated MODC compatible with all the preceding edges in the ordering. Given an ordering $x_1 \prec ... \prec x_j \prec ... \prec x_r$, the CODC for edge x_j can be defined as follows:

$$CODC(y_i, x_j) = \{m \in MODC_j | \forall l < j, (m \notin CODC_l) \vee (\text{for any value of } x_l,\ m \in MODC_j) \tag{3.3}$$

This approach gives the first edge the most flexibility. The successive edges are sacrificed in order to be compatible with previous ones. The ODC set for each node is thus reduced for compatibility. However, this approximation gives reasonable results and run time performance. In practice, the set of minterms in MODC can be represented symbolically using MDD's [SKMB90] [KB90]. Also, the CODC set for each node can be inherited by all input edges without modification. The formula thus can be constructed with MDD operations:

$$\begin{aligned} CODC(y_i, x_j) &= P_1(P_2(\cdots P_{j-1}(MODC(y_i, x_j)))) + CODC_{y_i} \\ P_k(F) &= \overline{CODC_{x_k}} \cdot F + \forall x_k \cdot F \\ CODC_{x_j} &= \prod_{i \in fanout(x_j)} CODC(y_i, x_j) \end{aligned} \tag{3.4}$$

$CODC(y_i, x_j)$ is the *edge*-CODC for edge $x_j \rightarrow y_i$. $CODC_{y_i}$ is the computed *node*-CODC for the fanout node y_i, represented in the primary input space. This is passed to all fanin nodes of y_i without modification. $P_k(F)$ is the compatibility operation which is applied to each fanin node of y_i that precedes x_j in the pre-assigned order. $CODC(y_i, x_j)$ is computed for each fanout edge of node x_j and they are intersected to give the CODC at node x_j.

Theorem 3.2 (CODC) *The set of minterms computed by (3.4) are don't cares for node x_j and they remain to be don't cares if the function of any other node changes within their computed CODC's.*

Proof. $CODC^C_{x_j}$ is a set of don't care minterms because it is a subset of $MODC_{x_j}$, which by definition is a don't care for x_j. Let PI be the primary input space for the network. Let relation $g : PI \rightarrow P_1 \times ... \times P_n$ be the mapping from primary input space to the local input space of y_i. The changes of nodes other than $x_1, \ldots, x_r$ do not affect the CODC set of x_j. Assume $x_1, \ldots, x_r$ are the only nodes that are changed after the network optimization. Suppose there exists minterm $m_q \in MODC^C_{x_j}$ that is no longer a don't care for x_j after optimization. The mapping $(m_{PI}, m_q) \in g$ has changed as a result of the minimization of nodes $x_1, \ldots, x_r$, or in other words, m_{PI} has been used as a don't care by node $x_l, l \in \{0, \ldots, x_j, x_{j+1}, \ldots, x_r\}$. This contradicts with (3.4).

3.2.3 Maximal set of observability partial cares

CODC's do not capture all the flexibility for MV-networks. For the input edge x_j of node i, there are such minterms that y_i is insensitive to only a subset of the values for x_j. In other words, minterms under which a subset of the values for x_j is indistinguishable at y_i. These minterms are, by the definition given in Section 2, partial cares for the function implemented at node x_j, and can also be used in the minimization of x_j. The maximal set of partial cares for edge $x_j \rightarrow y_i$ can be defined on the power set, 2^{P_j}, of the values for x_j.

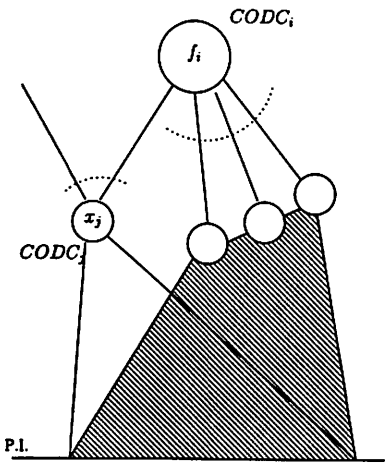

Figure 3.1: Calculation of compatible set of observability don't cares

Let $x_j \in P_j = \{0, \ldots, m\}$ and $v = \{r_1, \ldots, r_v\} \in 2^{P_j}$. For each subset of the values $v_t \in 2^{P_j}$ for x_j, there exists a set of minterms S_t in the local input space, such that the values in the subset v_t can not be distinguished at the output y_i. We can compute the these sets of minterms for each such subset values v_t. When mapped into the primary input space, S_t represents the set of partial care minterms for node x_i, where x_i can take any value in v_t.

$$\begin{aligned} OPC(f, v, x_j) &= \{m | f(m[x_j = r_1]) = \cdots = f(m[x_j = r_v]), m \in P_1 \times \cdots \times P_r\} \\ MOPC(y_i, x_j) &= \{(m, v) | m \in OPC(y_i, v, x_j), \forall v \in 2^{P_j}\} \end{aligned} \tag{3.5}$$

$OPC(f, v, x_j)$ gives the set of minterms in the local input space of f, such that a subset v of the values for x_j are indistinguishable at y_i, i.e. the output function of y_i does not change, if we arbitrarily change the value of x_j within the subset of values $v = \{r_1, \ldots, r_v\}$, while keeping the other parts of m fixed. Therefore, MOPC by definition is a set of pairs, (m, v), where $m \in P_1 \times \cdots \times P_r$, is a minterm in the local input space of y_i, and $v \in \{r_1, \ldots, r_v\}$ is a subset of the values for x_j. MOPC's can be computed by multi-valued cofactoring, similar to (3.1).

$$
\begin{aligned}
& MOPC(y_i, x_j) \\
= & \sum_{v \in 2^{P_j}} \left(v, \left(f^0_{x_j^{r_1}} \cdot f^0_{x_j^{r_2}} \cdots f^0_{x_j^{r_v}} + f^1_{x_j^{r_1}} \cdot f^1_{x_j^{r_2}} \cdots f^1_{x_j^{r_v}} + \cdots + f^n_{x_j^{r_1}} \cdot f^n_{x_j^{r_2}} \cdots f^n_{x_j^{r_v}} \right) \right) \\
= & \sum_{v \in 2^{P_j}} \left(v, \sum_{l=0}^{n} \prod_{k \in v} f^l_{x_j^k} \right) \qquad (3.6)
\end{aligned}
$$

This computation can be expensive due to the power set summation. The OPC needs to be computed for every subset of $|x_j|$, which is exponential. In practice multi-valued variables usually have a reasonably small set of values to choose from, thus suggesting the feasibility of (3.6). However algorithmic trade-offs need to be explored to assess the practicality of MOPC. For instance, one can choose only the subsets of a certain size, or subsets of particular values that are of interest.

3.2.4 Compatible set of observability partial cares

Like observability don't cares, observability partial cares may also become invalid if the functions of related nodes are changed. Similar to the approach used in CODC, we can order the input edges for each node, and make the MOPC set compatible with each preceding edge. However the computation of COPC is much more complicated than CODC because subset values for each input edge have to be considered. The advantages of COPC for MV-network minimization needs to be evaluated.

Consider input edges x_j and x_l. Let $(m_j, v_j) \in MOPC_j$, $(m_l, v_l) \in MOPC_l$. Let $m_q \in P_1 \times \cdots \times P_r$ be a minterm in the input space of y_i, such that, $m_q \in \exists x_j(m_j), m_q \in \exists x_l(m_l)$, and $m_q[l] \in v_l, m_q[j] \in v_j$. Then it is possible that the same minterm in the primary input space PI is used for both x_j and x_l as a partial care, thus destroying the validity of both partial cares. Therefore, $(m_j, v_j) \in MOPC_j$ is said to be compatible with $(m_l, v_l) \in MOPC_l$ if

$$(m_j[l] \notin v_l) \vee (m_l[j] \notin v_j) \vee (\forall x_l(m_j) \notin m_l) \vee (\forall x_j(m_l) \notin m_j)$$

The following computation can be applied to make MOPC's independent with previous edges in the ordering, although some optimality is given up.

$$
\begin{aligned}
COPC(y_i, x_j) &= P_1(P_2(\cdots P_{j-1}(MOPC(f, x_j)))) + COPC_f \\
P_k(F) &= \sum_{\forall (m_j, v_j) \in MOPC_j} \left(\bigcup_{\forall v_k \in PP_k} (m_j[k] \notin v_k) + \bigcup_{\forall (m_k, v_k) \in MOPC_k} (\forall x_k(m_k) \notin m_j) \right) \\
COPC_{x_j} &= \prod_{i \in fanout(x_j)} COPC(y_i, x_j)
\end{aligned} \tag{3.7}
$$

Proposition 3.1 (correctness of COPC) *The set of minterms computed by (3.7) are partial cares for node x_j, and they remain to be partial cares if the function of any other node changes.*

This computation is expensive because one has to compute the set of partial care minterms for each subset of the multi-value range, which is exponential. The complexity of the compatibility computation is the square of the average number of subsets. Therefore some heuristics need to be applied.

3.3 Don't care-based logic network minimization

We give an algorithm to compute a set of CODC's for each node in a multi-level MV-network. The algorithm is an extension of the binary CODC computation from [Sav92]. The computation of MODC and CODC is implemented in MV-SIS [Be99]. MOPC and COPC computations are expensive and require a multi-valued relation minimizer for the optimization of each node; it is not implemented in the current version.

3.3.1 MV-network optimization

All computation is carried out using MDD operations [KB90]. A heuristic depth-first search MDD variable ordering is implemented. The logic function of each node in the network is represented by a multi-valued *table* structure, as defined in VIS [VIS96]. A *table* is a sum of product representation of a multi-valued function. Each row is partitioned into an input part and an output part, and represents a multi-valued cube.

The algorithm traverses the MV-network in a reverse topological order from primary outputs to primary inputs. Each node in the network is traversed once. At each node, the $CODC$ set is computed for each fanout edge, and then intersected to give the

Algorithm [CODC-based MV-network minimization]:
input: MV-network ntk
input: external don't care XDC_j at each primary output j
local $CODC_i$: CODC set for node i
local DC_i: complete don't care set for node i

Traverse each node j in ntk in reverse DFS order
    If j is primary output
        $CODC_j =$ external don't cares (XDC_j)
        Continue;
    For each fanout node k
        D = MODC(f_k, y_j) computed using MDD universal quantification;
        For each fanin node i of k that is already visited
            $D = [\overline{CODC_i} + \forall y_i] \cdot D$
        $D = D + CODC_k$;
        $CODC_j = CODC_j \cap D$;

    Collapse $CODC_j$ into primary input space
    Remove the supporting variables not in the transitive fanin cone of y_j
    $DC_j = \neg$image($\neg CODC_j$)
    MINIMIZE($ONSET_j, DC_j$)
End

Figure 3.2: CODC-based MV-network minmization algorithm

approximated $CODC$ set for this node. Once calculated, the $CODC$ set for each node is inherited by each of its own fanin nodes. The $CODC$ set is mapped into the primary input space by variable substitution, and then mapped into the local input space by image computation. Given the DC set, the logic minimization of a multi-valued node is carried out using ESPRESSO-MV [RSV88].

3.3.2 Multi-valued image computation

Two methods exist for image computation, namely transition relation and recursive range computation. We extend the recursive range computation from binary domain to multi-valued domain. For binary image computation, the readers are referred to [Sav92]. Multi-valued cofactoring is used to reduce the computation in a recursive fashion.

In the local input space of node y_i, each input variable is cofactored by the complement of the don't care set $A(x)$, which is an MDD in terms of primary input variables. This array of cofactored functions give the transition functions that map the entire primary input space PI into the local care set of node y_i.

$$\begin{aligned} F_{A(x)} &= [(f_1)_{A(x)}, (f_2)_{A(x)}, \cdots, (f_r)_{A(x)}] \\ A(x) &= \overline{CODC^{y_i}_{PI}}(x) \end{aligned}$$

Each f_k is the multi-valued function for one of the fanins, x_k, of y_i. Each f_k is represented by an array of MDD's; each MDD represents the onset for one of the values of f_k. The cofactor $(f_k)_{A(x)}$ is obtained by *constraining* the MDD function for each value of f_k against $A(x)$. This is called the *constrain* computation. Once we have the range function, we apply output cofactoring to carry out the recursive image computation:

$$\begin{aligned} & \overline{CODC^{y_i}_{local}} \\ = \; & IMAGE(\overline{CODC^{y_i}_{PI}}) = RANGE(F_{A(x)}) = RANGE(f_1, f_2, \cdots, f_r) \\ = \; & RANGE([0, f_2, \cdots, f_r]_{f_1^0}) + \cdots + RANGE([|P_1|, f_2, \cdots, f_r]_{f_1^{|P_1|}}) \\ = \; & y_1^0 \cdot RANGE([f_2, \cdots, f_r]_{f_1^0}) + \cdots + y_1^{|P_1|} \cdot RANGE([f_2, \cdots, f_r]_{f_1^{|P_1|}}) \\ = \; & \sum_{k=0}^{|P_1|} y_1^k \cdot RANGE([f_2, \cdots, f_r]_{f_1^k}) \end{aligned}$$

In the above formula, f_k^0 denotes the MDD function that evaluates to the 0^{th} value of f_k, and y_i^0 denotes the literal for intermediate variable y_i that takes the 0^{th} value. The range computation is recursively applied to the list of functions, until every one has been cofactored. The final result is a set of cubes in the local input space of y_i, which can then be used in the minimization of node y_i.

3.4 Multi-level logic network encoding

In order evaluate how much value is added by multi-valued optimization in the hardware synthesis flow, we need to encode all MV-variables into binary codes and apply binary optimization. A trivial encoding will obscure multi-valued optimization efforts. This is an input-output encoding problem with multi-level implementation, which does not have an established theory yet. Previous literature [VKBSV97] [VSV90] on the encoding problem has been divided into four categories: input encoding, output encoding, input-output encoding and state encoding. Each category has a two-level implementation and a multi-level implementation version. Research on the multi-level encoding problem has focused on state encoding, where there is only one multi-valued variable appearing at both the primary input and primary output. The theory of the input-output encoding problem for two-level implementation has been well established and efficient software packages exist [VSV90].

The encoding problem we are facing here is formulated as follows:

Given: a multi-level MV-network, ntk, with a set of multi-valued primary inputs PI, a set of multi-valued primary outputs PO and a set of multi-valued intermediate nodes $NODE$;
Find: a binary code, bin, for each MV-variable in PI, PO and $NODE$, such that the following is minimized:

1. $\sum_{ntk} cost(cubes, literals)$
2. $\sum_{i \in PI} |bin_i| + \sum_{i \in PO} |bin_i| + \sum_{i \in NODE} |bin_i|$

The goal is to minimize the total number of cubes and literals in the network, using a number of binary codes as small as possible. We give an algorithm that combines binary encoding with observability don't care computation. The algorithm applies two-level input-output encoding for each MV-node using methods given in [VSV90] [VKBSV97].

Algorithm [MV-network encoding]:
input: MV-network ntk
input: external don't care XDC_j at each primary output j
local $CODC_i$: CODC set for node i
local BIN_i: binary code for the output of node i
local Con_i: face constraints for BIN_i

Traverse each node j in ntk in reverse DFS order
    If j is primary output
        $CODC_j$ = external don't cares (XDC_j)
        MINIMIZE($ONSET_j, CODC_j$)
        $BIN_j \leftarrow$ Output encoding using NOVA
        Create face constraints for each input MV-variable
        Continue;
    For each fanout node k
        $CODC_j = CODC_j \cap CODC(j,k)$;
        $Con_k \leftarrow$ face constraint from node k
        $Con_j = Con_j \cup Con_k$
    $BIN_j \leftarrow$ CONSTRAINT-SAT(Con_j)

    Distribute $CODC_j$ to BIN_j
    $DC_j(BIN_j) = \overline{\text{image}(\overline{CODC_j(BIN_j)})}$
    MINIMIZE($ONSET_j, DC_j(BIN_j)$)
    If $j \notin PI$
        Create face constraints for each input MV-variable
End

Figure 3.3: MV-network binary encoding algorithm

The algorithm traverses the MV-network from primary outputs to primary inputs. At the primary outputs, external don't cares are used to minimize the MV-function; NOVA is called to encode the output variable and generate face constraints for each MV-input. At an intermediate node, the face constraints are collected from each fanout node, and a constraint satisfaction routine from NOVA is called to trade off the number of bits used and the number of constraints to be satisfied. Once the code for this node is decided, the MV-CODC is computed using the algorithm described in Figure 3.2; the MV-CODC is then distributed to each binary bit. ESPRESSO-MV is then called to minimize the multi-binary-output function after encoding. The minimization result generates face constraints for each MV-input in turn. This procedure stops when all MV-nodes have been traversed. A more aggressive approach is to generate the CODC set for each binary wire after the face constraint satisfaction rather than generating MV-CODC all at once. This would give more flexibility to each binary wire. The trade off between these two approaches is to be explored in the near future. We do not have an implementation for this yet.

3.5 Experimental results

example	PI.	PO.	nodes
sort	8	8	32
matmul	8	4	12
max	8	1	15
test1	14	14	56
test2	40	40	120
test3	40	40	180
abp.sender	6	5	245
abp.receiver	6	5	175
elevator	13	10	645
bakery.proc	12	6	267
slider.nsf	9	9	325

Table 3.1: Characteristics of the examples

Some experiments are performed on a number of real and artificial MV-networks examples. Table 3.1 shows the characteristics of the examples that we tried. *PI* is the number of primary inputs; *PO* is the number of primary outputs; *nodes* is the number of MV-nodes contained in the network. In the first set of examples, *Sort*, *matmul* and *max*

are small made-up examples; *test1*, *test2* and *test3* are randomly generated MV-networks. The second set of examples come from the multi-valued benchmark set distributed with VIS [VIS96]. We extract individual combinational modules and remove the latches to obtain associated MV-networks. For example, *4-arbit.cell* is the module *cell* in benchmark *4-arbit.* As can be seen, the VIS examples have a large number of nodes as a result of human specification. We currently do not have an algebraic package to collapse small nodes into larger ones.

	#cubes			*#literals*		
example	original	mv-simp	mv-fullsimp	original	mv-simp	mv-fullsimp
sort	432	234	234	2612	1518	1358
matmul	160	140	140	820	616	616
max	98	105	105	1414	301	301
test1	416	375	375	2645	2399	2397
test2	785	740	739	5096	4840	4829
test3	1617	1497	1468	10770	10015	9772
total	3508	3091	3061	23357	19689	19273

Table 3.2: Experiments on MV-networks

The CODC-based MV-network minimization algorithm is implemented in MVSIS as command mv-fullsimp. As a comparison, we implemented command mv-simp, which calls ESPRESSO-MV for each node directly without computing MV-CODC's. Table 3.2 and 3.3 shows the total number of multi-valued cubes and multi-valued literals in the MV-network. The multi-columns *#cubes* and *#literal* show the number of multi-valued cubes and literals in the MV-network. Column *original* shows the number of specified term in the original network; column *mv-simp* shows the number after applying command mv-simp; column *mv-fullsimp* shows the number after applying command mv-fullsimp. The results show that the algorithms presented above are able to reduce the product term representation for each MV-node. This translates into implementation cost savings whether in software or in hardware. As can be seen, the first set of examples provides little room for observability don't care optimization, partly because of the small number of nodes in the network.

The experiments are performed on an Intel 500MHz machine with 128MB memory. The run time ranges from 1-10 minutes depending on the size of the examples. There is about a 40% average reduction in both cube count and literal count. The reason is that when the multi-valued logic functions are initially specified using *Blif-mv*, they are designed for

humans to understand, not for efficient implementation. At the time of this implementation, multi-valued algebraic methods are not yet applied. If we combine a number of algebraic methods and don't care minimization methods together in a script, like what has been done in SIS, we expect much more powerful improvements.

	#cubes			*#literals*		
example	original	mv-simp	mv-fullsimp	original	mv-simp	mv-fullsimp
abp.sender	821	821	386	2536	2536	1313
abp.receiver	509	509	177	1597	1572	643
elevator	639	558	316	1508	1294	912
bakery.proc	886	868	168	2753	2558	505
slider.nsf	1251	1251	852	3591	3591	2393
total	4106	4007	1899	11,985	11,551	5766

Table 3.3: Experiments on hardware examples from VIS

Chapter 4

Multi-valued logic for embedded compilation

Since hardware implementation eventually has to resort to binary logic, multi-level multi-valued logic networks have been primarily a topic of academic research. However, there are various application domains other than hardware synthesis that can benefit from the powerful optimization techniques for multi-valued logic networks. In this chapter, we describe some of the research we have conducted in this area.

4.1 Compiler optimization for embedded systems

An embedded system is an electronic component of a larger physical system, which works in a reactive and time-constrained environment. The implementation of an embedded system ranges from full hardware like ASIC and configurable logic, to pure software running on a standard micro-processor. Hardware implementation is deployed for performance requirements; software implementation is used for providing features and flexibility. As technology developments advance, with the capability of integrating one billion transistors in a single die [ntr99], embedded system-on-a-chip is becoming a reality.

The advances of embedded micro-processor architecture and IC technology offer software implementations with competing performance and costs. The increasing importance of time-to-market considerations gradually lead to engineering designs with a larger share of software components. As a result, software synthesis is gaining researchers' attention[Lee99]. As pointed out in [BCG+99] and [ELLSV97], *software synthesis* is distin-

guished from *software compilation* according to the type of input specification. The term *software compilation* is associated with machine code generation for C-like imperative, usually non-concurrent languages; *software synthesis* is used to denote the translation from a high-level specification that describes the function rather than implementation.

Here we propose a software compiler optimization scheme which utilizes advanced multi-valued logic network minimization techniques. The target so far has been the optimization of a software implementation rather than synthesis from a higher level description. We will see that even at this level, there is still plenty of opportunity for optimization if the time of compilation can be relaxed, which is true for embedded applications. This section describes how we model software using multi-valued logic and the algorithms we use for the optimization.

4.2 Control dominated software optimization

We are targeting control dominated software applications for embedded systems. As described by Lee [Lee99], hardware is composed of functional units that operate in parallel and interact via synchronous or asynchronous communication; software is an assembly of components that trade off the use of a central processor and communicate by leaving traces of their execution on a stack or in memory. The intrinsic characteristics decide that hardware is good at pure complex computations and software is good at control paths.

In the design space exploration of embedded system architectures, it is critical to decide which components are to be implemented in hardware and which in software. Tensilica's [XTE] approach of building customized instructions and architectures for a specified domain of applications, is essentially putting data computation into hardware in the form of complex instructions, while leaving control paths within software by implementing simple RISC instructions. Therefore a lion's share of software optimization opportunities lies in the control part. This is the area where multi-valued logic minimization can be beneficial in order to explore bit-level and instruction-level parallelism.

As shown in Figure 4.1, after the front-end lexical analysis and parsing, the proposed optimizing compiler first decomposes the intermediate representation (IR) of the program into a data evaluation part and a control part. The control part is then translated into an MV-network representation, which is optimized using both algebraic and don't care based algorithms. The optimizing criteria are configured to reflect the cost of the final

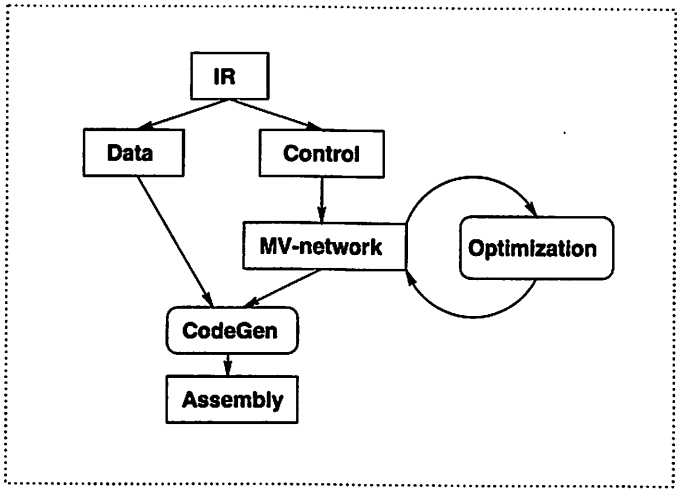

Figure 4.1: MV-network based software optimization flow

software implementation, accounting for both speed and code size. The code generation step combines the data evaluation and control part and produces assembly code. Since we are not exploring the architectural design space, we target our compiler for the series of embedded processors from ARM [ARM].

There have been research attempts to apply logic optimization techniques to software compilation or software synthesis. POLIS [BCG+97] mainly uses a BDD-based approach. There, communicating finite state machines (CFSM) are used to represent the control part once a similar partitioning is done from an high level specification like Esterel [BG92]. The transition relations of the CFSM's are then converted into BDD's, which is in turn optimized based on variable ordering techniques. Finally a set of if-then-else like C code is generated from the BDD's [CGH+95]. There are still optimization opportunities that cannot be discovered by BDD variable ordering, as has been seen in hardware synthesis. We would like to compare the MV-network based approach with POLIS BDD-based approach. In [Edw97] [Edw00], Edwards applies binary logic optimization to software synthesis from synchronous specifications. The results have been very encouraging.

4.3 From code to MV-network

Motivating example: We are still experimenting with methods of transforming

control codes in embedded software into MV-networks for optimization of speed and code size. The following is a simple example of a control dominated code fragment and the corresponding code after optimization. This is one of the optimization results we are trying to achieve through MVSIS.

```
if (a=0) then
  if (b=0) then
    c := 0;
  else
    c := 1;
else if (a=1) then
  if (b=0) then
    c := 2;
  else
    c := 0;
else if (a=2) then
  if (b=0) then
    c := 1;
  else
    c := 2;
```

⇒

```
if (a=0 & b=0)|(a=1 & b=1) then
  c := 0;
else if (a=0 & b=1)|(a=2 & b=0) then
  c := 1;
else if (a=1 & b=0)|(a=2 & b=1) then
  c := 2;
```

Control variables in the original code are converted into multi-valued variables in MV-networks; control structures and conditional branches are converted into MV-functions stored in MV-nodes. Through multi-valued synthesis we explore the bit-level parallelism and possibly some of the instruction level parallelism in the code.

4.4 From MV-networks to code

We have developed methods to convert a MV-network into software code either in primitive C or in assembly code by evaluating and simulating the MV-network. There are two approaches: MDD-based and table-based.

4.4.1 MDD-based

Each MV-node in the network is represented using a multi-valued decision diagram (MDD) [KB90]. The output C code resembles the MDD structure, due to the direct correspondence between MDD nodes and C primitives. Each MDD node is converted to an if-then-else or switch-case struct and go-to statements. The unmodulized code may hinder readability, but allows greater efficiency. This approach is similar to [CGH+95], where an S-graph is used as an intermediate representation of control codes and variable ordering techniques are applied. However, we expect the MDD-based approach to be a generalization of the S-graph and BDD-based one.

4.4.2 Table-based

The function for each MV-node is represented in a sum-of-product form (SOP), and stored in a *table* structure, which is used for representing BLIF-MV [BCH+91] format in VIS [VIS96]. We simulate the function of a node by evaluating the given minterm at the input against each cube contained in the SOP table.

Software codes are generated to simulate the MV-network following the procedure shown in Figure 4.2. The generated code can be C or assembly. C is easy to generate, but the final cost, speed and code size, can not be predicted precisely. We chose assembly for the following reasons: first the core of the simulation code is simple and small, which can be hand crafted efficiently; second the speed and code size can be accurately estimated as a metric for the network minimization. We currently have an implementation for the ARM RISC processor cores[ARM]. It is also relatively easy to retarget for other machines.

A table at a multi-valued node stores a sum-of-products, or a set of cubes, for each output value. One of the output values is default and is omitted. The following example MV-node has 5 inputs with 18 total number of input values and 4 output values, value 3 being the default.

x_1	x_2	x_3	x_4	x_5	z
100	1001	0011	110	1111	0
010	0110	1001	100	1000	0
001	0111	0100	111	0110	1
101	0101	0010	001	1101	2
101	1100	1111	101	0001	2

```
Procedure [Table-based MV-network simulation]:
input: MV-network ntk, input value PI_i at each primary input i
output: output value PO_i at each primary output i
local o_i: output value for node i
local v_i: input vector for node i

Traverse each node i in ntk in DFS order
        If i is primary input then
            o_i = PI_i
        else
            Obtain input vector v_i;
            For each output value k
                    For each cube C contained in the function of value k
                        If v_i ∈ C then
                        o_i = k; break;
        If i is primary output then
            PO_i = o_i
        else
            For each fanout j of i
                    Update v_j with o_i ;
End
```

Figure 4.2: MV-network simulation

The core of the table-based network simulation code is to test if a particular minterm is contained in a cube. This is achieved with a single *AND* instruction, based on the fact that $m \in C \Leftrightarrow m \cdot \overline{C} = \emptyset$. The cubes are stored in memory in complement form so that only one *AND* instruction is need for the testing. We generate the following ARM assembly code for this task:

```
value_start:
   LDR R7, [R6,R4]      ;load the last cube address for this output value
cube_start:
   LDR R2, [R3,#4]!     ;load the next cube (auto-increment)
   TST R2,R1            ;test cube containment
   BEQ found            ;output value found
   CMP R3, R7           ;test the cube address boundary
   BNE cube_start       ;process next cube
   ADD R4,R4, #4        ;increment value index
   CMP R4,R5
   BNE value_start      ;process next value
```

The assumptions made here are: (1) The total number of input values for each table is less than or equal to 32 so that they can be located inside a single register usually of the size 32-bits in common embedded architectures. Given this assumption, the testing of a minterm containment can be completed in a single *AND* instruction. (2) If there are K output values for a table, then the functions for the first $(K-1)$ values are stored in the table, the last being the default. Since the functions for different values are disjoint, this approach results in no sharing between values. We are also investigating an encoding scheme, where the output values are encoded into binary codes and the cubes can be stored more efficiently. However, every cube has to be tested for containment before the conclusion can be made as to which value the output takes. Also since the fanout tables expect a one-hot encoded MV-variable as input, a decoding function is needed before the output variable fans out. Table 4.1 shows a comparison between the one-hot and encoded schemes.

The time it takes to evaluate the whole network is a function of the number of nodes, the number of cubes, the number of values and fanouts for each node. The cost function for the ARM assembly code in terms of the number of instructions for execution

Features	One-hot	Binary
sharing between values	No	Yes
total number of cubes	Large	Small
early decision	Yes	No
decoding overhead	No	Yes

Table 4.1: Comparison between one-hot and binary encoded tables

and the number of instructions for storage is shown as follows (one-hot version):

$$\begin{aligned} \text{Speed} &= 3 \cdot \text{CUBES} + 12 \cdot \text{NODES} + 10 \cdot \text{FANOUT} + 3 \cdot \text{VALUES} + 3 \\ \text{Size} &= 6 \cdot \text{NODES} + 2 \cdot \text{FANOUT} + \text{VALUES} + 37 \end{aligned}$$

The memory organization for the generated assembly code is shown in Figure 4.3. The core of the simulation is a fixed set of assembly instructions, while the data, i.e. cubes for each table, is examined in a data stream. The advantages of this scheme are: (1) the core instructions are hand tuned to be compact and efficient; (2) software pipelining techniques can be applied to the core loops in order to reduce pipeline stall of common pipelined architectures and exploit loop-level parallelism; (3) the core loop code is small enough to be fit inside the instruction cache so that cache misses are reduced. The disadvantage is the loop overhead and unavoidable memory load for each cube.

4.4.3 Experiments

There is a third method for evaluating MV-networks besides the MDD-based and Table-based methods described above, i.e. look-up-table (LUT) based approach. This has been used in FPGA technology mapping. The idea is to partition the network into regions, each to be mapped into a LUT. Constrained by the size of the LUT, each region is limited, and only, by the number of inputs and outputs. The software implementation of a LUT is a constant array, which takes constant time to evaluate for a given input. However, the LUT size, a truth table, is exponential in the number of inputs and outputs. The question is how to trade off the size of the LUTs and the speed of evaluation. Comparing the three approaches of mapping a MV-node into software evaluation code, they are different trade-off schemes for time and space. Given the same structuring of MV-nodes, represented in the three different ways, the evaluation time and the storage space differ accordingly. (1)

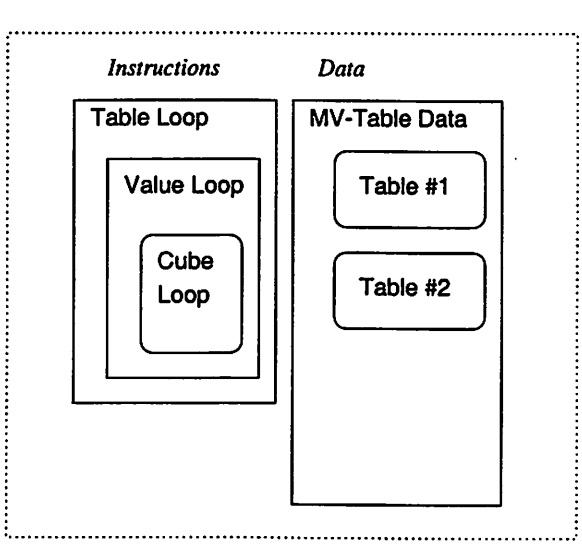

Figure 4.3: Memory organization for table valuation code

MDD representation: the average evaluation time is linear in the number of levels in the MDD, which in worst case equals the number of input variables. The worst case evaluation and average evaluation are about the same. However, the size of the MDD can potentially explode if not treated carefully. (2) *LUT representation*: evaluation time is constant, but memory storage size is exponential in the total number of variables. (3) *Minimized sum-of-product representation* (one-hot output encoding): the average evaluation time is half the number of cubes, while the worst case evaluation time is the total number of cubes. The storage space is far less than exponential if the minimization is effective, but there is no cube sharing between values. (4) *Minimized sum-of-product representation* (binary output encoding): the average and worst case evaluation time equal the number of cubes. The storage space is less than one-hot encoding because of possible sharing among values.

We have implemented the code generation scheme described in the previous section for embedded ARM processors. We use the *Armulator* to simulate the performance and code size of the produced assembly code. In the first set of experiments, we start from a set of MV-network examples instead of pure C files. The characteristics of the MV-examples is shown in Table 3.1.

4.5 Other applications

There are many other interesting applications for multi-valued logic optimization that are being studied by groups of people from both industry and academia. The more traditional applications are in fuzzy control system designs and machine learning. More recently, multi-valued logic synthesis is being applied to asynchronous designs by researchers from Theseus Logic [THE]. In [LFS+], they consider a particular subclass of asynchronous circuits, Null Convention Logic (NCL) [FB96], and suggests a design flow that is completely based on commercial CAD tools. There, multi-valued logic network optimization techniques are applied to synthesize a delay-insensitive combinational circuit.

As the synthesis techniques are getting more mature, we believe that many more applications will be found for multi-valued logic.

Chapter 5

Conclusions and Future Directions

In this chapter we summarize our contributions and point out some future directions where this research can proceed.

5.1 Conclusion

We have investigated multi-valued logic minimization problem in both two-level implementation and multi-level implementation. As a design effort we implemented a software test-bed called MVSIS, which is an assembly of optimizing packages like SIS. We have studied the application of multi-valued logic minimization in software compilation for embedded applications. We summarized our contributions below:

- We generalized the notion of don't cares used in binary logic to multi-valued logic, which incorporates both incomplete specification and nondeterminism. We defined partial cares for multi-valued relations to capture the flexibility given by nondeterminism. For multi-level implementations, we extended the theory of binary observability don't cares for multi-valued combinational logic networks, where each node has many multi-valued inputs and a single multi-valued output. We developed a method to construct multi-valued observability don't cares in a combinational network and showed how to generate observability partial cares. Experimental results were given to show the effectiveness of using CODC's for node minimization.

- As an application of multi-valued relation minimization using partial cares, we studied the problem of protocol converter synthesis as a control synthesis problem. We showed

how the interface between two different synchronous protocols can be generated as an environment that controls the system composed of the two protocols. The resulting protocol converter can be optimized structurally rather than behaviorally using partial cares. Some examples were given to show the performance the proposed scheme.

- As an application of multi-valued logic network minimization, we studied the problem of software compilation for embedded applications. We proposed MV-networks as an intermediate representation for control dominated software optimization and developed methods to interface between software codes and MV-networks. We proposed methods to generate assembly code from MV-networks for simulation and studied the trade off between different schemes.

5.2 Future work

Some future research directions are:

Multi-level encoding: We built up a framework for multi-level multi-valued logic minimizations in MVSIS. One research direction is to develop the theory and practice for the encoding problem of multi-level multi-valued circuits, where each intermediate variable has multiple values. The encoding of an intermediate wire will be constrained by the driving node and all fanout nodes. A particular encoding can generate further don't cares through dominance relations and equivalence relations between different binary codes. An efficient encoding scheme will benefit both hardware implementation and software implementation.

Multi-valued network minimization scripts: We are developing a suite of optimizing packages for manipulating a MV-network, including both algebraic and don't care-based methods. This is being done in a fashion like SIS [SSL+92], where a set of commands are applied repetitively until no further improvements. The goal of minimization depends on the final types of target implementation, but minimizing the number of cubes and literals is generally beneficial for both hardware and software. Algebraic methods include division, factorization, collapsing, resubstitution and decomposition; don't care-based methods include node minimization using observability don't cares and possibly partial cares; also network structuring methods that are not present in SIS are table grouping and splitting,

input variable pairing and encoding, input-output variable encoding, etc.. We believe that such a multi-valued logic optimizing package will find it's application in various important areas.

Multi-valued technology mapping: In the hardware world, there is only a limited set of standard cell gates performing basic logic operations, e.g. AND, OR, XOR. However in software, the final implementation is a sequence of instructions that run on a standard processor. Instruction sets for common RISC architectures include both logic manipulations and complex arithmetic operations. Until now, all proposed methods that apply logic optimization to software compilation are able to use only a small subset of the available instructions, i.e. logic operations like AND, OR, XOR. We believe the following, if successfully formulated and carried out, would be of great contribution: technology mapping from multi-valued logic networks to complex instruction sets including arithmetic operations like addition, subtraction, shifting and multiplication, etc..

Software synthesis: One distinguishing feature of embedded software from general purpose software is that the application is part of a larger system and needs to interact with other components concurrently in real time. The problems involved in this area of embedded compilation are: how to deal with the concurrency between different software processes and between software and hardware components; how to perform timing analysis and predict the worst case execution time of a software process; if timing constraints are not met, how to reduce software critical path. These are problems that have been fully understood and solved in the hardware world, but just brought up recently in the software world. There have been studies of the above problems in various contexts. We believe that a software framework or methodology can be built to tackle these problems systematically, benefiting from the research in the hardware world. The key question is the interfacing of different problem domains.

New embedded architecture: MV-network based software evaluation is pointing to a new type of embedded architecture, where only a small set of logic instructions are needed for the control path, while data computation are carried out using special or configurable hardware. This may be much more efficient from a cost perspective for some applications.

Bibliography

[AH96] R. Alur and T. A. Henzinger. Reactive modules. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 207–218. IEEE Computer Society Press, 1996.

[AHM+98] R. Alur, T. A. Henzinger, F. Y.C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Proceddings of the Tenth International Conference on Computer-aided Verification*, 1998.

[ARM] ARM Inc. `http://www.arm.com`.

[BCG+97] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B.Tabbara. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach.* Kluwer Academic Press, 1997.

[BCG+99] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. L. Sangiovanni-Vincentelli, E. M. Sentovich, and K. Suzuki. Synthesis of software programs for embedded control applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 18(6):834–49, June 1999.

[BCH+91] R. K. Brayton, M. Chiodo, R. Hojati, T. Kam, K. Kodandapani, R. P. Kurshan, S. Malik, A. L. Sangiovanni-Vincentelli, E. M. Sentovich, T. Shiple, K. J. Singh, and H.-Y. Wang. BLIF-MV: An Interchange Format for Design Verification and Synthesis. Technical Report UCB/ERL M91/97, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, November 1991.

[Be99] R. K. Brayton and etc. EECS219B class project. *UC.Berkeley*, 1999.

[BG92] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 1992.

[BK99] R. K. Brayton and S. P. Khatri. Multi-valued logic synthesis. In *Proceddings of the International conference on VLSI Design*, 1999.

[Bra99] R. K. Brayton. Algebraic methods for multi-valued logic. Technical Report UCB/ERL M99/62, Electronics Research Laboratory, University of California, Berkeley, Dec. 1999.

[BS89a] R. K. Brayton and F. Somenzi. An Exact Minimizer for Boolean Relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 316–319, November 1989.

[BS89b] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *Proc. of the Intl. Conf. on VLSI*, pages 231–240, August 1989.

[BS89c] R. K. Brayton and F. Somenzi. Boolean Relations and the Incomplete Specification of Logic Networks. In *Proc. of the Intl. Conf. on VLSI'89*, Munich, August 1989.

[BS89d] R. K. Brayton and F. Somenzi. Minimization of Boolean Relations. In *Proc. of the Intl. Symposium on Circuits and Systems*, pages 738–743, May 1989.

[CGH+95] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. Synthesis of software programs from CFSM specifications. In *Proc. of the Design Automation Conf.*, June 1995.

[DM90] M. Damiani and G. De. Micheli. Observability don't care sets and boolean relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, 1990.

[Edw97] S. Edwards. *The Specification and Execution of Heterogeneous Synchronous Reactive Systems.* PhD thesis, University of California Berkeley, 1997.

[Edw00] S. Edwards. Compiling esterel into sequential code. In *Proc. of the Design Automation Conf.*, June 2000.

[ELLSV97] S. Edwards, L. Lavagno, E. A. Lee, and A. L. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proc. of the IEEE*, 85(3), March 1997.

[FB96] K. M. Fant and S. A. Brandt. NULL conventional logic: A complete and consistent logic for asynchronous digital circuit synthesis. In *Proceddings of the International conference on application-specific systems, architectures and processors*, 1996.

[GB00] Minxi Gao and R. K. Brayton. Semi-algebraic methods for multi-valued logic. In *Proc. of the Intl. Workshop on Logic Synthesis*, May. 2000.

[GDN92] A. Ghosh, S. Devadas, and A. R. Newton. Heuristic minimization of boolean relations using testing techniques. *IEEE Transaction on CAD*, 1992.

[GP98] S. Grygiel and M. Perkowski. New compact representation of multiple-valued functions, relations, and non-deterministic state machines. In *Proceedings of the 23st Design Automation Conference*, 1998.

[KB90] T. Kam and R. K. Brayton. Multi-valued deisoin diagrams. Technical Report UCB/ERL M90/125, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, December 1990.

[Lee99] E. A. Lee. Embedded software - an agenda for research. Technical Report UCB/ERL M99/63, Electronics Research Laboratory, University of California, Berkeley, Dec 1999.

[LFS+] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial hdl synthesis tools. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000).*

[LMBSV90] L Lavagno, S Malik, R Brayton, and A Sangiovanni-Vincentelli. MIS-MV: Optimization of multi-level logic with multiple-valued inputs. In *Proceedings of the International Conference on Computer-Aided Design*, 1990.

[LS90] B. Lin and F. Somenzi. Minimization of Symbolic Relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 88–91, November 1990.

[ntr99] The International Tecnology Roadmap for Semiconductors. `http://www.itrs.net/ntrs/publntrs.nsf`, 1999.

[RSV88] R. L. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Transaction on CAD*, 1988.

[Sav92] Hamid Savoj. *Don't Cares in Multi-Level Network Optimization.* PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[SB90] H. Savoj and R. K. Brayton. The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks. In *Proc. of the Design Automation Conf.*, pages 297–301, June 1990.

[SBT91] H. Savoj, R. K. Brayton, and H. Touati. Extracting Local Don't Cares for Network Optimization. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 514–517, November 1991.

[SKMB90] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for Discrete Function Manipulation. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 92–95, November 1990.

[SSB93] E. M. Sentovich, V. Singhal, and R. K. Brayton. Multiple Boolean Relations. In *Workshop Notes of the Intl. Workshop on Logic Synthesis*, Tahoe City, CA, May 1993.

[SSL+92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

[THE] Theseus Logic Inc. `http://www.theseus.com`.

[VIS96] VISgroup. VIS: A system for verification and synthesis. In *IEEE International Conference on Computer-Aided Verification*, 1996.

[VKBSV97] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Logic Optimization.* Kluwer Academic Publishers, 1997.

[VSV89] T. Villa and A. L. Sangiovanni-Vincentelli. NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations. In *Proc. of the Design Automation Conf.*, pages 327–332, 1989.

[VSV90] T. Villa and A. L. Sangiovanni-Vincentelli. NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 9(9):905–924, September 1990.

[WB91a] Y. Watanabe and R. K. Brayton. Heuristic Minimization of Boolean Relations. In *Proc. of the MCNC Intl. Workshop on Logic Synthesis*, volume I, May 1991.

[WB91b] Y. Watanabe and R. K. Brayton. Heuristic Minimization of Multiple-Valued Relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 126–129, November 1991.

[WB93a] Y. Watanabe and R. K. Brayton. Heuristic minimization of multiple-valued relations. *IEEE Transaction on CAD*, 1993.

[WB93b] Y. Watanabe and R. K. Brayton. The Maximum Set of Permissible Behaviors for FSM Networks. In *Proc. of the Intl. Conf. on Computer-Aided Design*, 1993.

[WGB96] Y. Watanabe, L. M. Guerra, and R. K. Brayton. Permissible functions for multioutput components in combinational logic optimization. *IEEE Transaction on CAD*, 1996.

[XTE] Tensilica Inc. `http://www.tensilica.com`.