OPTIMAL ENCODING OF CONTROL LOGIC

Giovanni De Micheli
IBM-Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598

Abstract. The design of control-units for VLSI systems, such as microprocessors, is addressed in this paper. Control units are modeled as finite automata and an encoding scheme for the primary inputs and the internal states is presented that minimizes the silicon area requirement for the sequencing and control store.

1. INTRODUCTION

The Computer-Aided synthesis of Very Large Scale Integration (VLSI) system modules must include design optimization procedures to be effective [NEWT81]. The design of sequential functions, such as microprocessor control units, is a good test case for automated synthesis tools.

This paper addresses the computer-aided design of VLSI control units. The functional specifications of the system being designed are assumed to be given in a Hardware Description Language program, a flow-chart or an equivalent representation. Similarly it is assumed that the design is partitioned into two major components: the data-flow and the control unit.

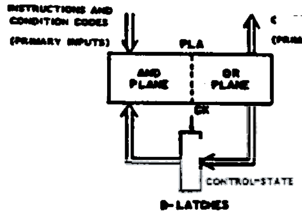
Control-unit implementations have followed different strategies. A customized design of the control unit can be achieved by interconnecting logic gates. For example, the control part of the Z8000 microprocessor was implemented by "random logic" gates. Such a design style may lead to a compact and high-performance implementation, but it is highly dependent on the particular control flow. Moreover, design time is longer in comparison to other structured implementations and engineering changes may require a complete redesign. Other microprocessor are completely microprogrammed, as in the case of the M68000. In this case, the sequencing and control functions have a structured implementation, that is referred to as sequencing and control-store or more simply control-store. The control-store is implemented by a Read Only Memory (ROM), a Programmable Logic Array (PLA) or a more elaborate structure [ANDR80]. There are several advantages in using microprogramming. The design of the control unit is flexible, can be defined at a later stage of the processor design and can be easily modified. Moreover microprogrammed processors can emulate the instruction set of other machines.

With the advent of VLSI circuits, microprogramming became a way to increase the regularity of the chip structure [LEWI81]. Davis [DAVI72] recognized this new view of microprogramming and gave the following definition: "One particular class of control mechanism uses regularly organized storage arrays to contain a large part of the control information. Machines employing such control mechanism are said to be microprogrammed." Different implementation strategies of microprogrammed processors have been followed: the control-store can be located on-chip or off-chip and can be writable or not.

We consider here the problem of designing control units with read-only on-chip control store, i.e. the programmability feature is retained until the logic design stage of the control unit. Most design have shown a common problem: the control-store takes a large fraction of the chip. It is therefore important to reduce the silicon area taken by the control-store, while keeping the programmability feature. Moreover reducing the storage area corresponds in general to shorter connection wires and consequently to faster switching-time performance.

The problem of reducing the silicon area taken by the control-store in a ROM-based microprogrammed control-units has been investigated by several researchers and surveyed in [AGER76]. Recently, PLAs have shown to be effective components for implementing control logic [GRAS83] [PAPA79] [LOGU75]. PLA design is regular and structured and can be supported by computer-aided tools [DEMI84a]. Moreover PLAs implement logic functions more efficiently than ROMs, as far as silicon area is concerned. Several techniques, like logic minimization [HONG74] [BRAY84] and topological compaction [HACH82a] [DEMI83c], allow the design of area-effective PLA implementations. Therefore PLA-based control unit designs can be optimized with regard to silicon area requirement and subsequently to switching-time performance.

Control units are modeled here as deterministic synchronous finite automata [HART66] and implemented by two components: a combinational component and a set of registers, that are synchronized to the system clock and that store the state of the control unit. The combinational circuit implements the sequencing and control store. It generates the control signals to the data-flow (primary outputs) and the next control state as a function of the present control state, operation and condition codes (primary inputs). The combinational component can be implemented by a single PLA or be partitioned into two or more arrays. In the former case, the PLA implements both the sequencing and control store. In the latter an array implements the sequencing store that generates the sequence of control states as a function of the operation and condition codes and the present control-state. Another array generates the control signals as a function of the control-states. The former model is used in the sequel. The extension of the technique presented here to the latter is straight-forward.



PLAs implement two-level switching functions as sum-of-products or equivalent representation. The silicon area taken by a PLA has a complex functional dependence on the binary representation of the control-states and primary inputs.

We propose a new technique for encoding the control states and primary inputs to minimize the PLA area implementing the sequencing and control store. This problem is related to the state assignment problem for deterministic automata that has been recently investigated [DEMI83f] [DEMI83g] [DEMI84b] [DEMI84b].

The state assignment problem has been the object of extensive theoretical research. A critical survey of the major published results is presented in [DEMI83g]. However, despite of all these efforts, to the best of my knowledge, no computer-aided design tool is in use today for an optimal encoding of control logic.

2. CONTROL-UNIT REPRESENTATION AND MINIMIZATION

For the sake of our analysis, we assume that a control-unit is described by a **table of micro-operations** [LANG82]. The table of micro-operations can be constructed from a Hardware Description Language or flow-chart description. The processor state in defined at any time by a mnemonic string, defining the **control state** . We assume that each control-state corresponds to a data-flow cycle. The table of micro-operations describes, for each control state and instruction, the action to be taken by the data-flow, which is specified by the control point activation signals. The table specifies as well the control-state in which the control-unit will be at the next cycle. Though the technique presented here is fairly general, we concentrate on a particular control-unit design for the sake of concreteness.

Example 2.1: We consider here the microprocessor design described by Langdon in Chapter 5 of [LANG82]. Langdon presented a custom PLA implementation of the control store (he did not specify the next-state function implementation). The purpose of showing this example is not to claim a better implementation, but to present a design method on a documented example. The following table of micro-operations is adapted from [LANG82] and describes the memory-reference instructions. (Therefore extended op-code and the corresponding control signal is not considered.)

TABLE OF MICRO-OPERATIONS

STATE	OP-CODE	MODE	NEXT-STATE	CONTROL-SIGNALS
I1			I2	00000000000000111
I2			A1	00000000111011000
A1	JMP	DIRECT	I1	00000100000100100
A1	SRJ	DIRECT	A3	00000000001000001
A1	SAC	DIRECT	A4	00000010000000000
A1	ISZ	DIRECT	A4	00000000000000000
A1	LAC	DIRECT	A4	00000000000000000
A1	AND	DIRECT	A4	00000000000000000
A1	ADD	DIRECT	A4	00000000000000000
A1	JMP	INDIRECT	A2	00000000000000000
A1	SRJ	INDIRECT	A2	00000000001000001
A1	SAC	INDIRECT	A2	00000000000000000
A1	ISZ	INDIRECT	A2	00000000000000000
A1	LAC	INDIRECT	A2	00000000000000000
A1	AND	INDIRECT	A2	00000000000000000
A1	ADD	INDIRECT	A2	00000000000000000
A1	JMP	INDEXED	I1	00001101000100100
A1	SRJ	INDEXED	A2	00000000001000001
A1	SAC	INDEXED	A3	00001101000100000
A1	ISZ	INDEXED	A3	00001101000100000
A1	LAC	INDEXED	A3	00001101000100000
A1	AND	INDEXED	A3	00001101000100000
A1	ADD	INDEXED	A3	00001101000100000
A2		DIRECT	A3	00000001010001000
A2		INDIRECT	A3	00000001010001000
A2		INDEXED	A3	00001101000100100
A3	JMP		I1	00010101000000100
A3	SRJ	DIRECT	A4	00010110000000110
A3	SRJ	INDIRECT	A4	00010110000000110
A3	SRJ	INDEXED	A4	00000010000000111
A3	SAC		A4	00000010000000000
A3	ISZ		A4	00000000000000000
A3	LAC		A4	00000000000000000
A3	AND		A4	00000000000000000
A3	ADD		A4	00000000000000000
A4	JMP		E1	00000001010000001
A4	SRJ		I1	00000001000000001
A4	SAC		I1	00000001000000001
A4	ISZ		E1	00000000010000000
A4	LAC		E1	00000001010000001
A4	AND		E1	00000001010000001
A4	ADD		E1	00000001010000001
E1	LAC		I1	00110100000000000
E1	AND		I1	01000100000000000
E1	ADD		I1	00100100000000000
E1	ISZ		E2	00010100001000010
E2	ISZ		E3	10010110000000010
E3			I1	00000001000000101

The control-unit has nine states, corresponding to instruction-fetch, operand-address evaluation and instruction execution. The states are labeled by mnemonic strings, namely: I1, I2, A1, A2, A3, A4, E1, E2, E3. We consider seven operations, namely: JMP (jump), SRJ (subroutine jump), SAC (store accumulator), ISZ (increase and skip on 0), LAC (load accumulator), AND (and), ADD (add). Three modes of memory addressing are considered: DIRECT, INDIRECT and INDEXED. The operation and addressing mode are specified by two instruction fields.
Each row of the table shows an action as a consequence of particular conditions. There are five fields in each row. Two fields correspond to the present and next control states. Two fields correspond to the primary inputs, i.e. the operation code and addressing mode. The last field corresponds to the control signals. The first four fields are described by mnemonic fields, the last by binary variables. We chose here to represent control signals by binary variables. However note that control signals could be described by a mnemonic field as well.
For example, the first row shows that when the control-unit is in state I1, the state of the control-unit at the next cycle is I2 and control signal 00000000000000111 (incrementing the program-counter) is issued. The third row shows that when the control unit is in state A1, the op-code is JMP and the addressing mode is DIRECT, then the next control-state is I1 and the control signal is 00000100000100100. A unspecified field in the table corresponds to a "don't care" condition. For example the op-code and the mode are "don't care" conditions for the transition specified by the first row.

The table of micro-operations can be implemented by a ROM or a PLA in a straight-forward way. Each row can be associated to a multiple-output minterm or product-term. Each state can be associated to a state signal line and stored by a single latch. Each primary input, i.e. each op-code and mode, can be associated to an input signal line. Such a representation leads to an inefficient use of silicon area, even when the sequencing and control-store are implemented by a PLA. (The size of a ROM implementing such a table grows exponentially with the number of signals carrying state and condition information. PLA implementations are more area-efficient than ROM implementations.) The waste of silicon area corresponds as well to a degradation of the circuit performance. We proved in [DEMI84b] that it is always possible to construct binary encodings of the primary inputs and control-states of a deterministic automaton leading to a more efficient implementation. However the problem of determining an encoding that minimizes the PLA area is extremely complex.

For this reason, some simplifying assumptions are needed. As a first step, topological compaction techniques to reduce the PLA area, such as folding [DEMI83c] and partitioning [DEM83d] are not considered. Under this assumption, the PLA area is proportional to the product of the number of rows (implementing the product-terms) times the number of columns (carrying the input/output and state information). Both row and column cardinality depend on the encoding of the control-states and primary inputs (i.e. operation codes and addressing modes) represented by mnemonic strings in the table of micro-operations. The (minimum) number of rows is the cardinality of the (minimum) cover of the control store according to a given encoding. The code-length (i.e. the number of bits used to represent the mnemonic strings) is related to the number of PLA columns and in particular to the number of PLA input and output columns corresponding to the present/next control-states, operation codes and modes. Therefore the PLA area has a complex functional dependence on state and instruction representation. For this reason two simpler optimal encoding problems are defined:

i) Find the encodings of minimum code length among the encodings that minimize the number of rows of the PLA.

ii) Find the encodings that minimizes the number of rows of the PLA among the encoding of given code length.

The optimum solution to the control logic encoding problem, which minimizes the PLA area, can be seen as a trade-off between the solutions to problem i) and ii). Note that the above problems are still computationally difficult and to date no method (other than exhaustive search) is known that solves them exactly. Therefore heuristic strategies are used to approximate their solution.

The control-state representation is local to the control-unit. Therefore there are in general no constraints on the control-state representation and in particular on the encoding length. However the instruction fields, that are primary inputs to the control-unit, have to be compatible with the processor architecture requirements. Therefore it is desirable that the number of bits corresponding to the primary-input representation matches the instruction field width. In this case, no instruction decoder is needed, and the appropriate instruction fields can be gated directly as inputs to the control-unit.

The encoding technique reported in the sequel is related to the optimal state assignment problem for Finite State Machines [DEMI83g] [DEMI84a] [DEMI84b]. The strategy is based on the following idea: logic minimization of the combinational component of the control-unit is applied before the encoding. For this reason, logic minimization is performed on a symbolic (code independent) representation. The table of micro-operations is a **symbolic cover** of the control store. Symbolic covers have been introduced in [DEMI83f] to specify a combinational function by means of binary and mnemonic strings. A symbolic cover is a set of primitive elements called **symbolic implicants**. Each row of the table of micro-operations is a symbolic implicant and consists of a set of (mnemonic and/or binary) fields describing a state transition and the corresponding primary inputs and outputs.

A symbolic cover can be considered as a logic cover of a multiple-valued logic function [SU72] [HONG74], where each entry in each mnemonic field takes a different logic level and is represented by a character string. Several notations are used to represent multiple-valued logic covers. For example, the different logic levels can be represented by integer values: 0,1,2, ... $p-1$. This is an extension of the binary notation to a p-valued representation.

The **positional cube** notation is used here [SU72]. A p-valued logical variable is represented by a string of p binary symbols. Value r is represented by a "1" in the r^{th} position, all others being "0". Note that the positional cube notation allows the representation of a set of values with one string. The disjunction (multiple-valued logical OR) of several values is represented by a string having "1"s in the corresponding positions. Therefore the "don't care" value is represented by a string of "1"s and the empty value by a string of "0"s.

The transformation of a table of micro-operations into a multiple-valued cover with positional cube notation is straight-forward, since the transformation involves only symbol translations.

Example 2.2: The table of micro-operations of Example 2.1 can be translated into a multiple-valued positional-cube representation by associating a value to each state, operation code and addressing mode. There are 9 states, 7 op-codes and 3 modes represented by 9-bit, 7-bit and 3-bit strings respectively. For example I1 is represented by 100000000, I2 by 010000000, etc. Similarly operation JMP is represented by 1000000, ... addressing mode DIRECT is represented by 100, etc.

SYMBOLIC COVER

STATE	OP-CODE	MODE	NEXT-STATE	CONTROL-SIGNALS
100000000	1111111	111	010000000	00000000000000111
010000000	1111111	111	001000000	00000000111011000
001000000	1000000	100	100000000	00000100000100100
001000000	0100000	100	000010000	00000000001000001
001000000	0010000	100	000001000	00000010000000000
001000000	0001000	100	000001000	00000000000000000
001000000	0000100	100	000001000	00000000000000000
001000000	0000010	100	000001000	00000000000000000
001000000	0000001	100	000001000	00000000000000000
001000000	1000000	010	000100000	00000000000000000
001000000	0100000	010	000100000	00000000001000001
001000000	0010000	010	000100000	00000000000000000
001000000	0001000	010	000100000	00000000000000000
001000000	0000100	010	000100000	00000000000000000
001000000	0000010	010	000100000	00000000000000000
001000000	0000001	010	000100000	00000000000000000
001000000	1000000	001	100000000	00001101000100100
001000000	0100000	001	000100000	00000000001000001
001000000	0010000	001	000010000	00001101000100000
001000000	0001000	001	000010000	00001101000100000
001000000	0000100	001	000010000	00001101000100000
001000000	0000010	001	000010000	00001101000100000
001000000	0000001	001	000010000	00001101000100000
000100000	1111111	100	000010000	00000001010001000
000100000	1111111	010	000010000	00000001010001000
000100000	1111111	001	000010000	00001101000100100
000010000	1000000	111	100000000	00010101000000100
000010000	0100000	100	000001000	00010110000000110
000010000	0100000	010	000001000	00010110000000110
000010000	0100000	001	000001000	00000010000000111
000010000	0010000	111	000001000	00000010000000000
000010000	0001000	111	000001000	00000000000000000
000010000	0000100	111	000001000	00000000000000000
000010000	0000010	111	000001000	00000000000000000
000010000	0000001	111	000001000	00000000000000000
000001000	1000000	111	000000100	00000001010000001
000001000	0100000	111	100000000	00000001000000001
000001000	0010000	111	100000000	00000001000000001
000001000	0001000	111	000000100	00000000010000000
000001000	0000100	111	000000100	00000001010000001
000001000	0000010	111	000000100	00000001010000001
000001000	0000001	111	000000100	00000001010000001
000000100	0000100	111	100000000	00110100000000000
000000100	0000010	111	100000000	01000100000000000
000000100	0000001	111	100000000	00100100000000000
000000100	0001000	111	000000010	00010100001000010
000000010	0001000	111	000000001	10010110000000010
000000001	1111111	111	100000000	00000001000000101

Minimizing a symbolic cover is equivalent to finding a representation of the control store with the minimum number of symbolic implicants. Finding a minimum multiple-valued cover is a computationally expensive problem. Heuristic multiple-valued logic minimizers, such as MINI [HONG74] can be used to compute a minimal (local minimum) cover. (Program MINI [HONG74] is used in general for binary-valued logic minimization; however it supports multiple-valued minimization as well.) Alternatively, the positional-cube representation can be seen as a binary-valued encoding of a multiple-valued function. This encoding is referred to as **1-hot coding**, because each value of the multiple-valued function corresponds to one and only one binary value "1" (HIGH) in the coded representation.[1] By using this representation, binary-valued minimizers, such as PRESTO [BROW80], POP, MINI [HONG74] and ESPRESSO-II [BRAY84], can be used to obtain minimal symbolic covers. Experimental results have shown that ESPRESSO-II yields minimal (symbolic) covers that are quite close to the minimum (symbolic) cover, for problems for which the minimum cover can be determined [DEMI84a].

Example 2.3: Consider the symbolic cover of Example 2.2. A minimal symbolic (multiple-valued) cover, obtained by ESPRESSO-II, is the following:

MINIMAL SYMBOLIC COVER

STATE	OP-CODE	MODE	NEXT-STATE	CONTROL SIGNALS
000000001	1111111	111	100000000	00000001000000101
100000000	1111111	111	010000000	00000000000000111
010000000	1111111	111	001000000	00000000111011000
000100000	1111111	110	000010000	00000001010001000
000010000	0111111	111	000001000	00000000000000000
000001000	1000111	111	000000100	00000001010000001
001100000	0011111	001	000010000	00001101000100000
001000000	1111111	010	000100000	00000000000000000
000001000	0110000	111	100000000	00000001000000001
001000000	0001111	100	000001000	00000000000000000
000001000	0001000	111	000000100	00000000010000000
000000100	0000010	111	100000000	01000100000000000
000000100	0000001	111	100000000	00100100000000000
000000100	0000100	111	100000000	00110100000000000
000000100	0001000	111	000000010	00010100001000010
000000010	1111111	111	000000001	10010110000000010
000010000	0010000	111	000000000	00000010000000000
001010110	0010000	100	000001000	00000010000000000
001000000	0100000	011	000100000	00000000001000001
000010000	1000000	011	100000000	00010101000000100
000100000	1100000	001	000010000	00001101000100100
000010000	0100000	110	000000000	00010110000000110
001000000	0100000	100	000010000	00000000001000001
001000000	1000000	100	100000000	00000100000100100
000010000	0100000	001	000000000	00000010000000111
001000000	1000000	001	100000000	00001101000100100

[1] The 1-hot representation has a different interpretation than the positional cube notation. An appropriate "don't care" set must be specified for the 1-hot representation, to specify that n-hot encodings do not represent existing values. The interested reader is referred to [BRAY84] and [DEMI83g] for details.

Consider now the 7-th symbolic implicant from the top:
001100000 0011111 001 000010000 00001101000100000
This implicant shows that operations SAC, ISZ, LAC, AND, ADD and mode INDEXED cause a transition from either state A1 or A2 to state A3 and implies the control signals specified by the last field.

The example above shows that the effect of symbolic (multiple-valued) logic minimization is to group together the transitions from some control-state and under some op-code and mode into the same next-state and activating the same control signals. Each proper subset of mnemonics represented in the same field and containing more than one element is termed **group**.

Example 2.4 Let us consider the mnemonic fields corresponding to the control-states, operation codes and addressing modes in the minimal symbolic cover of the control-unit of Example 2.1 2.2 2.3. There are two groups of control states, namely: {A1; A2} and {A1; A3; E1; E2}. There are six different groups of op-codes, namely: {ISZ; LAC; AND; ADD}, {JMP; LAC; AND; ADD }, {SAC; ISZ; LAC; AND; ADD}, {SRJ; SAC; ISZ; LAC; AND; ADD}, {JMP; SRJ}, {SRJ; SAC}. There are two different groups of addressing modes, namely: {DIRECT; INDIRECT} and {INDIRECT; INDEXED}.

Given an encoding and a group, the corresponding **group face** (or simply **face**) is the minimal dimension Boolean subspace containing the encodings of the mnemonics assigned to that group (or equivalently the bit-wise disjunction of the encodings assigned to the mnemonics in that group).

The goal of the encoding technique presented here is to group together the encodings in binary-valued logical implicants in the same way mnemonics are grouped in the minimal symbolic (multiple-valued) cover. In particular, an encoding is sought, such that each symbolic implicant can be coded by one binary-valued implicant. For this assignment, there exists a binary-valued cover of the control-unit having as many implicants as the minimal symbolic cover.

An encoding, such that each group face contains the encodings of the mnemonic strings included in the corresponding group and no other, satisfies the above requirement. For this reason, a **constrained encoding problem** is considered:

Given a set of groups, find an encoding such that each group face does not intersect the code assigned to any mnemonic string not contained in the corresponding group.

In view of the previous considerations, any solution to the constrained encoding problem is an assignment such that the encoded Boolean cover has the same cardinality as the minimal symbolic cover. We proved in [DEMI84a] that there always exist solutions to this problem. Unfortunately this problem does not specify the encoding length. In some cases, the encoding length is a design specification. Then we would like to find a solution to the above problem that satisfies a bound on the encoding length. (We assume that the bound is greater or equal to the ceiling of the logarithm of the number of mnemonic strings.) In general, a solution to this problem may not exist. We could then consider the problem of finding an encoding of bounded length such that a maximal number of group faces do not intersect the code assigned to any mnemonic not contained in the corresponding group. A solution to this problem would in general not allow to encode every symbolic implicant of the minimal cover by a Boolean implicant only. However, if most group constraints are satisfied, then only few symbolic implicants have to be encoded by more than one Boolean implicant. For this reason, it is important to relate the unsatisfied group constraints to the number of additional product-terms needed to implement the Boolean cover. A **length-bounded constrained encoding problem** can be stated as follows:

Given a minimal symbolic cover and a bound on the code-length, find an encoding of bounded length that minimizes the cardinality of the corresponding Boolean cover.

3. CONSTRAINED ENCODING

The minimal symbolic representation defines the constraints of an encoding problem, whose solutions are the encodings that allow the implementation of the control store with as many product terms as the cardinality of the minimal symbolic cover.

We consider now the first encoding problem. In our example, the encoding of control-states, op-codes and modes are independent of each other. Therefore we concentrate on the encoding of the mnemonic strings in the same field.

Even if no bound on the encoding length is specified, it is desirable to encode the mnemonics with the minimal number of bits. Therefore an optimal solution to the constrained encoding problem is a minimal length-solution. The geometric interpretation of the optimal encoding problem is: ***finding the minimal dimension Boolean space in which each group face is a subspace which does not intersect the encoding assigned to any mnemonic not contained in the corresponding group*** .

Encoding is restricted here to one-to-one mappings between the set of mnemonics and a subset of the vertices of the Boolean hypercube, i.e. each encoding is a 0-dimensional subspace. This restriction is motivated as follows. A 0-dimensional assignment that is a solution to the constrained encoding problem, can be derived from a n-dimensional (n>0) solution by assigning to each mnemonic a vertex contained in the corresponding n-dimensional assignment. Therefore a 0-dimensional solution has code-length less than or at most equal to the code-length of any n-dimensional solution.

Optimal constrained encoding is a complex problem of combinatorial optimization. To date, it is not known whether an optimal solution can be computed by an non-enumerative procedure. The frame of a heuristic algorithm is presented here, that constructs a state assignment satisfying the constraint relation. The algorithm is described in detail in [DEMI84b] and [DEMI84c].

We introduce first some definitions. Let n_s be the number of mnemonics to encode, n_l the number of groups and n_b the code length. To be consistent with the positional-cube notation, groups are represented by a 1-0 matrix and in particular by the subset of the columns of the minimal multiple-valued cover corresponding to the field under consideration. The **constraint matrix** A is a matrix: $A \in \{0,1\}^{n_l \times n_s}$ representing n_l groups. Mnemonic string j belongs to group i if $a_{ij} = 1$.

Example 3.1: The following constraint matrix is derived from the minimal symbolic cover of Example 2.3 and represent the state groups: {A1; A2}, {A1; A3; E1; E2}

$$A = \begin{bmatrix} 001100000 \\ 001010110 \end{bmatrix}$$

The **encoding matrix** S is a matrix $S \in \{0,1\}^{n_s \times n_b}$ whose rows are the encodings. Our problem is to determine the encoding matrix S, given a constraint matrix A. An encoding matrix S is said to **satisfy the constraint relation** for a given A if S is a solution to the constrained encoding problem specified by A.

The encoding algorithm constructs an encoding matrix S row by row and column by column by an iterative procedure. At each step a larger set of mnemonics is considered and an encoding matrix S is computed that satisfies the constraint relation for the corresponding columns of A. For each mnemonic that is being considered, a new row is appended to S . The encoding matrix S is initialized to a 1-column matrix, and columns are appended to S (i.e. the code-length n_b is increased) only when needed to satisfy the constraint relation. The structure of the algorithm is the following:

STEP 1: Select an uncoded mnemonic.
STEP 2: Determine the encodings for that mnemonic satisfying the constraint relation.
STEP 3: If no encoding exists, increase encoding length and go to STEP 2.
STEP 4: Assign an encoding to the selected mnemonic.
STEP 5: If all mnemonic have been encoded, stop. Else go to STEP 1.

Mnemonics are selected at STEP 1 according to a heuristic criterion, fully referenced in [DEMI84b]. At STEP 2 all the possible encodings for the selected mnemonic are determined, so that the corresponding partial encoding matrix S satisfies the constraint relation for the corresponding columns of A . An appropriate encoding is selected at STEP 4, according to a heuristic rule [DEMI84b] [DEMI84c].

The encoding algorithm constructs an encoding matrix S that satisfies the constraint relation for the given constraint matrix A, i.e. S is a solution of the constrained encoding problem. Experimental results show that the length of the encoding generated by the algorithm is reasonably short, and often equal to the minimum length solution when this is known.

Example 3.2 Let us consider the constrained encoding of the control states of Example 2.3. The constraint matrix A is reported in Example 3.1. Suppose states are selected according to the following sequence: {A1; A2; A3; E1; E2; E3; I1; I2; A4}. Note that the last four states in the sequence do not belong to any group, and their encoding is not critical to the problem. The first state to be encoded is A1, and is encoded by 0. The second state is A2, and is encoded by 1. At this point $S = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and S satisfies the constraint relation for the corresponding columns of the constraint matrix, i.e. $\begin{bmatrix} 11 \\ 10 \end{bmatrix}$. The next selected state is A3. There is no 1-dimensional encoding that can be assigned to A3. Therefore the code space dimension n_b is increased by one, by appending a column of 0s to S, i.e. $S = \begin{bmatrix} 00 \\ 10 \end{bmatrix}$ Then a valid encoding for A3 is 01, because the face containing the encoding of A1 and A3 does not intersect the code of A2 and vice versa, or equivalently $S = \begin{bmatrix} 00 \\ 10 \\ 01 \end{bmatrix}$ satisfies the constraint relation for $\begin{bmatrix} 110 \\ 101 \end{bmatrix}$. State E1 is selected next. Encoding 11 cannot be assigned to E1 because the face corresponding to the partial group {A1;A3;E1} would intersect the encoding of A2 that is not in that group. Hence the code space dimension is increased again by appending to S a column of 0s and $S = \begin{bmatrix} 000 \\ 100 \\ 010 \end{bmatrix}$. Now both 001 and 011 are valid encodings for E1, i.e. the constraint relation is satisfied for either choice. Let us assign 001 to E1. State E2 is considered now. E2 can be encoded by 011. The remaining states do not belong to any group, and can be assigned to any encoding that does not intersect the existing faces. There are four states to be encoded, and three available encodings in the three dimensional space: i.e. 110 101 111. Therefore the space code dimension n_b must be increased to 4. The encoding matrix S constructed by the algorithm is:

$$S = \begin{bmatrix} 0000 \\ 1000 \\ 0100 \\ 0010 \\ 0110 \\ 1100 \\ 1010 \\ 1110 \\ 0001 \end{bmatrix} \cong \begin{bmatrix} A1 \\ A2 \\ A3 \\ E1 \\ E2 \\ E3 \\ I1 \\ I2 \\ A4 \end{bmatrix}$$

Each row from top to bottom is an encoding of a state according to the sequence given above. Note that the length of the encoding n_b is 4, and 4 is the minimum number of bits to encode 9 states.

Example 3.3: Consider now the mnemonic field corresponding to the op-codes. The corresponding constraint matrix is:

$$A = \begin{bmatrix} 0001111 \\ 1000111 \\ 0011111 \\ 0111111 \\ 1100000 \\ 0110000 \end{bmatrix}$$

where the columns, from left to right, correspond to the op-codes in the following sequence: {JMP; SRJ; SAC; ISZ; LAC; AND; ADD}. By inspecting the constraint matrix, it is possible to see that no encoding with fewer than 6 bits can satisfy the constraint relation. In particular, by considering the first two rows of A, it is clear that 4 is the minimal dimension of a Boolean space to encode the op-codes {JMP, ISZ, LAC, AND ADD}. In fact, while the last three op-codes can be encoded in a two-dimensional subspace, JMP and ISZ must be encoded along two different coordinate axes so that the face containing the encodings of JMP and {LAC, AND, ADD} does not contain the encoding of ISZ and vice versa. Moreover, by considering the first row of A, it is evident that SAC cannot be encoded in the three dimensional subspace spanned by the encodings of {ISZ, LAC, AND , ADD}, and by considering the fist two rows of A, SAC cannot be assigned to any vertex of the 4-dimensional space without violating some constraint. Hence at least 5 bits are required to encode {JMP, SAC, ISZ, LAC, AND, ADD}. Similarly SRJ cannot be assigned to any vertex of the 5-dimensional space and therefore at least 6 bits are needed. The encoding matrix for the op-codes computed by the algorithm is the following:

$$S = \begin{bmatrix} 110100 \\ 110001 \\ 110010 \\ 111000 \\ 000000 \\ 010000 \\ 100000 \end{bmatrix} \cong \begin{bmatrix} JMP \\ SRJ \\ SAC \\ ISZ \\ LAC \\ AND \\ ADD \end{bmatrix}$$

where each row, from top to bottom, is an encoding of an op-code according to the sequence given above. Note that the length of the encoding constructed by the algorithm is 6 and corresponds to the minimal-length of an encoding satisfying the above constraints. However note that the 7 op-codes could be encoded by using 3 bits, if we do not require to satisfy the constraints (or some of them).

A Boolean cover of the control store can be obtained by replacing the computed encodings into the minimal multi-valued cover.

Example 3.4: Consider the minimal symbolic cover of Example 2.3, and the encodings for the states and op-codes specified by Examples 3.2 and 3.3. The addressing modes DIRECT, INDIRECT and INDEXED are encoded by 00, 01, 11 respectively, so that the corresponding constraint relation is satisfied. To obtain a Boolean cover, the positional-cube notations are replaced by the corresponding encodings (or by the disjunction of the corresponding encodings).

MINIMAL BOOLEAN COVER

STATE	OP-CODE	NO	N-ST	CONTROL SIGNALS
1100	••••••	••	1010	000000001000000101
1010	••••••	••	1110	000000000000000111
1110	••••••	••	0000	000000000111011000
1000	••••••	0•	0100	000000001010001000
0100	•••0••	••	0001	000000000000000000
0001	••0•00	••	0010	000000001010000001
•000	•••0•0	11	0100	000011101000100000
0000	••••••	01	1000	000000000000000000
0001	1100••	••	1010	000000001000000001
0000	•••000	00	0001	000000000000000000
0001	111000	••	0010	000000000010000000
0010	010000	••	1010	010001000000000000
0010	100000	••	1010	001001000000000000
0010	000000	••	1010	001101000000000000
0010	111000	••	0110	000101000001000010
0110	••••••	••	1100	100101100000000010
0100	110010	••	0000	000000100000000000
0••0	110010	00	0001	000000100000000000
0000	110001	•1	1000	000000000001000001
0100	110100	•1	1010	000101010000000100
1000	110•0•	11	0100	000011101000100100
0100	110001	0•	0000	000101100000000110
0000	110001	00	0100	000000000001000001
0000	110100	00	1010	000001000000100100
0100	110001	11	0000	000000100000000111
0000	110100	11	1010	000011101000100100

The sequencing and control-store can be implemented by a PLA having 26 rows and 33 columns.

Remark 3.1: This encoding method transforms a minimal symbolic cover into a non-necessarily-minimal Boolean cover, because the information about next-control states is not considered [DEMI84b]. For example, suppose we reverse the first coordinate of the state encodings, or equivalently we complement the first column of the present and next-state field. Then, the 8th product-term from the top can be deleted, because its output part consists of 0s only. Encoding techniques that take into account the next-state information are still under investigation.

We consider now the second encoding problem, which is also relevant for our particular example, because it is desirable to encode the op-codes using three bits. In this case, we look for a solution in a Boolean space of the given dimension. If such a solution is not found, some constraints are relaxed to make the encoding possible. In general, constraints can be relaxed by modifying the constraint matrix or dropping some rows. As a general consequence, the Boolean cover of the control-unit cannot be obtained by replacing the positional-cube notations by the corresponding encodings and not every symbolic implicant can be expressed by one Boolean implicant. For this reason, it is important to be able to relate the release of a constraint to the possible increase of the Boolean cover cardinality. In the following algorithm, constraints are relaxed only by splitting a group into two (not necessarily disjoint) groups. This corresponds to replacing a row of the constraint matrix A by two rows, whose bit-wise disjunction is the original row. Since every group corresponds to a subset of implicants, a split corresponds to duplicating these implicants, and it is therefore possible to assign a weight to each group accordingly. It is obvious that by repeating group splitting, the constraint matrix will eventually be an empty matrix and and encoding can be found for any original set of groups and any bound. (By definition, a group has more than one element and rows with one non-zero element only can be dropped from A.) The structure of the bounded-length constrained encoding algorithm is the following:

STEP 1: Select an uncoded mnemonic.
STEP 2: Determine the encodings for that mnemonic satisfying the constraint relation.
STEP 3: If no encoding exists and the encoding length is strictly shorter than the given bound, increase encoding length and go to STEP 2. If no encoding exists and the encoding length is equal to the given bound, relax a constraint and go to STEP 2.
STEP 4: Assign an encoding to the selected mnemonic.
STEP 5: If all mnemonic have been encoded, stop. Else go to STEP 1.

The algorithm differs from the previous one in STEP 3 and in the heuristic selection rules. A heuristic criterion is used to select the group to be split and how the split is done. Several factors are taken into account: the group weight, the group cardinality and the possibility of splitting a group into subgroups that are compatible with the computed partial encoding.

Example 3.5: Consider again the problem of encoding the op-codes and suppose that 3 is an upper bound on the encoding length. Suppose op-codes are encoded in the following sequence: {LAC; AND; ADD; JMP; ISZ; SAC; SRJ}. The encoding of the first four op-codes can be achieved in a three-dimensional space as before: i.e. the partial encoding matrix is: $S = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 011 \end{bmatrix}$. No encoding of ISZ can be found in the three dimensional space, because the encoding of the group {JMP; LAC; AND; ADD} spans the entire space. For this reason this group is selected at STEP 3 and split into the two groups: {JMP; ADD} {LAC;AND}. This particular split is chosen because the two new groups span a 1-dimensional space each, giving an "efficient use" of the Boolean space. Now ISZ can be encoded by 110. Next SAC is selected. Again no encoding of SAC can be determined because the partial group {SAC; ISZ; LAC; AND; ADD} spans the entire space. Therefore both the following groups must be split: {SRJ; SAC; ISZ; LAC; AND; ADD} and {SAC; ISZ; LAC; AND; ADD}. Since {ISZ; LAC; AND; ADD} is a group spanning a two-dimensional space, it is convenient to split the above groups into: {SRJ; SAC} U {ISZ; LAC; AND; ADD} and {SAC} U {ISZ; LAC; AND; ADD}. Now SAC can be encoded by 101 and SRJ by 001. The encoding matrix is:

$$S = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 011 \\ 110 \\ 101 \\ 001 \end{bmatrix} \equiv \begin{bmatrix} LAC \\ AND \\ ADD \\ JMP \\ ISZ \\ SAC \\ SRJ \end{bmatrix}$$

where each row, from top to bottom, is an encoding of the op-codes according to the sequence given above. This encoding allow to specify the op-codes by three bits. The price of breaking three groups corresponds to implement three additional product-terms in the Boolean cover.

MINIMAL BOOLEAN COVER

STATE	OPC	NO	N-ST	CONTROL SIGNALS
1100	•••	••	1010	000000001000000101
1010	•••	••	1110	000000000000000111
1110	•••	••	0000	000000000111011000
1000	•••	0•	0100	000000001010001000
0100	•01	••	0001	000000000000000000
0100	••0	••	0001	000000000000000000
0001	01•	••	0010	000000001010000001
0001	•00	••	0010	000000001010000001
•000	101	11	0100	000011101000100000
•000	••0	11	0100	000011101000100000
0000	•••	01	1000	000000000000000000
0001	•01	••	1010	000000001000000001
0000	••0	00	0001	000000000000000000
0001	110	••	0010	000000000010000000
0010	100	••	1010	010001000000000000
0010	010	••	1010	001001000000000000
0010	000	••	1010	001101000000000000
0010	110	••	0110	000101000001000010
0110	•••	••	1100	100101100000000010
0100	101	••	0000	000000100000000000
0••0	101	00	0001	000000100000000000
0000	001	•1	1000	000000000001000001
0100	011	•1	1010	000101010000000100
1000	0•1	11	0100	000011101000100100
0100	001	0•	0000	000101100000000110
0000	001	00	0100	000000000001000001
0000	011	00	1010	000001000000100100
0100	001	11	0000	000000100000000111
0000	011	11	1010	000011101000100100

The sequencing and control-store can be implemented by a PLA having 29 rows and 30 columns. Therefore this implementation requires a slightly (1.5%) larger area than the previous one but fewer inputs are needed.

4. CONCLUSIONS

We have presented a new technique for encoding PLA-based control-units. Control units, specified at the functional level by tables of mnemonic strings, are encoded into a Boolean representation that mi.iimizes the size of the control-store implemented by a PLA. The proposed method is based on symbolic minimization of the combinational component of the control unit and and on two related constrained encoding problems. Symbolic minimization yields a minimal sum-of-product representation of the next-state transition functions, independently of the encoding of the primary inputs and control-states. The first encoding problem is finding the minimum length encoding among those that minimize the number of product-terms of a PLA implementation. The second is finding a bounded-length encoding that minimize the number of product-terms of a PLA implementation. Minimal-area PLA implementations of the control-store can be found by trading-off the solution to these problems.

Two heuristic algorithms for solving the above problems have been presented. Both algorithms have been implemented in a computer program. We refer the interested reader to [DEMI84b] and [DEMI84c] for further details on the first algorithm, the computer program implementation and the experimental results.

5. ACKNOWLEDGMENTS

The author wish to thank Bob Brayton, Gary Ditlow, Curt McMullen, Richard Rudell, Alberto Sangiovanni-Vincentelli and Tiziano Villa for several helpful discussions.

REFERENCES

[AGER76] T. Agerwala "Microprogram Optimization: a Survey" IEEE Trans on Comput., vol. C-25, pp. 962-973, oct 1976.

[ANDR80] M. Andrews Principle of Firmware Engineering in Microprogram Control Computer Science Press, 1980.

[BRAY84] R.Brayton,G.D.Hachtel,C.McMullen and A.L.Sangiovanni-Vincentelli, "ESPRESSO-II: Logic Minimization Algorithms for VLSI Synthesis", in preparation.

[BROW81] D.W.Brown, "A State-Machine Synthesizer - SMS", Des. Autom. Conf., pp. 301-304, Nashville, jun. 1981.

[CLAR75] C.R.Clare, "Designing Logic Systems using State Machines", McGraw Hill, 1975.

[DAVI72] P. Davies "Readings in Microprogramming", IBM Jour. of Res. and Dev., vol. 11, No. 1, pp. 16-40, jan 1972.

[DEMI83c] G.De Micheli and A.L.Sangiovanni Vincentelli , "Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications", IEEE Trans. on Comput. Aided Des. of Int. Circ. vol. CAD-2, No. 3 pp. 167-180 jul. 1983.

[DEMI83d] G.De Micheli and M.Santomauro, "SMILE: A Computer Program for Partitioning of Programmed Logic Array", Computer Aided Design No. 2 pp. 89-97, mar. 1983 and Memorandum UCB/ERL No. 82/74.

[DEMI83f] G. De Micheli, A.Sangiovanni-Vincentelli and T.Villa, "Computer-Aided Synthesis of PLA-based Finite State Machines", Int. Conf. on Comp. Aid. Des., Santa Clara pp. 154-157, sep 1983.

[DEMI83g] G. De Micheli "Computer-Aided Synthesis of PLA-based Systems" Ph.D. Dissertation, University of California,Berkeley, 1983.

[DEMI84a] G.De Micheli, M.Hoffman, A.R.Newton and A.L.Sangiovanni Vincentelli, "A Design System for PLA-based Digital Circuits", Advances in Computer Engineering Design, Jai Press, 1984 (in print).

[DEMI84b] G.De Micheli, R.Brayton and A.L.Sangiovanni Vincentelli, "Optimal State Assignment for Finite State Machines", IBM Research Report RC 10599 and submitted for publication.

[DEMI84c] G.De Micheli, R.Brayton and A.L.Sangiovanni Vincentelli, "KISS: a Program for Optimal State Assignment of Finite State Machines", Int. Conf. on Comp. Aid. Des., Santa Clara, nov 1984.

[GRAS83] W. Grass "A Synthes System for PLA-Based Programmable Hardware" Microprocessing and Microprogramming No. 12 pp. 15-31 dec 1983.

[HACH82a] G.D.Hachtel,A.R.Newton and A.L.Sangiovanni Vincentelli, "An Algorithm for Optimal PLA Folding", IEEE Trans. on CAD of Int. Circ. and Syst. , pp. 63-77 vol. 1, No. 2, apr. 1982.

[HART66] J.Hartmanis and R.E.Stearns, "Algebraic Structure Theory of Sequential Machines", Prentice Hall, 1966.

[HONG74] S.J.Hong,R.G.Cain and D.L.Ostapko, "MINI: a Heuristic Approach for Logic Minimization", IBM Jour. of Res. and Dev., vol. 18, pp. 443-458, sep. 1974.

[LANG82] G.Langdon "Computer Design", Computech Press, 1982.

[LEWI81] T. Lewis and B. Shriver "Introduction to Special Issue on Microprogramming" IEEE Trans on Comput., vol. C-30, pp. 457-459, jul 1981.

[LOGU75] J.C.Logue N.F.Brickman F.Howley J.W.Jones and W.W.Wu, "Hardware Implementation of a Small System in Programmable Logic Arrays", IBM Jour. of Res. and Dev., vol. 19, pp. 110-119, mar. 1975.

[NEWT81] A.R.Newton, D.O.Pederson, A.L.Sangiovanni Vincentelli and C.H.Sequin, "Design Aids for VLSI: the Berkeley Perspective", IEEE Trans. on Circ. and Syst., vol. CAS 28 pp. 618-633 jul. 1981.

[PAPA79] C. Papachristou "A Scheme for Implementing Microprogram Addressing with Programmable Logic Arrays" Digital Processes No. 5 pp. 235-256 may 1979.

[SU72] S.Y.H.Su and P.T.Cheung, "Computer Minimization of Multi-Valued Switching Functions", IEEE Trans on Comput., vol 21, pp. 995-1003, 1972.